Automatische Vorhersage von Änderungsausbreitungen am Beispiel von Automatisierungssystemen

Master's Thesis von

Sandro Koch

an der Fakultät für Informatik
Institut für Programmstrukturen und Datenorganisation (IPD)

Erstgutachter:	Prof. Dr. rer. nat. Ralf H. Reussner
Zweitgutachter:	Jun.-Prof. Dr.-Ing. Anne Koziolek
Betreuende Mitarbeiterin:	Dipl.-Inform. Kiana Rostami
Zweiter betreuender Mitarbeiter:	Dr. rer. nat. Robert Heinrich

01. Dezember 2016 – 28. April 2017

Karlsruher Institut für Technologie
Fakultät für Informatik
Postfach 6980
76128 Karlsruhe

Ich versichere wahrheitsgemäß, die Arbeit selbstständig angefertigt, alle benutzten Hilfsmittel vollständig und genau angegeben und alles kenntlich gemacht zu haben, was aus Arbeiten anderer unverändert oder mit Änderungen entnommen wurde.

Karlsruhe, 28.04.2017

. . . . . .

(Sandro Koch)

Zusammenfassung

Das Karlsruher Architecture Maintainablity Prediction Framework (KAMP) bestimmt anhand der Architektur eines Systems, wie sich eine Änderung durch ein System propagiert. Zum aktuellen Zeitpunkt umfassen die Domänen, die mit KAMP analysiert werden können, die Softwaredomäne und Geschäftsprozesse. Diese Arbeit evaluiert die Möglichkeit, ob das KAMP-Framework derart erweitert werden kann, dass es auch in der Domäne der automatisierten Produktionssysteme (aPS) Einsatz finden wird. Auf der Basis einer bestehenden Produktionsanlage der technischen Universität München wird das Framework derart erweitert, dass zuvor definierte Änderungsszenarien evaluiert werden können. Bisher wurde die Änderungsausbreitung in aPS auf Hardwareebene betrachtet, entkoppelt von der darauf eingesetzten Software. Hier wird eine bisher nicht betrachtete Möglichkeit aufgezeigt, wie bei der Änderungsausbreitung sowohl Hardware als auch Software gemeinsam untersucht werden können. Dazu wird jeweils ein Metamodell für die Anlage, eines für die Software und eines für die Auslieferung der Software auf die Hardware vorgestellt. Darüber hinaus werden Ausbreitungsregeln herausgearbeitet, so dass die Änderungspropagation für die Hardware und Software eines aPS berechnet werden kann. Dadurch wird es möglich sein, die Änderungsausbreitung als ganzheitliches System zu betrachten und schlussendlich auch zu verwenden.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Liste der Codebeispiele

Liste der Algorithmen

1. Einleitung

Software durchdringt so gut wie jeden Bereich unseres alltäglichen Lebens. Wir werden geweckt durch einen elektronischen Wecker, lesen die aktuellsten Nachrichten auf einem Tablet und werden von unserem Kühlschrank darüber informiert, dass die Milch bald nachgekauft werden muss[30]. Auf dem Weg zur Arbeit telefonieren wir bereits mit den Kollegen über ein anstehendes Meeting und im Falle eines Falles bremst der Wagen selbstständig, um einen Auffahrunfall zu verhindern. Software wird aber nicht nur präsenter, auch die Anforderungen an selbige werden anspruchsvoller. Bestehende Software soll weiterverwendet werden, da meist kein unbeträchtlicher Aufwand in die Entwicklung der Programme investiert wurde. Also gilt es, weitere Funktionalität hinzuzufügen. Ein anderes Beispiel liegt vor, wenn aufgrund einer geänderten Gesetzeslage Anpassungen vorgenommen werden müssen. Auch sind Änderungen notwendig, wenn es darum geht in einem bestehenden System Fehler zu korrigieren oder geänderte Anforderungen der Kunden umzusetzen. Ist geklärt welche Anforderungen geändert wurden, muss bestimmt werden, welche Artefakte des Systems angepasst werden müssen. Dies dient der Aufwands- und Kostenschätzung und soll in erster Linie ermöglichen, das Risiko angemessen zu kalkulieren.

Aufgrund immer komplexer werdender Software ist die Ausbreitung einer Änderung innerhalb der Software keine triviale Aufgabe. Denn es gilt festzustellen, welche Artefakte innerhalb des Systems geändert werden müssen und wie sich die Änderung auf weitere Artefakte auswirkt und somit im Softwaresystem ausbreitet. Aber auch in anderen als der Softwaredomäne spielen Änderungen der Anforderungen und den daraus resultierenden Änderungen in einem System eine Rolle. Konkret wird auf die Vorhersage von Änderungsausbreitungen an automatisierten Produktionsanlagen eingegangen. Diese Anlagen sind in der Domäne der Automatisierungssysteme zu verorten. Zum einen bestehen derartige Systeme aus mechanischen und elektronischen Komponenten und zum anderen wird die Anlage mithilfe von Programmen gesteuert und geregelt. Somit muss bei der Änderungsausbreitung die Auswirkung auf die Hardware (Mechanik und Elektronik) und die Software, sowie deren Synergie betrachtet werden.

1.1. Grundbegriffe

Im Zuge dieser Arbeit werden die grundlegenden Begriffe aus den Domänen der Softwareentwicklung sowie der Automatisierungssysteme eingeführt, die zum Verständnis dieser Arbeit beitragen. Essentiell für die Einordnung der vorgestellten Vorgehensweise ist die Kenntnis über die modellgetriebene Softwareentwicklung (2.1.1). Zudem werden die verwendeten Werkzeuge vorgestellt (2.1.2), mit denen das Framework entwickelt wird und mit dem der Anwender das Framework verwendet. Der Begriff der Software Evolution

(2.1.3) wird eingeführt und aus den daraus gewonnenen Erkenntnissen werden parallelen zur Automatisierungstechnik gezogen. Außerdem wird das Zielsystem vorgestellt (2.2.2), das später der Evaluation dienen soll, ebenso wie die Zielsprache (2.2.3), die bei der Produktionsanlage eingesetzt wird. Denn die Art, wie Software für Automatisierungssysteme entwickelt wird, unterscheidet sich von der traditionellen Softwareentwicklung. In diesem Zusammenhang wird auf die europäische Norm eingegangen, auf der die Zielsprache basiert, zum anderen werden die Möglichkeiten vorgestellt, wie die Programme in der Zielsprache erstellt werden können. Dies begründet sich darin, dass sich die Art des Programmierens teilweise vom traditionellen Ansatz unterscheidet.

1.2. Änderungsvorhersage

Nachdem die Grundbegriffe behandelt wurden, wird das Problem der automatischen Vorhersage von Änderungsausbreitungen und deren Notwendigkeit vorgestellt (2.3.1). Dabei wird auf die Herausforderungen eingegangen, die sich aus dem Problem heraus ergeben. Außerdem wird der Nutzen skizziert, der eine gute automatische Vorhersage von Änderungsausbreitungen mit sich bringt. An einem Beispiel aus der Softwareentwicklung wird dargelegt, wie mit aktuellen Verfahren der Vorhersage von Änderungsausbreitungen in dieser Domäne umgegangen wird (2.3.2). Das vorgestellte Verfahren dient als Basis für die Erweiterung des Frameworks. Auf die Anpassung des Frameworks wird in Kapitel 5 eingegangen. Weiterhin werden auch mit dem in dieser Arbeit vorgestellten Ansatz verwandte Arbeiten behandelt. Die Ansätze sind in vier Klassen aufgeteilt. Verfahren zur aufgabenorientierten Projektplanung (3.1.1) bilden die erste Klasse, Verfahren zur architekturbasierten Projektplanung (3.1.2) die zweite Klasse. Die dritte Klasse besteht aus Verfahren zur architekturbasierten Software Evolution (3.1.3). Teil der letzten Klasse, der szenariobasierten Architekturanalyse (3.1.4), ist unter anderem das in dieser Arbeit vorgestellte Verfahren.

1.3. Umsetzung und Evaluation

Kern dieser Arbeit bildet die Untersuchung, ob die vorgestellten Verfahren aus der Softwareentwicklung in der Domäne der Automatisierungssysteme Anwendung finden können. Dazu werden die erstellten Metamodelle sowie die Ausbreitungsregeln und deren Umsetzung erörtert. Es gilt, ein Metamodell der Anlage (5.1) und des in der Automatisierungstechnik üblichen Softwarestandards (5.2) zu modellieren. Um eine Relation zwischen Hardware und Software herstellen zu können, wird das dafür notwendige Hardware-Software Korrelations-Modell vorgestellt (6). Auf die domänenspezifischen Metamodelle folgen die für das Framework notwendigen Metamodelle. Im Änderungsanfragen-Metamodell (7.1) werden die initial möglichen Modifikationen definiert und im Metamodell der Architekturmodellanreicherung (7.2) werden Artefakte und Relationen bestimmt, die nicht direkt Teil der Anlage sind.

Die Umsetzung der Ausbreitungsregeln wird in Kapitel 5 Abschnitt 8 gezeigt. Dabei werden drei Regeln vorgesellt, die jeweils auf der Änderung einer bestimmten Komponente im

System basieren. Ausgehend von der initialen Änderung wird die Änderungspropagation berechnet. Die Algorithmen werden beschrieben und der umfangreichste bezüglich der Komplexität analysiert.

Evaluiert wird die Umsetzung mittels vordefinierter Szenarien. Dazu werden zunächst die Szenarien vorgestellt (9.2). Dabei wird auf die jeweiligen Besonderheiten und mögliche Probleme eingegangen. Für jedes Szenario muss eine Evaluationsumgebung (9.3) erstellt werden. Dazu zählt ein Modell der Anlage, der darauf eingesetzten Software und der Hardware-Software Korrelation. Ebenso wird jeweils das Änderungsanfragen- und Anreicherungs-Metamodell vorgestellt. Abschließend wird für jedes Szenario, basierend auf den Modellen und den Ausbreitungsregeln die Änderungsausbreitung automatisiert berechnet und auf dieser Grundlage eine Aktivitäten-Liste erstellt. Das Ergebnis (9.4) wird mit einer manuell erstellten Liste verglichen und interpretiert.

2. Hintergrund

In diesem Kapitel wird die wissenschaftliche Grundlage vorgestellt, auf der diese Masterarbeit aufbaut. Zum einen wird auf die Frage eingegangen, was Änderungsausbreitung im Kontext der Softwareentwicklung bedeutet und wofür eine Vorhersage dieser sinnvoll ist. Zum anderen wird aufgezeigt, welchen Mehrwert die Vorhersage auch in anderen Domänen – in diesem konkreten Fall der Automatisierungstechnik – bietet. Daher soll analysiert werden, wie die automatische Vorhersage von Änderungsausbreitungen bei Automatisierungssystemen mithilfe bestehender Techniken aus der Softwareentwicklung bewältigt werden kann.

Zu einem wichtigen Werkzeug zählt hierbei die Modellierung von Softwaresystemen und die darauf aufbauende modellgetriebene Softwareentwicklung. Neben der strukturellen Erfassung eines Softwaresystems durch ein Modell zu Dokumentationszwecken kann eben dieses Modell herangezogen werden, um Änderungen und deren Ausbreitung zu analysieren. Nachdem der Stand der Technik in der Softwaredomäne betrachtet wurde, gilt es, den Zustand im Bereich der automatisierten Produktionssysteme zu bestimmen. Neben der Frage, wie momentan Änderungsausbreitung vorhergesagt werden kann, wird erörtert, ob und wie in diesem Bereich Modelle verwendet werden. Außerdem wird das Zielsystem, eine Produktionsanlage für Lehrzwecke, vorgestellt.

2.1. Grundlagen der Softwaredomäne

Um ein gemeinsames Verständnis zu schaffen, wird im Abschnitt der Grundlagen auf die in dieser Arbeit verwendeten Verfahren und Werkzeuge eingegangen. Dabei liegt der Fokus sowohl auf der Sicht eines Softwareentwicklers als auch auf der eines Maschinenbauers. Aus diesem Grund wird zuerst die modellgetriebene Softwareentwicklung (2.1.1), die im Framework verwendeten Werkzeuge (2.1.2) sowie das Prinzip der Software Evolution (2.1.3) vorgestellt. Darauf wird auf die Grundlagen der automatisierten Produktionssysteme (2.2) und das in diesem Feld verwendete Modellierungswerkzeug AutomationML (2.2.1) eingegangen. Außerdem wird das Zielsystem xPPU (2.2.2) sowie der Standard IEC 61131-3 (2.2.3) vorgestellt.

2.1.1. Modellgetriebene Softwareentwicklung

In diesem Abschnitt wird der Begriff der modellgetriebenen Softwareentwicklung, kurz MDSD, eingeführt. Dabei wird die Grundidee der MDSD vorgestellt und welche Ziele im Rahmen der Softwareentwicklung damit zu erreichen versucht werden. Dabei wird auf den Begriff „modellgetrieben“ genauer eingegangen und es werden die Unterschiede zur modellbasierten Softwareentwicklung aufgezeigt. Auch die mit der modellgetriebenen

Softwareentwicklung verbundenen Vorteile werden erörtert, sowie die Herausforderungen, die auftreten, wenn ein modellgetriebener Ansatz verfolgt wird.

2.1.1.1. Idee der MDSD

Laut Stahl et al. [66] ist MDSD eine Sammlung von Techniken, die aus formalen Modellen automatisiert lauffähige Software erzeugt. Formale Modelle in diesem Kontext sind nicht beliebige Modelle, wie sie oft in der Softwareentwicklung eingesetzt werden. Dazu zählen Skizzen, die auf Papier erstellt werden, oder auch UML-Modelle, die zu Dokumentationszwecken einen Teil der Software darstellen. In der MDSD ist dieser Begriff deutlich enger gefasst, dort ist ein Modell „formal“, wenn es einen Aspekt der Software vollständig beschreibt. Ziel ist nicht, ein abzubildendes Modell ganzheitlich darzustellen, sondern dass die Aspekte, die für die Funktionalität notwendig sind, vollständig beschrieben werden. Nicht zwangsläufig sind Modelle auf eine grafische Darstellung mittels der UML Notation beschränkt. Diese kann bei Zeiten sehr sperrig und zeitaufwändig sein. Daher ist es ebenfalls möglich, Modelle in textueller Form darzustellen.

Wie bereits beschrieben, ist es in der modellgetriebenen Softwareentwicklung notwendig, dass aus formalen Modellen lauffähige Software erzeugt wird. Wenn formale Modelle eingesetzt werden und diese ausschließlich zur Dokumentation bzw. als Spezifikation verwendet werden, ist der verfolgte Ansatz nicht modellgetrieben. Die Software muss dadurch noch immer manuell implementiert werden. Per Definition durch Stahl et al. wird erst ein Aspekt des MDSD Ansatzes umgesetzt, nämlich der der formalen Modelle. Nach [66] gibt es zwei Arten, wie aus einem formalen Modell automatisiert Software erzeugt werden kann. Zum einen sind es Generatoren: diese erzeugen aus einem Modell Quelltext in einer anderen Programmiersprache. Aus dem Quelltext kann das entsprechende Programm kompiliert werden. Zum anderen gibt es Interpreter, die zur Laufzeit ein Modell einlesen und aufgrund dessen entsprechende Aktionen ausführen. Dadurch, dass bei beiden Verfahren Modelle ausführbar gemacht werden, ist gezeigt, dass die verwendeten Modelle klar definiert sein müssen. Ansonsten ist es nicht möglich, die Software auszuführen oder die Lauffähigkeit der Software aufgrund fehlerhafter Sektoren zu gewährleisten.

Wie in [66] beschrieben, entspricht es noch nicht der modellgetriebenen Softwareentwicklung, wenn aus den Modellen einmal der Quelltext generiert wurde, daraufhin aber die Software entkoppelt vom Modell weiterentwickelt wird. Denn wenn der Quelltext einmalig erzeugt wird und dieser dann manuell weiterentwickelt wird, führt das dazu, dass Änderungen später im Quelltext vorgenommen werden und das Modell somit obsolet wird. Ziel ist es, dass der erzeugte Quelltext nur einen Zwischenschritt darstellt, der notwendig ist, um aus einem Modell lauffähige Software zu erzeugen. Damit ist gewährleistet, dass das Modell mit dem Quelltext immer konsistent ist und es somit den aktuellen Stand widerspiegelt. Dadurch ergibt sich automatisch eine immer aktuelle Dokumentation in Form der Modelle. Doch nicht der ganze Quelltext wird automatisch zu erzeugen sein, aber ein Großteil der für Entwickler repetitiven und fehleranfälligen Elemente können in den Generator ausgelagert werden.

2.1.1.2. Modell

Im Folgenden wird aufgezeigt, was ein Modell ist und wie der modellgetriebene Ansatz in den Grundzügen funktioniert. Nach Stachowiak[65] ist ein Modell über drei Merkmale definiert. Das Abbildungsmerkmal sagt aus, dass das Modell das Original abbildet. Das Original kann ebenfalls ein Modell sein. Das Verkürzungsmerkmal bedeutet, dass nicht alle Merkmale im Modell abgebildet werden. Und das letzte Merkmal, das pragmatische Merkmal bestimmt, dass für die Zielgruppe des Modells relevante Informationen modelliert werden. Dadurch ist gewährleistet, dass ein Modell für die Zielgruppe alle nötigen Informationen enthält und die unwichtigen Details entweder nicht dargestellt werden oder nicht enthalten sind.

Diese Art der Abstraktion erlaubt es, einzelne Elemente zu größeren Bausteinen zusammenzufassen. Das ermöglicht es, die Komplexität aufzuteilen, zum einen vom erstellten Modell auf das Zielsystem und zum anderen das Modellieren selbst. So werden die Aufgaben zwischen einem Domänenexperten und einem Technologieexperten differenziert[66]. Im Falle der Modellierung eines Metamodells für ein automatisiertes Produktionssystem liefert der Domänenexperte das domänenspezifische Wissen das nötig ist, um ein derartiges System zu planen und zu bauen. Für den mechanischen Teil könnte das ein Maschinenbauer und für den elektrischen Teil ein Elektrotechniker sein. Zusammen mit dem Technologieexperten, der in der Lage ist, die nötigen Werkzeuge zu handhaben, die für die Erstellung von Modellen, domänenspezifischen Sprachen und Editoren nötig sind[66].

2.1.1.3. Metamodell

Ziel von MDSD ist es, die Inhalte einer Domäne formal zu beschreiben. Dazu muss die Struktur der Domäne verstanden werden, damit diese formalisiert werden kann. Meist ist dazu ein Domänenexperte notwendig, der die zu modellierende Domäne versteht und somit beschreiben kann. Die Beschreibung der Domäne wird Metamodell genannt. Als Beispiel sei hier der UML-Standard genannt. Hier sind die einzelnen Elemente der Sprache in einem Metamodell definiert[66]. So ist im Metamodell des UML-Klassendiagramms definiert, was eine Klasse und eine Relation ist. Streng genommen ist das Metamodell deutlich komplexer. Das hier genannte Beispiel soll zunächst eine Idee davon vermitteln. Für mehr Details sei auf den OMG-Standard „OMG Unified Modeling Language“ verwiesen[73].

2.1.1.4. Vorteile von MDSD

Das Ziel der modellgetriebenen Softwareentwicklung wird in [66] wie folgt definiert. MDSD dient dazu, die Softwarequalität zu verbessern und die Wiederverwendbarkeit zu erhöhen. Auch soll die Entwicklungseffizienz gesteigert werden. Die Qualität ist beim MDSD Ansatz abhängig vom Modell und der Transformation in die jeweilige Programmiersprache. Da die Transformation immer das gleiche Ergebnis liefert, ist an dieser Stelle die Qualität gleichbleibend. Somit spielt nur noch die Qualität des Modells eine Rolle. Wenn hingegen der Quelltext manuell auf Basis eines Modells erstellt wird, ist die Qualität abhängig von der Kompetenz und Erfahrung des jeweiligen Entwicklers. Zusätzlich ermöglicht der modellgetriebene Ansatz den Entwicklern, besser mit sehr komplexen Software-Infrastrukturen umzugehen.

Dadurch, dass nicht mehr direkt auf Quelltextebene gearbeitet werden muss, erfolgt die Programmierung auf einer höheren Abstraktionsebene. Somit wird die Gesamtkomplexität aufgeteilt auf die Abbildung des Modells sowie auf die Zielsprache und der im Modell darzustellenden Komplexität. Diese Abstraktion ermöglicht es, besser mit der Komplexität umzugehen, da diese in Teilkomplexitäten aufgetrennt wird[66].

Auf langfristige Sicht bietet MDSD den Vorteil, dass auch bei einem gewachsenen und bereits veränderten System, die Dokumentation anhand der Modelle noch konform ist zur tatsächlichen Implementierung. Kurzfristig gesehen, also zum Beginn des Lebenszyklus des Systems, ist der Aufwand größer. Es müssen die Modelle und Generatoren bzw. Interpreter zunächst entworfen und entwickelt werden. Auch wenn auf bereits bestehende Systeme zurückgegriffen werden kann, ist die Einstiegshürde doch noch größer als bei der klassischen Softwareentwicklung. Da der Lebenszyklus solcher Systeme häufig einige Jahrzehnte andauern kann, mag der anfängliche Aufwand gerechtfertigt sein[66]. Mehr zum Lebenszyklus und der Evolution von Software folgt im Abschnitt 2.1.3 Software Evolution.

Laut [66] ist die Wiederverwendung von Teilen der Modelle und Generatoren sowie Interpretern, zumindest bei Produktlinien zur Herstellung von Softwaresystemen, gegeben. Auch ist die Plattformunabhängigkeit theoretisch gegeben, aber sie ist immer noch mit einem deutlichen Aufwand verbunden. Es ist zwar möglich, aber ob der Einsatz in einem Projekt gerechtfertigt ist, muss von Fall zu Fall entschieden werden.

2.1.2. Eclipse Modeling Framework

Dieser Abschnitt behandelt die Werkzeuge, die verwendet werden, um das KAMP- Framework zu erweitern, und die notwendig sind, selbiges zu nutzen. Als Basis für die Entwicklung dient die integrierte Entwicklungsumgebung (IDE) Eclipse sowie deren Erweiterung Eclipse Modeling Framework (EMF). Das Framework selbst ist als eine Erweiterung für Eclipse realisiert. Das bedeutet, dass für die Entwicklung sowie die Nutzung dieselbe Basis zum Einsatz kommt. Weitere Werkzeuge wie Versionsverwaltung in Form von Subversion (SVN)[20] und Git[17] seien hier nur am Rande erwähnt. Beide Systeme können direkt von Eclipse aus als Erweiterungen installiert werden.

2.1.2.1. Eclipse

Das Eclipse Projekt der Eclipse Foundation ist ein Open Source Projekt mit dem Ziel, eine offene Entwicklungsplattform bereitzustellen, die auf erweiterbaren Frameworks, Werkzeugen und Laufzeitumgebungen aufsetzt. Diese sollen es ermöglichen, Software zu entwickeln, zu verteilen und zu verwalten. Laut [33] wurde das Projekt im November 2001 durch IBM initiiert. Im Jahr 2004 wurde die Eclipse Foundation gegründet, mit dem Ziel eine herstellerunabhängige Entwicklung des Eclipse Projektes zu gewährleisten. Aufgrund der Erweiterbarkeit der Eclipse IDE ist es möglich, neben Java und Java EE Anwendungen auch Programme für andere Programmiersprachen und sogar andere Plattformen zu entwickeln.

So ist es beispielsweise möglich, mit Eclipse C/C++ Programme zu entwickeln. Aber auch PHP Anwendungen und Apps für das Smartphone Betriebssystem Android lassen sich mit Eclipse realisieren. Für all die genannten Möglichkeiten kann auf der Homepage des Eclipse Projektes eine dedizierte IDE heruntergeladen werden, die alle nötigen Plug-Ins enthält, um direkt mit der Entwicklung beginnen zu können. Aber es ist auch möglich, eine Standard Eclipse IDE herunterzuladen und die nötigen Plug-Ins selbst zu installieren. Dies kann über das Hinzufügen von Quellen der Erweiterungen geschehen oder komfortabel über den Eclipse Marketplace. Auch das EMF kann direkt von der Eclipse Homepage bezogen werden. Dort sind in der fertigen IDE alle nötigen Plug-Ins für die Entwicklung enthalten.

2.1.2.2. EMF

Das Eclipse Modeling Framework erweitert die Eclipse IDE dahingehend, dass die IDE nun Ecore Modelle erstellen, bearbeiten und darstellen kann. Dazu werden Werkzeuge bereitgestellt, die es ermöglichen, Modelle auf Ecore-Basis zu erstellen. Ziel ist es, den modellgetriebenen Ansatz in den Softwareentwicklungsprozess zu integrieren und Modelle als zentrale Elemente zu verwenden[69]. EMF bietet verschiedene Möglichkeiten, Metamodelle zu erstellen und zu bearbeiten. Eine davon ist es, sie in einem Baumeditor zu bearbeiten. EMF ermöglicht es, aus den erstellten Metamodellen Eclipse Plug-Ins zu generieren. Die Plug-Ins stellen einen einheitlichen Baumeditor zur Verfügung, mit dem es möglich ist, ein Modell auf der Basis des Metamodells zu erstellen. Der Quelltext für das Plug-In wird über eine von EMF bereitgestellte Transformation generiert. Neben der Unterstützung noch weiterer Möglichkeiten der Darstellung von Metamodellen, werden auch weitere Aspekte der modellgetriebenen Softwareentwicklung bereitgestellt. Im Rahmen dieser Arbeit spielt aber nur das Generieren der Plug-Ins eine Rolle. Daher wird auf weitere Möglichkeiten wie ATL[35], QVT[50] oder auch das Erzeugen von domänenspezifischen Sprachen (DSL) nicht weiter eingegangen.

2.1.3. Evolution in der Softwareentwicklung

Das Problem alternder Software ist bereits seit den 90er Jahren bekannt und Teil der Forschung[47]. Das mag auf den ersten Blick unsinnig erscheinen, da Software ein immaterielles Produkt ist und somit z.B. nicht von Abnutzung betroffen ist. Laut [47] kann der Alterungsprozess von Software mit dem des Menschen verglichen werden. Der Alterungsprozess ist unvermeidlich, aber im Gegensatz zu dem des Menschen lässt er sich hinauszögern und in seltenen Fällen auch invertieren. Es wurden Methoden entwickelt, Software über Refaktorisierung zu verjüngen[32]. Daraus lässt sich schließen, dass der Lebenszyklus von Software verlängert wird[2]. Software Evolution befasst sich mit der Tatsache, dass Software immer länger eingesetzt wird, und den Möglichkeiten, die negativen Auswirkungen des Alterungsprozesses zu vermeiden. Dieser Abschnitt führt den Begriff der Software Evolution ein und zeigt auf, wie sie im Zusammenhang zur Änderungsausbreitung steht.

2.1.3.1. Der Grund für den Alterungsprozess

Wie die ersten zwei Regeln der „Lehman's Laws" aussagen, ist Software kontinuierlichem Wandel unterzogen (I). Das bedeutet, Software die nicht angepasst wird, ist immer weniger zufriedenstellend. Software wird auch immer komplexer (II), wenn nichts dagegen unternommen wird[42]. Daraus folgt, dass ein Softwaresystem, das am Markt bestehen will, kontinuierlich angepasst werden muss, um den Anforderungen der Zielgruppe zu entsprechen. Im Umkehrschluss führt das zu einer steigenden Komplexität des Systems. Ein bestehendes System, das bereits einige dieser Änderungszyklen erfahren hat, wird Legacy System genannt. Wenn das System nicht auf Erweiterbarkeit ausgelegt oder den Entwicklern nicht bekannt ist, wie das bestehende System zu erweitern ist, sinkt die Gesamtqualität. Gründe für die Unkenntnis kann unzureichende Dokumentation sein oder bei älteren Systemen, die Komplexität die kein Entwickler im Ganzen verstehen kann. Unabhängig der Gründe ist das Resultat stets dasselbe: Der Code korrodiert und der Aufwand, Änderungen zu implementieren, wird immer größer. Das kann schlussendlich dazu führen, dass ein Legacy System nicht mehr angepasst wird, da der Aufwand zu groß wäre, die Änderungen einzupflegen. Robert C. Martin schildert in seinem Buch „Clean Code" einen Fall, in dem dieser Umstand zur Insolvenz eines Unternehmens geführt hat[44]. Im nachfolgenden Abschnitt werden die Auswirkungen skizziert, die aus gealterter Software resultieren.

2.1.3.2. Auswirkungen gealterter Software

Der Alterungsprozess hat auf unterschiedliche Bereiche der Software negative Auswirkungen. Die Auswirkungen können den Benutzer beeinträchtigen. Dazu zählen im wesentlichen erhöhte Ressourcenaufwände[16, 58], reduzierte Verfügbarkeiten[3, 52] und eine steigende Zahl an Fehlern im System[71, 55]. Aber auch das Unternehmen, das die Software produziert, ist von den Auswirkungen alternder Software betroffen. Wie im vorherigen Abschnitt bereits beschrieben, werden die Entwicklungszyklen neuer Features länger, aber auch das Beheben von Fehlern bedarf immer mehr Zeit. Somit nimmt die Codequalität mit der Zeit ab, bis das System nicht mehr wartbar und/oder erweiterbar ist.

2.1.3.3. Änderungsausbreitung

Die vorliegende Arbeit befasst sich mit der Ausbreitungsanalyse von Änderungen in einem System. Der bisher beschriebene Sachverhalt bezieht sich zwar auf Software Systeme, kann aber auf automatisierte Produktionssysteme erweitert werden. Zumindest in dem Sinne, dass sich die Anforderungen auch bei einem Produktionssystem während dessen Lebenszyklus ändern können. Änderungen resultieren aus geänderten oder auch neuen Anforderungen an ein System. Die Vorhersage der Änderungsausbreitung dient nicht dazu, die Korrosion des Systems zu verlangsamen oder zu verhindern. Vielmehr bietet sie die Möglichkeit, die Auswirkungen auf das System zu analysieren und somit eine Basis zu errichten, an der eine Risikoanalyse für die Anforderungen ansetzen kann. Im folgenden Abschnitt wird aufgezeigt, dass die Kriterien der Evolution wie sie hier dargestellt wurden, auch Anwendung bei Automatisierungssystemen finden.

2.2. Grundlagen automatisierter Produktionssysteme

Wie auch Software Systeme haben automatisierte Produktionssysteme bestimmte Eigenheiten, die es bei der Evolution zu beachten gilt bzw. es nötig machen, sich mit dem Problem der Evolution auseinanderzusetzen. Automatisierte Produktionssysteme können über einen Zeitraum von Jahrzehnten eingesetzt werden. Im Laufe des Lebenszyklus eines solchen Systems, werden sich die Anforderungen daran unweigerlich ändern[23]. Unter die Änderungen fallen Erweiterungen, Korrekturen und Reparaturen. Reparaturen deshalb, weil Ersatzteile nicht unbedingt über den ganzen Lebenszyklus erhältlich sein müssen und dadurch die Notwendigkeit entsteht, neue Komponenten in das System zu integrieren[80]. Neben den bereits erwähnten mechanischen Teilen, die aufgrund eines Defektes oder auch wegen Verschleiß ausgetauscht werden müssen, können noch nachfolgende Elemente von Änderungen betroffen sein. Beispielhaft dazu zeigt Abbildung 2.1 die Bauteilklassen eines aPS.

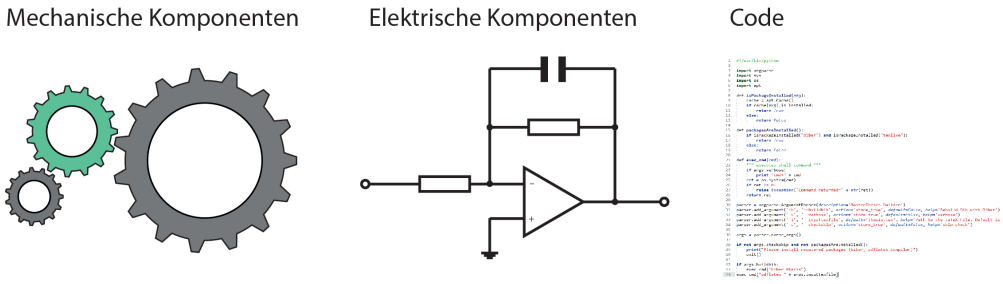

Abbildung 2.1.: Elemente eines automatischen Produktionssystems [77]

Fertige Komponenten sind nicht von Defekten ausgeschlossen, wobei das Änderungsintervall von elektrischen Komponenten mit ca. zehn bis 15 Jahren eine relativ große Zeitspanne aufweist. Im Vergleich dazu sind die mechanischen Teile mit einem Änderungsintervall von 20 bis 40 Jahren noch deutlich länger im Einsatz. Ein ähnliches Intervall wie die elektrischen Bauteile hat die Automatisierungs-Hardware vorzuweisen. Software ist am häufigsten von Änderungen betroffen, dort liegt das Intervall bei einmal wöchentlich bis einmal pro Jahr. Abbildung 2.2 stellt das Verhältnis der Änderungsintervalle noch einmal grafisch dar.

Im Gegensatz zu den physischen Komponenten kann Software meist zur Laufzeit aktualisiert werden. Indessen muss ein aPS gestoppt werden, bevor ein Austausch der Hardware stattfinden kann[80]. Um die Zeit, in der eine Maschine stillsteht, besser planen zu können oder ganz zu vermeiden, ist die Änderungsausbreitungsanalyse notwendig. Dazu wird ein Modell der Anlage benötigt. Darum wird im folgenden Abschnitt AutomationML vorgestellt, ein Standard, der darauf ausgelegt ist, Automatisierungssysteme zu modellieren.

2.2.1. AutomationML

In Zeiten von immer umfassender werdenden Projekten[66] und den Anforderungen, die das Konzept Industrie 4.0 mit sich bringt[24, 64, 25, 51], werden Werkzeuge benötigt die es ermöglichen, die Herausforderungen zu bewältigen. In der Softwareentwicklung

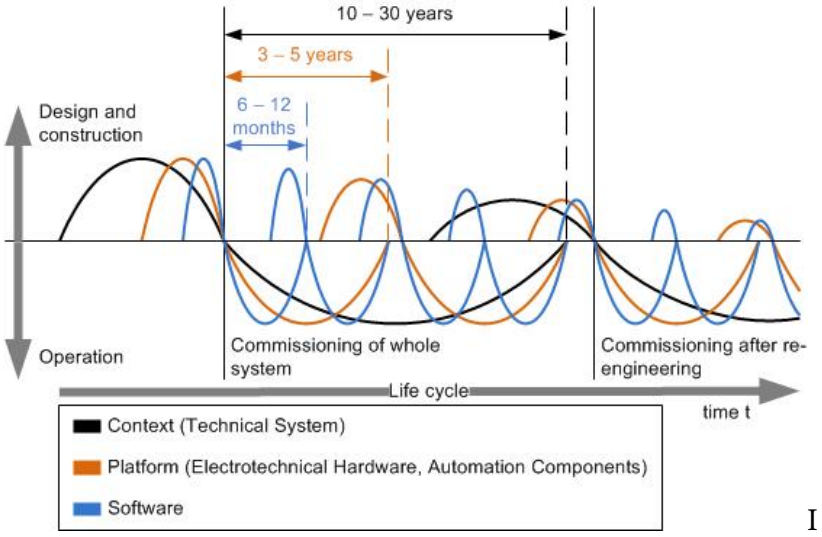

Abbildung 2.2.: Evolutionszyklen [43]

ist die Abstraktion das geeignete Mittel, um die Komplexität zu reduzieren. Als Beispiel seien hier Hochsprachen wie C und Haskell genannt. Sie wurden entwickelt damit Entwickler sich nicht mit der spezifischen Assemblersprache eines Chips befassen müssen und maschinenunabhängigen Code schreiben können. Die Abstraktion ermöglicht es, ein Programm zu schreiben und den Chip erst bei der Kompilierung festzulegen. Dadurch, dass die Plattform erst später festgelegt werden kann, ist es möglich, Programme, die in einer Hochsprache verfasst wurden, einfacher auf anderen Plattformen wiederzuverwenden. Der nächste Schritt hin zu einer noch umfassenderen Abstraktion ist die modellgetriebene Softwareentwicklung. Der modellgetriebene Ansatz ist zum gegenwärtigen Zeitpunkt noch Gegenstand aktueller Forschung, dennoch sind ihre Vorteile bereits vielversprechend. Mehr zur modellgetriebenen Softwareentwicklung und den Begriffsdefinitionen von Modell und Metamodell wurde in Abschnitt 2.1.1 thematisiert.

AutomationML nutzt den modellgetriebenen Ansatz, um Anlagen zu planen. Durch die bessere Beherrschbarkeit der Komplexität[40, 66] und die höhere Entwicklungsgeschwindigkeit[66] durch die Verwendung von Modellen soll sowohl die Qualität gewährleistet als auch die Kosten der Planung von Anlagen reduziert werden[24]. Die Planung selbst ist in Phasen unterteilt[56, 24]. Drath beschreibt vier Planungsphasen, die in Abbildung 2.3 skizziert werden.

Nicht nur die Kosten können durch die bisher genannten Punkte reduziert werden. Auch ist es dadurch besser möglich, einzelne Phasenabläufe zu optimieren und die Interphasenkopplung, welche oft fließend verläuft, zu verbessern[24]. Anwendung findet AutomationML im Bereich des Ingenieurwesens[56, 24] und wird als Austauschformat in der Anlagenentwicklung[56] sowie als Modellierungswerkzeug[24] eingesetzt. AutomationML ist in der IEC 62714 standardisiert, wovon die Teile *Architecture and general requirements* und *Role class libraries* zum internationalen Standard gehören[7]. Der Standard ermöglicht einen offenen, reibungslosen Datenaustausch zwischen den Planungsphasen. Im Gegensatz dazu existieren spezialisierte Insellösungen[24], die in dem jeweils vorgesehenen Einsatz-

Abbildung 2.3.: Planungsphasen von automatisierten Produktionsanlagen [24]

zweck eine bessere Lösung bieten. Dafür sind sie in der Interprozesskopplung schwerer zu handhaben, da nicht immer alle Standards erfüllt werden und somit Anpassungen nötig werden. Als Beispiel sei hier dem Standard IEC 61131-3 vorweggegriffen, bei dem der Standard genügend Interpretationsspielraum lässt, so dass Programme, die mit den Tools verschiedener Hersteller erstellt wurden, nicht immer kompatibel sind. In Abschnitt 2.2.3 wird die Thematik vertieft.

2.2.2. xPPU - Extended Pick and Place Unit

Die Pick and Place Unit (PPU) ist eine Prozessanlage für Herstellungsprozesse[78]. An der Technischen Universität München (TUM) am Institut für Automatisierung und Informationssysteme wird die PPU seit 2001 zu Lehr- und Forschungszwecken verwendet. Mithilfe der Anlage soll die Evolution von Prozessanlagen und die Auswirkung von Änderungen erforscht werden[72]. Beschrieben werden 16 Szenarien die Änderungen in den Bereichen der Mechanik, Automatisierungshardware und Software behandeln[41]. Abbildung 2.4 zeigt die erweiterte PPU (xPPU). Im Folgenden wird weiter auf die xPPU eingegangen und die Struktur der Anlage näher betrachtet.

2.2.2.1. Aufbau der xPPU

Eine Anlage selbst besteht aus s.g. Strukturen, die Anzahl ist abhängig vom Aufbau und Einsatzzweck der Anlage. Die xPPU hat eine bestimmte Anzahl an Strukturen (genauere Informationen finden sich in [72]). Strukturen bezeichnen funktionale Gruppen, die aus Modulen und/oder Komponenten bestehen, also nicht als einzelnes Bauteil betrachtet werden. Zu Strukturen zählen z.B. das Control Cabinet, das Communication Network und das Power Network. Module wiederum sind von den Entwicklern selbst zusammengestellte oder entwickelte Komponenten. Ein Modul kann aus weiteren Modulen zusammengesetzt

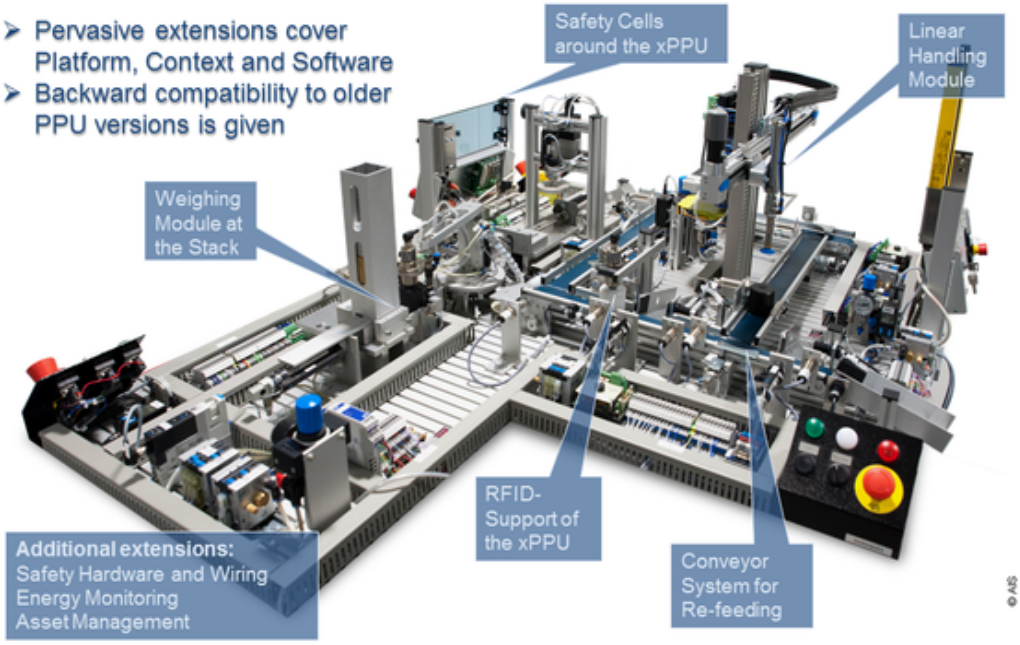

Abbildung 2.4.: Die xPPU [72]

sein. Im Gegensatz zu den Modulen stehen die Komponenten, die fertig eingekauft werden und nicht selbst entwickelt wurden. Für die Änderungsvorhersage bedeutet es, dass eine Komponente entweder getauscht wird, oder bestehen bleibt, aber nicht nur teilweise getauscht oder repariert werden kann. Daneben verfügen Komponenten über Schnittstellen, die definieren, wie Komponenten in die Anlage eingebunden werden. Abbildung 2.5 zeigt den geschilderten Aufbau der xPPU.

2.2.2.2. Metamodell der xPPU

Das von der TUM bereitgestellte Metamodell der xPPU wurde in AutomationML modelliert. Aufgrund bestimmter Eigenheiten im Modell bestand die Notwendigkeit ein KAMP-kompatibles Modell in Ecore zu erstellen.

Wie im Abschnitt 2.2.1 aufgezeigt, ist AutomationML spezialisiert auf die Modellierung von Automatisierungsanlagen. Dadurch ergeben sich Differenzen zur Modellierung in der Softwaretechnik. Als Beispiel sei hier die Änderung eines Datentyps genannt. Ändert sich ein Datentyp innerhalb einer Software, sind davon alle Elemente betroffen, die diesen Datentypen verwenden. Die Änderung geschieht auf der Metamodellebene. Dadurch ist es möglich, eine Regel zu definieren, die alle Vorkommnisse dieses Datentyps als Änderung deklariert. Ein Datentyp in der Automatisierungswelt kann z. B. eine Bus Box oder ein Zylinder sein. Ändert sich nun eine Bus Box, sind nicht alle Bus Boxen betroffen, sondern nur diese eine. Somit wandert die Änderung von der Modellebene auf die Instanzebene. Die daraus resultierenden Auswirkungen gilt es im weiteren Verlauf zu berücksichtigen. Im Abschnitt 5.1 wird auf die Details des Modells und des daraus resultierenden Metamodells eingegangen.

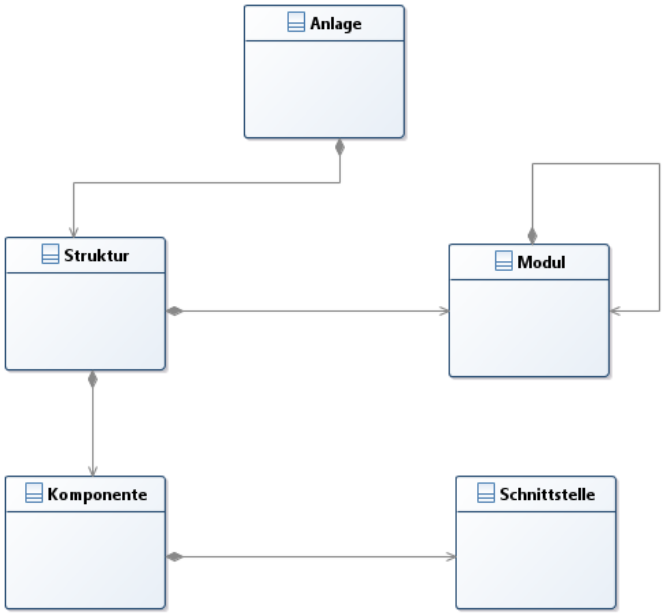

Abbildung 2.5.: Schematische Struktur der xPPU [77]

2.2.3. IEC 61131-3

Die internationale Norm IEC 61131 und die darauf basierende europäische Norm EN 61131 befassen sich mit den Grundlagen speicherprogrammierbarer Steuerungen (SPS)[26]. Die Norm ist in sechs Themengebiete aufgeteilt. Für diese Arbeit ist besonders der dritte Teil, „Teil 3: Programmiersprachen“[26] relevant. Der Standard kann im Webstore[1] der International Electrotechnical Commission für 350 CHF (330 Euro) bezogen werden. Aufgrund des hohen Preises kann auch auf das Buch „IEC 61131-3: Programming Industrial Automation Systems“[34] von John und Tiegelkamp Bezug genommen werden. Es stellt eine umfassende Beschreibung des Standards bereit. Laut [34] versteht sich die Norm eher als Leitfaden denn als eine strikte Sammlung von Regeln. Das hat zur Folge, dass aufgrund des hohen Umfangs des Standards die Hersteller nur Teile davon umsetzen. Wenn die Hersteller standardkonform sein wollen, müssen sie dokumentieren, an welchen Stellen ihre Umsetzung dem Standard entspricht und an welchen nicht. Das wird durch 62 Merkmalstabellen erreicht, in denen Anforderungen aufgelistet sind. Der Hersteller hat zu jedem Eintrag eine Anmerkung zu schreiben, ob die Anforderung ganz (fullfilled), teilweise (the following parts are fullfilled:...) oder nicht (not fullfilled) vorhanden ist[34].

Diese Freiheit, die den Herstellern dadurch gewährt wird, führt zu einer Fragmentierung der Implementierungen. Bei der xPPU werden Umsetzungen der Firmen CODESYS[2] sowie TwinCAT[3] eingesetzt. Grundlage unseres ersten Prototyps bilden die von der Firma CODESYS bereitgestellten Metamodelle ihrer Implementierung. Die Implementierung wird in [31] dargestellt.

[1] https://webstore.iec.ch/publication/4552
[2] https://www.codesys.com/
[3] http://www.beckhoff.de/default.asp?twincat/default.htm

Die Programmierung erfolgt durch so genannte Program Operation Units, kurz POUs. Sie entsprechen den Programm-, Organisations-, Sequenz-, und Funktionsblöcken der konventionellen SPS- Programmierung. POUs sind unterteilt in Funktion, Funktionsblock und Programm. Im Folgenden werden diese drei Sprachelemente näher erläutert.

2.2.3.1. Programm

Die POU Programm kann mit einem Hauptprogramm, z. B. der Main-Funktion in C, gleichgesetzt werden. Darin werden alle Variablen, die mit physischen Adressen verbunden sind deklariert. Dazu zählt z. B. die Ein- und Ausgabe der SPS. In allen anderen Belangen ist die POU „Program“ mit einem Funktionsblock identisch[34].

2.2.3.2. Funktionsblock

Ein Funktionsblock ist eine POU, der ebenfalls Parameter zugewiesen werden können. Im Gegensatz zur Funktion verfügt diese POU über statische Variablen. Dadurch können Zähler und Zeitgeber realisiert werden. Wenn ein Funktionsblock mehrfach mit den gleichen Parametern aufgerufen wird, ist die Ausgabe von dem Zustand der internen sowie externen Variablen abhängig. Diese bleiben zwischen den Aufrufen bestehen[34].

2.2.3.3. Funktion

Eine Funktion ist eine POU, der Parameter zugewiesen werden können. Sie hat keine statischen Variablen und somit auch keine Möglichkeit, Zustände zu speichern. Das hat zur Folge, dass eine Funktion die mehrfach mit den gleichen Parametern aufgerufen wird, immer die gleiche Ausgabe erzeugt[34].

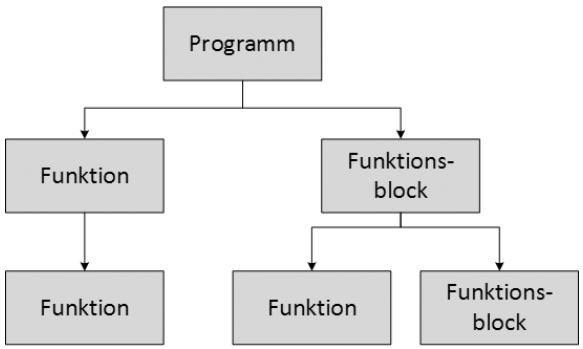

Abbildung 2.6.: Aufrufhierarchie der POUs [34]

Im Folgenden werden die Programmiersprachen aufzeigt, mit denen sich Programme für eine SPS nach der IEC 61131-3 Norm programmieren lassen. Zuvor aber noch eine Anmerkung zur Aufrufhierarchie von POUs. Wie in Abb. 2.6 zu sehen ist, kann die POU Programm, Funktionen, und Funktionsblöcke aufrufen. Funktionsblöcke haben Zugriff auf weitere Funktionsblöcke sowie Funktionen. Funktionen wiederum können nur weitere Funktionen aufrufen. Rekursion ist nicht möglich, POUs dürfen sich nicht selbst aufrufen[34].

Name	Abkürzung	Typ
Kontaktplan	KOP	grafisch, angelehnt an Stromlaufpläne
Anweisungsliste	AWL	textuell, verknüpft Ein- und Ausgänge
Funktionsplan	FUP	grafisch, Visualisierung von Funktionsblöcken
Ablaufsprache	AS	grafisch, Ablaufdiagramm
Strukturierter Text	ST	textuell

Tabelle 2.1.: Programmiersprachen des IEC 61131-3 [34]

Programme lassen sich durch fünf Programmiersprachen erstellen. Tabelle 2.1 gibt eine Übersicht über die in IEC 61131-3 definierten Programmiersprachen. Neben der Darstellung, sei es nun grafisch oder textuell, unterscheiden sich die Sprachen auch in ihrer Mächtigkeit.

2.2.3.4. Kontaktplan - KOP

Wie bereits erwähnt, orientiert sich der Kontaktplan an den Stromlaufplänen aus der Elektrotechnik. Dabei wird der Stromfluss in einer SPS beschrieben. Es ist eine simple Sprache, die hauptsächlich boolsche Signale verarbeiten kann. Schleifen und Verzweigungen sind schwer bis nicht möglich[34, 37]. Abb. 2.7 zeigt ein Beispielprogramm geschrieben mit

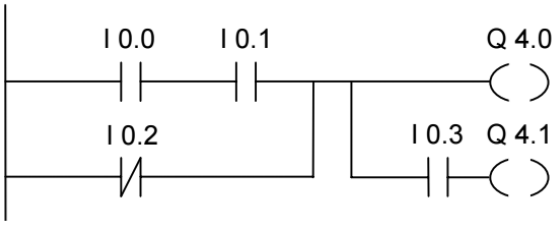

Abbildung 2.7.: Beispiel eines Kontaktplans [62]

einem Kontaktplan. Das Programm ist im folgenden Absatz erklärt.

- Der Zustand von Q4.0 ist „1“, wenn eine der folgenden Konditionen eintritt:
  - die Eingangssignale I0.0 und I0.1 sind „1“
  - oder das Eingangssignal I0.2 ist „0“
- Der Zustand von Q4.1 ist „1“, wenn eine der folgenden Konditionen eintritt:
  - die Eingangssignale I0.0 und I0.1 sind „1“
  - oder das Eingangssignal I0.2 ist „0“ und das Eingangssignal I0.3 ist „1“

2.2.3.5. Anweisungsliste - AWL

Eine Anweisungsliste ist eine textuelle Programmiersprache für SPS’en. Die Syntax einer AWL ist an Assembler angelehnt, kann sich aber von Hersteller zu Hersteller unterscheiden.

Dadurch, dass sich die Hersteller nicht exakt an den Standard halten müssen, ist eine Portierung eines Programms von einem Hersteller zum anderen oft nur mit Anpassungen möglich[34, 6]. Das Codebeispiel in Listing 2.1 zeigt, wie eine AWL aussehen kann.

```
L MD 0 Random pointer
LAR1
A I [AR1,P#10.7]
Ergebnis: Input 21.4 wird angesprochen
```

Listing 2.1: Beispiel für das Aufsummieren von Bitadressen > 7 [61]

2.2.3.6. Funktionsplan - FUP

Der Funktionsplan ist eine grafische Programmiersprache. Daher ist der FUP sehr gut für den Einstieg in die SPS Programmierung geeignet. Es lassen sich Funktionen und Funktionsbausteine inkl. Variablen darstellen und verbinden. Der Signalfluss im Programm läuft von links nach rechts, aber die Abarbeitungsreihenfolge lässt sich definieren[34, 27]. Abbildung 2.8 zeigt einen Funktionsplan mit fünf Eingängen und vier Logikgattern.

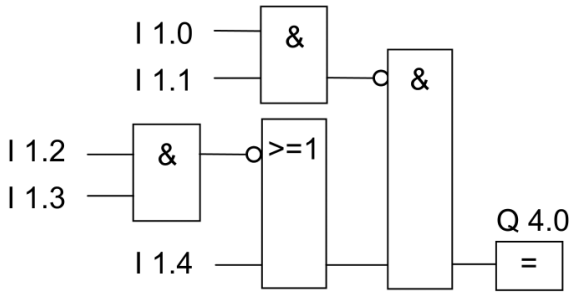

Abbildung 2.8.: Exemplarischer Funktionsplan [60]

Das Gatter mit dem „&“ ist eine logische UND-Verknüpfung, wohingegen „>=1“ einer logischen ODER-Verknüpfung entspricht. Ein Kreis zwischen einem Gatter und einer Leitung bedeutet, dass dieses Signal negiert wird. Das gezeigte Programm funktioniert wie folgt:

- Der Ausgang Q4.0 ist „1“, wenn
  - das Eingangssignal I1.0 UND I1.1 NICHT „1“ ist
  - UND das Eingangssignal I1.2 UND I1.3 NICHT „1“ ist
  - ODER das Eingangssignal an I1.4 NICHT „1“ ist

2.2.3.7. Ablaufsprache - AS

Die Ablaufsprache kann mit einem Petri-Netz verglichen werden. Es gibt Zustände und Zustandsübergänge wie in Abb. 2.9 gezeigt. Der Anfangszustand ist doppelwandig gekennzeichnet. Transitionen sind Bedingungen, die vom System erfüllt werden müssen. Diese Bedingungen gelangen über Sensoren in das System. So kann z.B. in einem Kochkessel eine Bedingung existieren, die bestimmt, dass der Füllstand eine bestimmte Höhe erreicht haben muss, bevor in den Folgezustand übergegangen wird[34, 1].

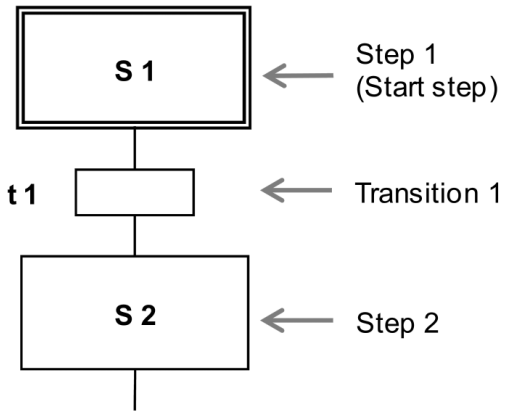

Abbildung 2.9.: Beispiel für den Aufbau eines Ablaufplans [59]

2.2.3.8. Strukturierter Text - ST

Die Syntax von strukturiertem Text, wie er durch den Standard IEC 61131-3 definiert wird, ähnelt der Programmiersprache Pascal. Bei Schüsselworten wird keine Unterscheidung nach Versalien gemacht. Die Sprache ist mächtiger als eine Anweisungsliste, kann aber auf kleineren SPS zu Speicherproblemen führen, da der kompilierte Code meist mehr Speicher einnimmt als kompilierter Code aus einer AWL[34, 70].

2.3. Änderungsausbreitung

In diesem Unterkapitel wird die grundlegende Theorie der Änderungsausbreitung eingeführt. Hier wird erläutert, welche Bedeutung die Änderungsausbreitung hat und warum diese notwendig ist. Der Aspekt wird durch das allgemeine Verfahren der Änderungsauswirkungsanalyse näher betrachtet. Das „Karlsruhe Architectural Maintainability Prediction“-Framework dient sowohl als Beispiel dafür, wie Änderungen im Bereich Software begegnet werden kann, als auch als Basis für die Erweiterung der Funktion des Frameworks für die Domäne der Automatisierungssysteme. Dazu wurden bisher automatisierte Produktionssysteme im Allgemeinen als Beispiel für Automatisierungssysteme vorgestellt. Im konkreten Fall ist die xPPU das Beispiel für ein bestehendes automatisiertes Produktionssystem. Im Folgenden wird die Notwendigkeit dargelegt, warum die Analyse der

```
FUNCTION FC11: REAL
VAR_INPUT
x1: REAL;
x2: REAL;
y1: REAL;
y2: REAL;
END_VAR
VAR_OUTPUT
Q2: REAL;
END_VAR
BEGIN // Code section
FC11:= SQRT // Return of function value
( (x2 - x1)**2 + (y2 - y1) **2 );
Q2:= x1;
END_FUNCTION
```

Listing 2.2: Strukturierter Text [63]

Änderungsausbreitung im Zeitalter Industrie 4.0 eine bedeutende Rolle spielt. Außerdem wird auf die Funktionsweise des KAMP-Frameworks eingegangen.

2.3.1. Notwendigkeit im Zeitalter der Industrie 4.0

Im Zeitalter der Industrie 4.0 gilt es, sich „als Produktionsstandort [...] in einer Hochlohnregion [zu] behaupten“[36]. Dazu ist „eine grundlegende Veränderung und Anpassung der industriellen Produktentwicklung und Produktion“[57] notwendig. „Deutschland und Österreich sind die einzigen der führenden westlichen Industrieländer, die in den vergangenen 20 Jahren den Anteil des produzierenden Gewerbes mehr oder weniger auf demselben Niveau gehalten haben“[57]. Laut einer Studie des Fraunhofer-Institut für Arbeitswirtschaft und Organisation IAO muss „Flexibilität [...] in Zukunft zielgerichtet und systematisch organisiert werden – »Pauschal-Flexibilität« reicht nicht mehr aus“[64]. Um wettbewerbsfähig bleiben zu können wird Flexibilität „zukünftig einer der wichtigen Schlüsselfaktoren [...] sein“[64]. Aber „Flexibilität gibt es nicht umsonst. Sie muss betriebsspezifsch und systematisch geplant werden“[64].

An diesem Punkt setzt die Änderungsausbreitungsanalyse an, um flexibel auf Änderungen am Markt reagieren zu können. Denn wenn es möglich ist, die betroffenen Elemente einer Änderung zu identifizieren, kann so der nötige Aufwand abgeschätzt werden. Mit dem dann bekannten Aufwand können die Kosten, die bei einer Änderung anfallen, berechnet werden[67]. Das ist aber nur ein Aspekt, durch den allein die geforderte Flexibilität noch nicht gegeben ist. Werden Änderungen verglichen, die zum selben Ergebnis führen, kann auf dieser Basis eine für das Unternehmen aus ökonomischer Sicht optimale Entscheidung getroffen werden. Selbstverständlich wird die Entscheidung, ob eine Änderung notwendig ist, von unserem Ansatz nicht betrachtet. Aber er erleichtert die qualitative und quantitative Entscheidungsfindung.

2.3.2. Änderungsausbreitung in Software-Systemen

Wie eingangs bereits erwähnt, lebt Software immer länger. Die Auswirkungen sowohl auf die Software selbst als auch auf die Entwickler und Unternehmen sind vielfältig. Durch einen langen Zeitraum, in dem Software stets weiterentwickelt und auch eingesetzt wird, verfällt diese – sie korrodiert. Korrosion in Software entsteht durch Änderungen, die im Laufe des Lebenszyklus der Software vorgenommen werden. Durch Entwurfsmuster und Prozesse kann dem Verfall entgegengewirkt werden. Dazu ist es aber notwendig, dass die Entwickler, die die Änderungen umsetzen, genügend Erfahrung und/oder Zeit zur Verfügung haben. An diesem Punkt setzt die Vorhersage der Änderungsausbreitung an. Ist bekannt, welche Artefakte geändert werden müssen, ist es möglich, abzuschätzen, wie viel Zeit notwendig ist, diese Änderungen umzusetzen. Die Menge der zu ändernden Artefakte kann nicht exakt bestimmt werden, da in der automatischen Analyse der Artefakte zumindest vom Quellcode ausgehend Unsicherheiten bestehen. Eine Annäherung ist dennoch möglich und führt zu akzeptablen Ergebnissen[67]. Die in diesem Kapitel vorgestellten Verfahren zeigen, wie dem Problem der automatischen Änderungsausbreitung begegnet werden kann.

2.3.2.1. Änderungsauswirkungsanalyse - Change Impact Analysis

„*Software is supposed to change — otherwise software functions would be implemented in Hardware*“ [14], Bohner S.42

Der Grund, warum die Change Impact Analyse gebraucht wird, ist die Auswirkungen und den s.g. Welleneffekt von geplanten Änderungen vorhersagen zu können[14]. Wie Bohner schreibt, sind Änderungen fester Bestandteil einer jeden Software. Dadurch ist Software so lange unvollständig, bis sie in ihrer Funktion nicht mehr gebraucht wird. Ohne die Vorhersage, welche Elemente durch eine Änderungsanfrage betroffen sind, ist es schwierig, die Konsequenzen der selbigen abzuschätzen. Das kann im schlimmsten Fall zu einer Verzögerung der Implementierung führen.

Das von Bohner vorgestellte allgemeine Verfahren ist in Abbildung 2.10 aufgezeigt. Das Straight Impact Set (SIS) beinhaltet die direkt von der Änderungsanfrage betroffenen Artefakte. Das Candidate Impact Set (CIS) enthält die abgeschätzten Artefakte, die von den Änderungen indirekt betroffen sind und ebenfalls geändert werden müssen. Alle Artefakte, die geändert werden müssen, sind im Actual Impact Set (AIS) enthalten. Es besteht die Möglichkeit, dass es mehrere mögliche AIS geben kann, da eine Änderung u.U. auf unterschiedliche Weise umgesetzt werden kann. Das Finden von Objekten, die geändert werden müssen, ist ein iterativer Prozess. Wird eine Änderung durchgeführt, können weitere Objekte gefunden werden, die geändert werden müssen. Das Discovered Impact Set (DIS) entspricht einer unterschätzten Menge von Auswirkungen, wohingegen das False Positive Impact Set (FPIS) die Menge überschätzt. CIS zusammen mit DIS und abzüglich FPIS sollte dem AIS entsprechen. In der Realität ist dem nicht der Fall, aber wie bereits erwähnt, ist eine gute Annäherung möglich[14, 13].

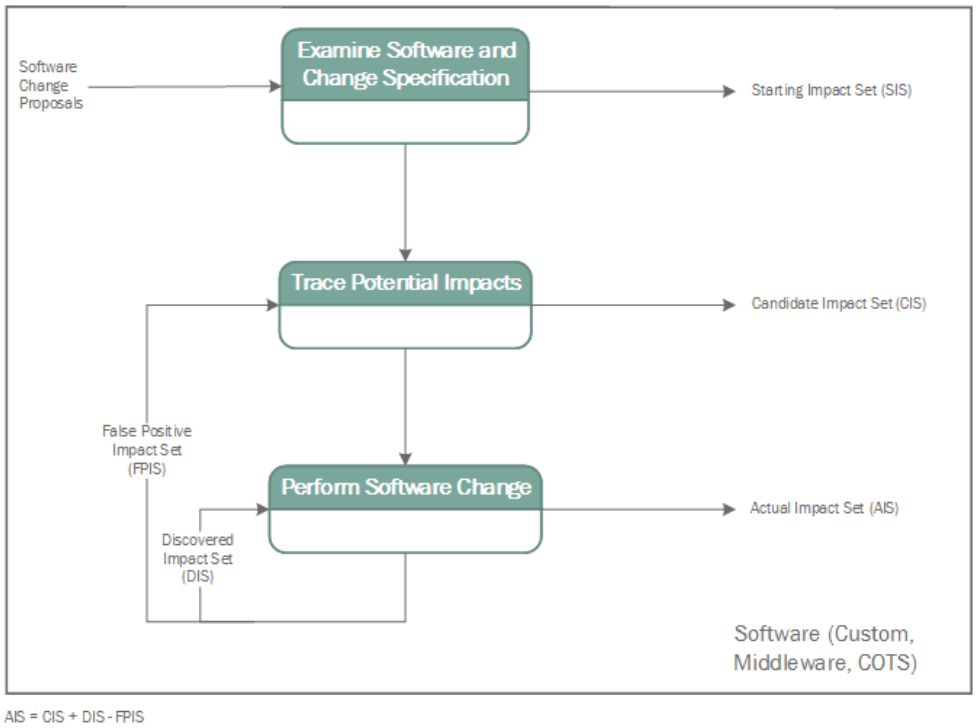

Abbildung 2.10.: Software Change Impact Analyse [13]

2.3.2.2. KAMP - Karlsruhe Architectural Maintainability Prediction

Das Karlsruher Architectural Maintainability Prediction Verfahren ermöglicht es, auf der Basis eines Architekturmodells Änderungsanfragenanalysen zu erstellen. Als Zielgruppe für das Verfahren sind Projektverantwortliche und Architekten angesetzt. Wie in der Einführung von Abschnitt 2.3.2 angeschnitten, ist es vor allem für Projektverantwortliche von besonderer Bedeutung, den Aufwand und somit die Kosten von Änderungen nachhaltig zu bestimmen. Ausgehend von einer Änderungsanfrage werden die Verantwortlichen entscheiden müssen, wie eine Änderung umgesetzt werden soll. Dazu werden die möglichen Umsetzungswege dargelegt, bewertet und geplant. Laut [67] gilt die Identifikation von Umsetzungswegen als kreativer Prozess, der von der Umsetzung der Anforderungen im System und den Möglichkeiten zur Lösung, die dem Architekten zur Verfügung stehen, abhängig ist. Dadurch ist ein manuelles Eingreifen notwendig. Die von KAMP ermittelten Lösungsvorschläge müssen von den Verantwortlichen gesichtet werden. Daraufhin wird auf der Basis der möglichen Lösungen, die für die bestehenden Rahmenbedingungen, beste Lösung bestimmt. KAMP liefert aber nicht nur mögliche Lösungen, die erörterten Lösungswege können auf Modellebene weiter verfeinert und auf deren strukturelle und projektorganisatorischen Auswirkungen hin untersucht werden. Wie in Abschnitt 2.3.2.1 aufgezeigt, können Änderungen an einer Komponente Auswirkungen auf weitere Komponenten haben.

Die daraus resultierenden Arbeitspakete und der damit verbundene personelle Aufwand spielt für die Projektverantwortlichen eine tragende Rolle bei der Entscheidung, welcher Umsetzungsweg gewählt werden soll. KAMP basiert auf einem Architekturmodell, wo-

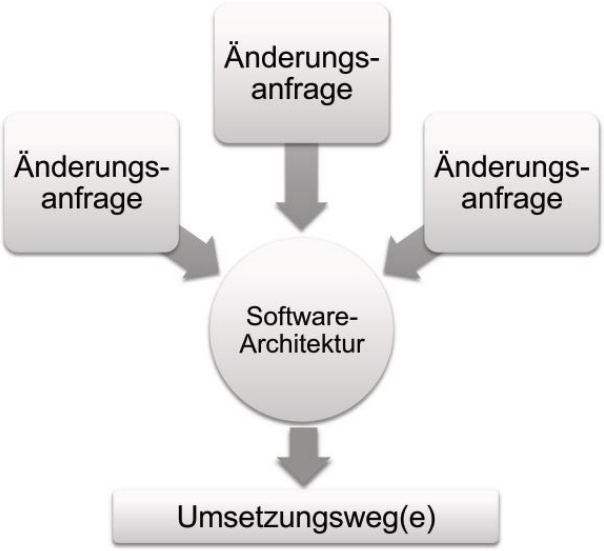

Abbildung 2.11.: Bestimmung der Umsetzungswege von Änderungsanfragen in der Softwareentwicklung [67]

durch es möglich ist, die Analyse auf Architekturebene durchzuführen. Dies ermöglicht dem Betrachter das System auf abstrakter Ebene zu analysieren. Für den Betrachter unwichtige Elemente werden nicht angezeigt. Somit ist es ihm möglich, sich auf das Wesentliche zu konzentrieren. Sind die Umsetzungswege bekannt, kann der Anwender zwischen den Varianten abwägen. Auch ist es möglich, ähnliche Umsetzungswege aus verschiedenen Änderungsanfragen zu kombinieren, sofern sie kompatibel sind[67].

Herausforderungen Projektverantwortliche müssen in der Lage sein, vorgeschlagene Umsetzungswege zu modellieren. Kenntnisse mit modellbasierten Ansätzen sind somit eine wichtige Voraussetzung für den Umgang mit dem KAMP-Framework. Eine gute Modellierung ist Voraussetzung, damit durch das Verfahren der Änderungsausbreitungsanalyse alle möglichen Änderungen betrachtet werden können und somit entsprechende Arbeitspakete und Aufgaben für die einzelnen Tätigkeitsbereiche bestimmt werden können. Bestimmt werden kann auch, wie viele Personen oder auch Personengruppen für die Änderungen benötigt werden und welche weiteren Kosten ggf. durch die Umsetzung entstehen[67].

Ablauf KAMP Prozess Wie Abbildung 2.12 zeigt, ist das Verfahren in zwei Hauptbereiche unterteilt: zum einen die Vorbereitungsphase und zum anderen die Änderungsanfragenanalyse. In der Vorbereitungsphase erstellt der Anwender von einem bestehenden System ein Architekturmodell und ein dazu passendes Anreicherungsmodell, das mit Kontextinformationen angereichert wird. Zu den Kontextinformationen zählen Quelltextentsprechungen, Technologiekonzepte, Baukonfiguration, Testfälle, Bereitstellung, Inbetriebnahme und Personalzuordnung. Auf die Vorbereitungsphase folgt die Ausbreitungsanfragenanalyse. Diese benötigt das angereicherte Architekturmodell sowie eine Menge an Änderungsanfragen. Für jede Änderungsanfrage wird ein Arbeitsplan erstellt. Als Ergebnis bekommt

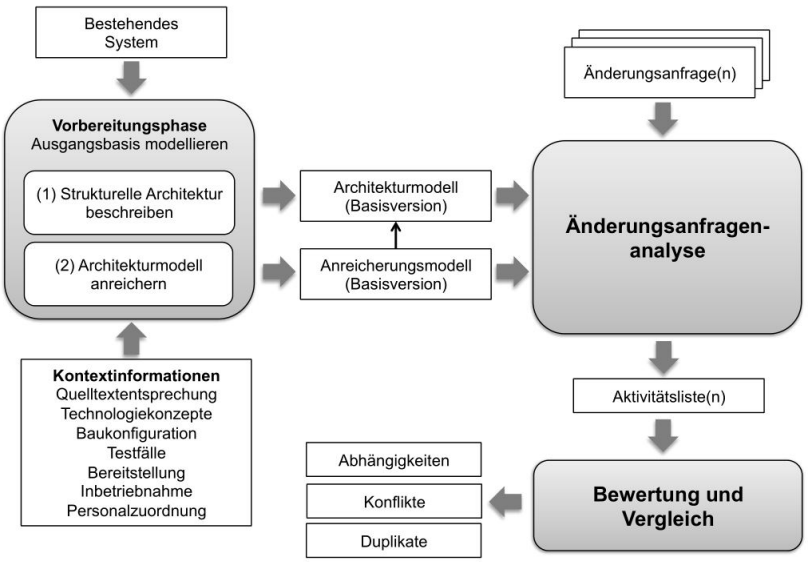

Abbildung 2.12.: Ablauf KAMP Prozess [67]

der Anwender Zielarchitekturmodelle und Aktivitätslisten, die die Umsetzungsschritte einer Änderungsanfrage beschreiben[67].

3. Verwandte Arbeiten

Im Rahmen dieses Kapitels soll dem Leser ein Überblick über Arbeiten geben werden, die sich mit dem Thema der Änderungsausbreitung befassen. Um eine bessere Übersicht zu ermöglichen, ist das Kapitel in zwei Themenbereiche gegliedert. Zum einen werden Arbeiten aufgezeigt, die das Thema der Änderungsausbreitung im Umfeld der Softwareentwicklung betrachten. Zum anderen werden Arbeiten aus der Domäne der automatisierten Produktionssysteme präsentiert, die sich mit der Änderungsausbreitung befassen. Das Ziel ist es, die vorliegende Arbeit von den bereits bestehenden Verfahren abzugrenzen. Dazu werden Gemeinsamkeiten erörtert und Restriktionen aufgezeigt, für die der vorliegende Ansatz Lösungen bieten kann.

3.1. Änderungsausbreitung in Softwaresystemen

Im folgenden Abschnitt werden die verwandten Arbeiten aufgezeigt, die sich mit der Änderungsausbreitung innerhalb der Softwaredomäne befassen. Laut Rostami[53] können die verwandten Arbeiten zum Thema Änderungsausbreitung und Aufwandsschätzung in vier Kategorien eingeteilt werden. Dabei ist für unser Verfahren die Aufwandsschätzung von sekundärer Relevanz. Die folgenden Unterabschnitte befassen sich mit den eben erwähnten Kategorien.

3.1.1. Aufgabenbasierte Projektplanung

Zu diesem Ansatz gehört die hierarchische Aufgabenanalyse (z.B. HTA)[5], die in den 1970ern von Annett und Duncan entwickelt wurde. Dabei gilt es, bestehende Aufgaben in Teilaufgaben zu unterteilen. Der Grad der Unterteilung obliegt dem Prozessplaner. Jede Teilaufgabe, auch Operation genannt, wird definiert durch ein Ziel und deren Eingabeparametern. Es werden Aktionen definiert, die dazu dienen, das gegebene Ziel zu erreichen. Teilaufgaben stehen mit den ihnen übergeordneten Aufgaben in einer Beziehung, die auch Plan genannt wird. Die Unterteilung wird gemacht, um mögliche Leistungseinbußen zu finden und zu optimieren[4]. Die Ursprünge der modernen Aufgabenanalyse liegen laut [68] im frühen 20. Jahrhundert des vergangenen Jahrtausends.

Ebenfalls Teil der aufgabenbasierten Projektplanung ist die Kostenschätzung. Als Beispiel sei hier CoCoMo erwähnt. CoCoMo wurde 1981 veröffentlicht, es wurde an der University of Southern California (USC) am Center for Software Engineering (CSE) entwickelt. Es liegt mittlerweile in einer zweiten Version namens CoCoMo II vor, die im Jahr 2000 veröffentlicht wurde[11]. Müssen HTA oder CoCoMo mit grob-granularen Systemmodellen arbeiten, führt das zu einer schlechten Vorhersage der Änderungsauswirkungen[53].

Wie auch die hier vorgestellten Verfahren nutzt unsere Methode Aufgaben als Basis, um daraus Aktivitäten zu generieren. Dazu werden die möglichen Aufgaben identifiziert, im vorliegenden Fall entspricht eine Aufgabe einer initialen Änderung. Aus den Aufgaben werden Ausbreitungsregeln abgeleitet, die auf ein System angewendet werden. Daraus resultiert dann die Aktivitätsliste. Im Gegensatz zur aufgabenorientierten Projektplanung ist nicht der Projektplan das zentrale Element der Analyse, sondern die Architektur des zu analysierenden Systems.

3.1.2. Architekturbasierte Projektplanung

Im Rahmen dieses Abschnitts werden Ansätze vorgestellt, die sich mit der architekturbasierten Projektplanung befassen. Bei der architekturbasierten Projektplanung wird die Architektur einer Software mit dem entsprechenden Projektplan verknüpft[53]. Die Architektur ist der Hauptbestandteil für nachhaltige und erweiterbare Software. Sie beeinflusst das Verständnis für das Programm und ermöglicht, so dass es schneller erweitert, getestet und gewartet werden kann[38]. Als maßgebliche Grundlage für die architekturbasierte Projektplanung gilt Conways Gesetz[21]. Die von M. Conway im Jahr 1968 verfasste Hypothese lautet, dass die Struktur eines Unternehmens in dessen Entwicklungen repräsentiert sein sollen. Laut [67] ist Conways Gesetz immer noch Diskussionsgrundlage aktueller Forschung. So soll es seit der Veröffentlichung weiter optimiert worden sein. Außerdem wurde die Relevanz für Softwaresysteme untersucht und weiterhin für gültig befunden[39].

Wie Stammel in seiner Dissertation[67] schlussfolgert, sei die Hypothese allein nicht ausreichend, um aktiv eine Lösung bieten zu können. Laut [53] zeigen die Verfahren von Carbon et al.[15] und Bass et al.[48] ebenfalls Möglichkeiten auf, wie die Projektplanung architekturbasiert durchgeführt werden kann. Rostami[53] kommt zu dem Schluss, dass keines dieser Verfahren die Möglichkeit bietet, die Auswirkungen von Änderungen zu benennen bzw. eine Aufgabenliste abzuleiten.

3.1.3. Architekturbasierte Software Evolution

Der Ansatz der architekturbasierten Software Evolution nutzt wie der vorherige Ansatz, die der Software zugrundeliegende Architektur, um die Auswirkungen einer Änderung zu bestimmen. Aber im Gegensatz zur architekturbasierten Projektplanung ist hierbei das Ziel, die Evolution der Software genauer unter die Lupe zu nehmen[53]. Als bedeutender Teil der Software Evolution ist Flexibilität, also der Grad bis zu dem ein System mögliche oder zukünftige Änderungen unterstützt[45], zu betrachten. Wie [46] zeigt, ist Flexibilität ein bedeutungsvolles Qualitätsmerkmal, dass bezogen auf das Verständnis, noch nicht voll und ganz durchdrungen ist. Aus diesem Grund wird in der Arbeit von Naab vorgeschlagen, Flexibilität als feste Anforderung im Architekturdesign zu verankern. So wird auf Basis der Architektur der Grundstein für langlebige Software gelegt.

Auch Garlan et al. befinden die Software Evolution auf Basis der Architektur für bedeutend. Bei deren Ansatz werden Pfade der Evolution definiert. Ähnlich wie bei den Entwurfsmustern der GoF[28] werden die Pfade aus der Erfahrung heraus abgeleitet. Durch eine formale Spezifikation können sie wiederverwendet werden und erlauben es, die Pfade

selbst und auch die Umsetzung auf Fehler und Qualität hin zu überprüfen. Außerdem ist es möglich, verschiedene Evolutionspfade zu vergleichen und somit den besten zu wählen[29]. Auch diese Ansätze sind nicht in der Lage, automatisiert eine Änderungsanalyse oder eine Aufwandsschätzung zu erstellen[53].

3.1.4. Szenariobasierte Architekturanalyse

Änderungen im System werden über Änderungen der Anforderungen bestimmt. Aber die Aufwandsschätzung anhand der Anforderungen ist nur möglich, wenn die Architektur des Systems miteinbezogen wird. Es gibt Ansätze, die die Architekturanalyse nutzen, aber entweder kein formales Modell der Architektur verwenden oder nur auf die Softwareentwicklung angewendet werden[54]. Ein Beispiel dafür ist die Software Architecture Analysis Method (SAAM) von Clements et al.[19]. Dabei werden Szenarien an Komponenten zugewiesen, die von Änderungen betroffen sind. Daraufhin werden entsprechend die Kosten geschätzt. Es werden auch weitere Methoden in [19] vorgestellt.

Unter anderem zählt dazu die „Architecture Trade-Off Analysis Method“ (ATAM). Dabei wird wie zuvor die Architektur als Basis genommen, um darauf basierend die Qualität zu bestimmen. Die Architecture-Level Prediction of Software Mainteneance (ALPSM) sowie die darauf basierende Architecture-Level Modifiablitiy Analysis (ALMA) von Bengtsson et al.[9, 10] analysiert die Propagation von Änderungen durch ein System, setzt aber sehr gute Kenntnisse in der Anwendung voraus[54]. Das bereits vorgestellte KAMP-Framework arbeitet ebenfalls auf Basis der Architektur und die Funktionalität ist anhand von Szenarien implementiert[67].

KAMP4aPS erweitert das KAMP-Framework, um bei aPS szenariobasierte Architekturanalysen durchführen zu können. Im Gegensatz zur architekturbasierten Projektplanung und der architekturbasierten Software Evolution erfolgt die Analyse im KAMP-Framework und somit im KAMP4aPS automatisiert.

3.2. Änderungsausbreitung in automatisierten Produktionssystemen

Der Abschnitt hat zum Ziel, die Verfahren der Änderungsausbreitungsanalysie in der Domäne der aPS zu disskutieren. Eine grundsätzliche Unterteilung wie in Abschnitt 3.1 ist ebenfalls möglich. Um Wiederholungen zu vermeiden werden die Gemeinsamkeiten kurz aufgezeigt, darauf hin wird auf den entsprechenden Unterabschnitt verwiesen.

Wie bereits in Abschnitt 2.2 dargelegt, besteht die Herausforderung in dieser Domäne darin, die verschiedenen Disziplinen (Mechanik, Elektronik und Software) zu koordinieren. Vogel-Heuser et al. berschreibt in [75] die Herausforderungen, denen bei der Entwicklung einer aPS begegnet werden muss.

Die HTA[5] kann ebenfalls bei aPS angewendet werden. Da Annett in dieser Publikation allgemein auf den Vorgang eingeht, wie Aufgaben hierarchisch strukturiert werden können. Dabei spielt die Domäne erst einmal keine Rolle. Ebenfalls sind die in das Verfahren eingebrachten Optimierungen[4] ebenso für die Domäne der aPS gültig.

Conways Gesetz[21] ist auch für Automatisierungssysteme anwendbar, bzw. hat Conway die Hypothese allgemein für Entwicklungsorganisationen[67] aufgestellt und später wurde die Hypothese für die Softwaredomäne verifiziert[39].

3.2.1. Feature-orientierte Analyse

Die Arbeit von Prähofer et al.[49] beschreibt eine feature-orientierte Analyse. Das Verfahren operiert unter der Prämisse, dass von einer Produktlinie ausgegangen wird (vgl. Software Produktlinie[18, 22, 76]). Als Beispiel führen Prähofer et al. ein Unternehmen an (Kaba AG), dass Hardware und Software für Automatisierungssysteme entwickelt. Dieses Unternehmen produziert die Hard- und Software in verschiedenen Varianten, um den verschiedenen Anforderungen am Markt zu entsprechen.

Dabei werden von den Kunden der Firma spezifische Automatisierungssysteme in Auftrag gegeben. Dadurch, dass jedes System den speziellen Kundenwünschen zu entsprechen hat, wird jedes System aus einer Produktlinie ausgewählt und individuell angepasst. Aus einer Menge an Basiskonfigurationen zu wählen und diese entsprechend den Anforderungen zu modifizieren entspricht der Idee der Produktlinie.

Um die Auswirkungen der Änderungen an der Basiskonfiguration zu berechnen, wird als Grundlage für die Auswirkungsanalyse eine konfigurationsbasierte Methode vorgeschlagen. Dabei ist in den Konfigurationen die mögliche Variabilität hinterlegt. Die Analyse erfolgt nicht voll automatisiert, da der Entwickler bei jeder neuen Version die möglichen Auswirkungen manuell anpassen muss[49].

3.2.2. Fallstudienbasierte Analyse

Bei der fallstudienbasierten Analyse von Vogel-Heuser et al.[80] dienen Fallstudien dazu, Änderungsszenarien in einem System zu beschreiben. Als Beispiel für ein aPS wird die PPU aufgezeigt. Die PPU dient als Demonstrator um u. a. die Evolution eines aPS zu erforschen. Die Evolutionsstufen sind in Szenarien unterteilt, diese stellen die verschiedenen Evolutionsstufen der PPU dar.

Die Szenarien können sequentielle Evolutionsschritte beschreiben, aber auch parallele Evolutionsschritte sind möglich. Zusätzlich zur Fallstudie der PPU für aPS wird CoCoME als Fallstudie für Softwaresysteme behandelt. Die Analyse der Änderungen erfolgt für die PPU nicht automatisiert. Die Publikation dient der Annäherung der beiden Domänen im Bezug auf die Änderungsausbreitungsanalyse.

3.2.3. Delta-Extraktion

In [79] stellen Vogel-Heuser et al. eine Möglichkeit vor, wie mit der Hilfe der Delta-Extraktion Änderungen bestimmt werden können. Wie auch die feature-orientierte Analyse[49] nutzt die Delta-Extraktion das Prinzip der Produktlinie um verschiedene Varianten eines Systems zu bilden. Ein Delta gibt an, wie eine validen Variante eines Systems in eine andere Variante transformiert werden kann.

Dabei muss das System als Modell vorliegen, so dass die Transformation auf diesem Modell durchgeführt werden kann. Laut [79] ist das manuelle Erstellen der Deltas fehler-

behaftet, so dass diese semi-automatisch erzeugt werden. Dabei werden Varianten einer Produktlinien erzeugt, um aus diesen mittels eines Vergleichs die Unterschiede zu erhalten. Aus dem Vergleich kann dann automatisiert ein lauffähiges Delta erzeugt werden.

4. KAMP für aPS

In diesem Kapitel wird der Ansatz dieser Arbeit beschrieben. Dabei wird darauf eingegangen, wie das KAMP-Framework erweitert wird. Dazu werden die notwendigen Metamodelle beschrieben, die für die Änderungsanfragenanalyse notwendig sind. Ebenso wird der Prozess der Änderungsanfragenanalyse erläutert, um daraufhin die Ansatzpunkte herauszuarbeiten, die bei der Erweiterung benötigt werden. Die Grundzüge des KAMP-Frameworks wurden in Abschnitt 2.3.2.2 beschrieben. Die Erweiterung des Frameworks wird KAMP für automatisierte Produktionssysteme genannt, kurz KAMP4aPS.

4.1. Änderungsanfragenanalyse

Für die Änderungsanfragenanalyse werden mindestens drei Metamodelle benötigt: das Architekturmodell, das Änderungsanfragenmodell und das Anreicherungsmodell. In diesem Abschnitt werden die Metamodelle kurz beschrieben, und im Kontext der Änderungsanfragenanalyse dargestellt. Dazu wird die allgemeine Funktion erläutert. Im weiteren Verlauf dieser Arbeit, wird auf die Metamodelle im Kontext der aPS in den Kapiteln 5 und 7 eingegangen. Dabei werden die für diese Domäne erstellten Metamodelle dargestellt. Alle Metamodelle, sowie die Änderungsbeschreibungen werden in der Vorbereitungsphase erhoben. Dabei ist hervorzuheben, dass die gewonnenen Informationen für die Analyse mehrerer Änderungsanfragen verwendet werden können. Nach der Vorstellung der benötigten Metamodelle, wird der Ablauf der Änderungsanfragenanalyse skizziert. Abbildung 4.1 stellt den Ablauf der Änderungsanfragenanalyse dar.

4.1.1. Architekturmodell

Das Architekturmodell beschreibt ein System, auf dem die Änderungsanfragenanalyse durchgeführt werden soll. Im Kontext der Dissertation von Stammel [67] beschreibt das Architekturmodell ein Softwaresystem, für die Domäne der aPS, wird es eine Produktionsanlage beschreiben. Dabei ist zu beachten, dass das Architekturmodell den Anforderungen eines Metamodells entspricht. Somit sollten in diesem Architekturmodell nur Elemente enthalten sein, die entweder von einer Änderung betroffen sein können, oder die für die Änderungspropagation benötigt werden. Dazu zählen z. B. Komponenten oder Schnittstellen, sowohl in einem Softwaresystem, als auch in einem aPS[67].

Der Vorteil dieser Art der Darstellung ist, dass der Anwender auf abstrakter Ebene die Struktur des Systems darstellen kann. Selbst bei einer genauen Anforderungsabbildung auf das Architekturmodell, müssen die nötigen Regeln zur Änderungspropagation manuell umgesetzt werden. Das ist notwendig, da die Änderungsausbreitung abhängig ist von den entsprechenden Schnittstellen. In Bezug auf ein Softwaresystem kann nicht garantiert

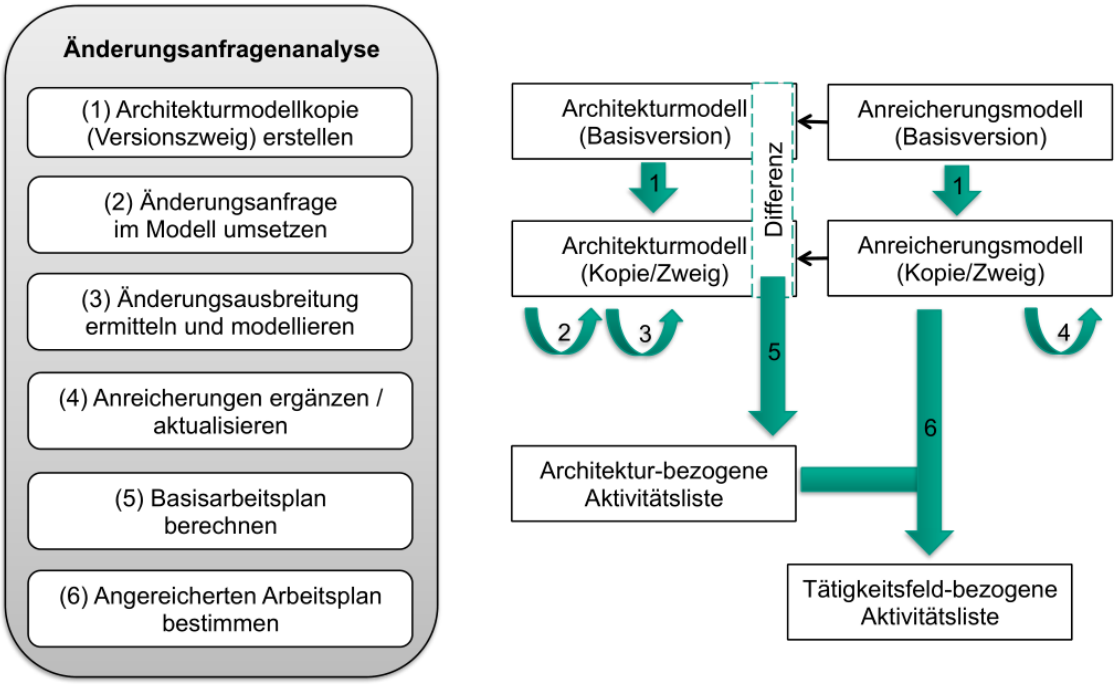

Abbildung 4.1.: Ablauf der Änderungsanfragenanalyse [67]

werden, dass die Signatur einer Schnittstelle stabil bleibt. Das Architekturmodell dient als Basis für die Analyse, die auf Architekturebene durchgeführt wird. Dabei wird von den Projektverantwortlichen vorausgesetzt, dass diese die Umsetzungswege im strukturellen Architekturmodell umsetzen können[67].

Bei der Änderungsbeschreibung ist zu beachten, dass diese im Architekturmodell darstellbar ist. Dazu muss u. U. das Architekturmodell angepasst werden, falls es nicht möglich ist, die Änderungsbeschreibung abzubilden. Abhängig von der Art der Änderung, muss diese in einem zweiten Architekturmodell abgebildet werden. Der Grund dafür ist, dass im KAMP-Framework bestimmte Arten von Änderungen nicht im Änderungsanfragemodell abgebildet werden können. Dazu zählt das Hinzufügen und Entfernen eines Elements im Architekturmodell. In Abbildung 4.1 ist dieser Punkt im Diagramm unter „Differenz“ skizziert. Dabei wird die Differenz der Architekturmodelle vor und nach der Änderung gebildet, um die hinzugefügten und entfernten Elemente zu identifizieren[67].

4.1.2. Änderungsanfragenmodell

Im Änderungsanfragemodell werden die Änderungen, die am System durchgeführt werden sollen, angelegt. Dabei ist zu beachten, dass Änderungen am Architekturmodell, die dem Hinzufügen oder Entfernen entsprechen, nicht damit abgebildet werden können. Dafür wird ein zweites Architekturmodell benötigt. Der Entwickler legt im Änderungsanfragemetamodell fest, welche Änderungen prinzipiell möglich sind. Diese können dann vom Projektverantwortlichen im Änderungsanfragemodell angelegt werden.

Neben der Änderung selbst, die angelegt werden muss, muss das zu ändernde Element im Architekturmodell referenziert werden. Denn die Änderungen sollten so angelegt sein, dass diese für jeden Typ des Elements verwendet werden kann. Damit ist gewährleistet, dass somit mehrere Änderungsanfragen bearbeitet werden können. Außerdem ist der mögliche Aufwand beim Erstellen des Metamodells und beim Umsetzen der Ausbreitungsregeln geringer.

Die Änderungsanfragenanalyse wird in diesem Modell initiiert. Dazu muss der Projektverantwortliche, nachdem die gewünschten Änderungen angelegt wurden, die Architekturanalyse starten. Neben der Architekturanalyse stehen dem Anwender zwei weitere Analyseschritte zur Verfügung. Der zweite Analyseschritt, Nutzerentscheidungen, in die Analyse mit einfließen zu lassen, wird für diese Arbeit nicht benötigt. Daher wird darauf nicht weiter eingegangen. Im letzten Analyseschritt, dem Erzeugen der Aktivitätsliste, werden die Informationen aus dem Anreicherungsmodell, mit dem Ergebnis der Architekturanalyse angereichert. Abschließend wird die Aktivitätsliste angelegt. Es ist wichtig, dass die drei Schritte in eben dieser Reihenfolge ausgeführt werden, da ansonsten die Liste nicht erzeugt werden kann.

4.1.3. Anreicherungsmodell

Im Anreicherungsmodell werden die nicht-strukturellen Änderungen erfasst. Dazu zählen alle Artefakte, die nicht im Architekturmodell enthalten sind, aber dennoch von einer Änderung betroffen sein können. Stammel bezeichnet in [67] dies als organisatorische Konsequenzen für die Umsetzung. Als Beispiel führt Stammel das Bedürfnis der Projektverantwortlichen, nach eben diesen organisatorischen Konsequenzen, an. Dazu zählen die Auswirkungen auf Arbeitspakete und den daraus resultierenden Aufgaben, aber auch, wie viele Personengruppen oder Personen für die Umsetzung der Änderung(en) benötigt werden. Dabei gilt für das Anreicherungsmodell, dass nur die für die Projektplanung relevanten Informationen darin abgebildet werden. Mit dem Erstellen dieses Modells werden die darin enthaltenen Informationen der automatisierten szenarienbasierten Architekturanalyse und Aktivitätsh erleitung verfügbar gemacht[67].

4.1.4. Ablauf der Änderungsanfragenanalyse

Wenn alle nötigen Modelle erstellt wurden, kann die Änderungsanfragenanalyse gestartet werden. Dabei ist zu beachten, dass kein weiteres Architekturmodell benötigt wird, wenn keine Differenz der Modelle erstellt werden soll. Ebenso kann ein Anreicherungsmodell nur im Zusammenhang mit einem Architekturmodell verwendet werden. Wird also kein weiteres Architekturmodell erstellt, wird auch kein zusätzliches Anreicherungsmodell benötigt.

Die Änderungsanfragenanalyse läuft ab wie folgt: es wird eine Kopie des Architektur- und Anreicherungsmodells erstellt (1), die definierten Änderungsanfragen im Änderungsanfragenmodell werden umgesetzt (2), die Änderungsausbreitung wird ermittelt und modelliert (3), das Anreicherungsmodell wird ergänzt/aktualisiert (4), der Basisarbeitsplan wird berechnet (5), auf Grund des Basisarbeitsplans wird der angereicherte Arbeitsplan berechnet (6). Die Arbeitsschritte 1-4 sind manuell durchzuführen, Arbeitsschritt 5 und 6

werden manuell gestartet, aber automatisiert durchgeführt. Abbildung 4.1 zeigt den Ablauf der Änderungsanfragenanalyse.

4.2. Erweiterung des KAMP-Frameworks

Im vorherigen Abschnitt wurde der allgemeine Ablauf der Änderungsanfragenanalyse geschildert. In diesem Abschnitt wird der Ablauf im Kontext der aPS erläutert. Dabei werden die zu erstellenden Metamodelle kurz beschrieben und auch die Extraktion der Ausbreitungsregeln wird behandelt. Dieser Abschnitt dient zur generellen Richtungsweisung. In den nachfolgenden Kapiteln werden die Metamodelle und Ausbreitungsregeln genau beschrieben. Abbildung 4.2 zeigt die Modelle des KAMP4aPS.

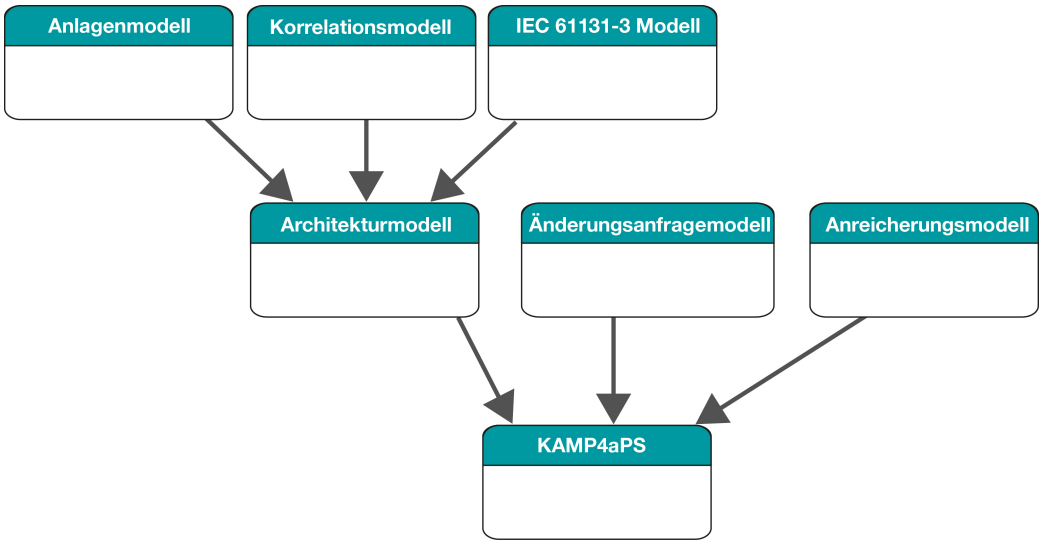

Abbildung 4.2.: Modelle des KAMP4aPS

4.2.1. Architekturmetamodelle

Für die Domäne der aPS gilt es, passende Architekturmodelle zu erstellen. Wie bereits beschrieben, bestehen aPS aus mechanischen und elektronischen Bauteilen, die mittels Software gesteuert werden. Daher bietet es sich an, für jede dieser Klassen ein Metamodell zu erstellen. Um das Expertenwissen der TUM mit in diese Metamodelle einfließen zu lassen, wird der Hardwareteil auf einem Metamodell der TUM basieren. Das Metamodell der Software nutzt den IEC 61131-3-Standard als Grundlage. Somit bestehen die Architekturmodelle aus einem Metamodell für die Hardware und aus einem für die Software.

Das Metamodell der Hardware wird zunächst auf die xPPU zugeschnitten sein. Dadurch kann evaluiert werden, ob das KAMP-Framework derart erweitert werden kann, um die Änderungsausbreitung aPS zu berechnen. Nach einer erfolgreichen Evaluation besteht die Möglichkeit, das Metamodell der xPPU durch ein allgemeines Metamodell einer aPS zu ersetzen. Das Herausarbeiten eines allgemeinen Metamodells hätte den Umfang dieser Arbeit überschritten, aus diesem Grund liegt der Fokus auf der xPPU.

4.2.2. Hardware-Software Korrelationsmetamodell

Um die Änderungsausbreitung von der Hardware auf die Software berechnen zu können, müssen diese beiden Metamodelle in Relation gesetzt werden. Das wird mithilfe des Hardware-Software Korrelationsmodells realisiert. Der Projektverantwortliche legt darin fest, welches Hardwareelement bei einer Änderung, eine Änderung in der Software provoziert. Dabei kann die Software aus beliebig vielen verschiedenen Programmen bestehen. Das hat den Vorteil, dass diese wiederverwendet werden kann. Außerdem muss somit nur die Software eingebunden werden, die von einer Änderung betroffen sein kann. Somit ist die Übersichtlichkeit für den Projektverantwortlichen besser handhabbar.

4.2.3. Änderungsanfragemetamodell

Im Änderungsanfragemetamodell werden die Elemente einer aPS definiert, die der Projektverantwortliche im Änderungsanfragemodell anlegen können soll. Die in diesem Metamodell enthaltenen Elemente sind abhängig von den Ausbreitungsregeln. Wie diese extrahiert werden, wird im Unterabschnitt 4.2.5 beschrieben. Im Rahmen dieser Arbeit steht die Änderung der Hardware im Vordergrund. Die Änderungen werden aus einer Menge an Szenarien ausgewählt, somit sind die zur Auswahl stehenden Änderungen von diesen abhängig.

4.2.4. Anreicherungsmetamodell

Das Anreicherungsmetamodell enthält für die aPS spezifische, nicht-strukturellen Änderungen und Personen- sowie Rolleninformationen. Teil der nicht-strukturellen Änderungen sind die Tests der Anlage, die nach einer Änderung durchgeführt werden müssen. Ebenso gehören Anpassungen der Dokumentation und der ECAD-Zeichnungen dazu. Aber auch die Mensch-Maschine Schnittstelle ist Teil der nicht-strukturellen Änderungen. Eine Besonderheit der aPS ist die Lagerhaltung der nicht verbauten Elemente. Auch wenn Komponenten bestellt werden müssen, wird das in der Lagerhaltung hinterlegt.

Zusätzlich zu den nicht-strukturellen Änderungen, werden im Anreicherungsmetamodell die Rollen der Mitarbeiter festgelegt. Dabei sind nur die Mitarbeiterrollen von Bedeutung, die die Aktivitäten zur Änderung durchzuführen haben. Auch ist es möglich, dass der Projektverantwortliche einer Aufgabe konkrete Mitarbeiter zuweisen kann. Dies ist notwendig, um eine Aufwands- und Kostenschätzung durchführen zu können. Diese wird im Rahmen der vorliegenden Arbeit nicht betrachtet.

4.2.5. Extraktion der Ausbreitungsregeln

Da KAMP ein szenariobasierter Ansatz ist, müssen zuerst Szenarien definiert werden, die als Basis für die Ausbreitungsregeln dienen. Im vorliegenden Fall der xPPU existieren bereits von der TUM definierte Änderungsszenarien. Für die vorliegende Arbeit wurden zwei dieser Szenarien ausgewählt, die für die Evaluation umgesetzt werden.

Die Szenarien beschreiben den Grund der Änderung, also welches Ziel damit erreicht werden soll. Ebenso sind die Elemente der Anlage beschrieben, die von der Änderung

betroffen sind. Vogel-Heuser et al. beschreiben in [78] eben diese Änderungsszenarien der xPPU. Wobei nur die Auswirkung der Änderungen auf die Hardware betrachtet wird, die Software spielt eine untergeordnete Rolle. Daher muss die Auswirkung auf die Software in Zusammenarbeit mit der TUM bestimmt werden.

Das Herausarbeiten der Ausbreitungsregeln erfolgt in zwei Schritten: zuerst wird eine Aktivitätsliste auf der Basis eines Szenarios erstellt, daraufhin werden Ausbreitungsregeln, die zu dieser Liste geführt haben, notiert. Diese beiden Schritte können fließend ineinander übergehen. Wichtig ist, dass neben den Änderungen an der Hard- und Software, auch die nicht-strukturellen Änderungen sowie entsprechende Rollen in der Aktivitätsliste ebenfalls berücksichtigt werden. Auch wenn diese nicht Teil der Ausbreitungsregeln innerhalb des Architekturmodells sind.

4.2.6. Extraktion der Anreicherungsabhängigkeiten

Anhand der manuell erstellten Aktivitätsliste sind, neben den Ausbreitungsregeln, ebenfalls die Anreicherungsabhängigkeiten zu bestimmen. Bezogen auf das Anreicherungsmetamodell sind die Elemente einer Anlage zu bestimmen, die nicht-strukturelle Änderungen provozieren. Dabei ist zu beachten, dass nicht alle nicht-strukturellen Änderungen in einem Szenario enthalten sein müssen. Dementsprechend muss das Metamodell für jedes Szenario angepasst werden, sofern dies notwendig ist. Ebenso besteht die Möglichkeit, dass ein Szenario keine derartigen Änderungen benötigt.

Für jede Änderung kann ebenfalls eine Rolle definiert werden. Der Projektverantwortliche ist in der Lage, konkrete Mitarbeiter einer Rolle zuzuweisen. Somit wird neben der Aufwands- und Kostenschätzung auch eine personelle Aufwandsschätzung betrieben. Es besteht auch die Möglichkeit, dass Anreicherungsmodell nicht anzulegen. Dadurch werden in der Aktivitätsliste nur die Auswirkungen auf die Hard- und Software betrachtet. Das kann im ersten Schritt u. U. sinnvoll sein, um aus verschiedenen Möglichkeiten wählen zu können.

5. Metamodelle der Hard- und Software

Die Grundfunktionalität des Frameworks, also die Struktur zur Berechnung der Änderungsausbreitung, ist bereits vorhanden. Um nun die Domäne der automatisierten Produktionssysteme zu erschließen, muss das Framework erweitert werden. Dies gilt für jede neue Domäne, nicht nur für die der aPS. Dazu ist es notwendig, die Domäne zu modellieren. Bezogen auf automatisierte Produktionssysteme und im Speziellen auf die xPPU muss ein Metamodell für diese erstellt werden. Das Metamodell der xPPU ist im Abschnitt 5.1 zu finden.

Da außerdem die Auswirkung auf die Software analysiert wird, muss ebenfalls ein Metamodell des Standards IEC 61131-3 vorhanden sein, das in Abschnitt 5.2 erläutert wird. Zusätzlich zur Anlage und deren Software müssen die beiden Elemente noch in Verbindung gebracht werden. Daher wurde ein Metamodell definiert, dass die Verbindung zwischen der Hardware und der Software herstellt. Um nun diese domänenspezifischen Metamodelle dem Framework und dem Nutzer zugänglich zu machen, muss das Änderungsanfragenmodell erstellt werden. Darin wird definiert, welche initialen Änderungen dem Nutzer zur Verfügung stehen.

Das Änderungsanfragenmodell ist in Abschnitt 7.1 erklärt. Zusätzlich zu den Modellen einer jeden Domäne ist es ebenfalls notwendig, Ausbreitungsregeln zu identifizieren, zu definieren und zu implementieren. Aus diesem Grund wird in Abschnitt 8 erläutert, welche Ausbreitungsregeln umgesetzt, aber auch, wie sie implementiert wurden. Dabei werden die Regeln als Pseudo-Code vorgestellt und analysiert. Im folgenden Abschnitt werden die im Rahmen dieser Arbeit erstellten Metamodelle und die darauf basierenden Modelle vorgestellt. Zu Beginn werden die Metamodelle des bestehenden Systems, der xPPU, aufgezeigt. Dabei wird differenziert zwischen dem Metamodell einer Produktionsanlage und der xPPU als Instanz davon. Daraufhin werden die für KAMP notwendigen Metamodelle der Änderungsanfrage und Architekturmodellanreicherung eingeführt und die für die aPS notwendigen Anpassungen erläutert.

Das Metamodell einer Anlage ist aber nur ein Teil der notwendigen Modelle, um eine Vorhersage der Änderungspropagation zu erhalten. Denn neben den mechanischen und elektronischen Bauteilen ist die Software ein elementarer Bestandteil einer automatisierten Produktionsanlage. Daher haben wir neben dem Metamodell einer Anlage, auch ein Metamodell des Programmierstandards IEC 61131-3 erstellt, um die Auswirkungen von Änderungen auf die Software ebenfalls untersuchen zu können. Im Metamodell der Anlage sind zwar die Komponenten enthalten, aber die Information, ob sie mit dieser Software laufen, ist noch nicht bekannt.

Aus diesem Grund wurde das Hardware-Software Korrelations-Metamodell eingeführt, das die Relation zwischen Hardware und Software herstellt. Nachdem die Metamodelle der Anlage vorgestellt wurden, wird auf die Metamodelle des Frameworks KAMP4aPS eingegangen. Zu diesen gehört das Metamodell der Änderungsanfragen. Darin werden

die Modifikationen für die Elemente der Anlage definiert. Im Metamodell der Architekturmodellanreicherung werden die Elemente festgelegt, die nicht direkt in der Anlage vorzufinden sind, aber dennoch von Änderungen an dieser betroffen sein können. Die ganze Erweiterung des KAMP-Frameworks für die Domäne der aPS trägt den Namen KAMP für automatisierte Produktionssysteme oder kurz KAMP4aPS.[1]

5.1. Extended Pick and Place Unit

Als Referenzmodell wird das in AutomationML erstellte Modell der xPPU verwendet. Die extended Pick and Place Unit wurde in Abschnitt 2.2.2 bereits vorgestellt. Das Modell wurde an der TUM auf der Basis der xPPU eintwickelt[77]. Ziel ist es, einen ersten Entwurf des Metamodells zu erstellen, um auf dessen Basis die Ausbreitungsregeln zu testen. Neben simplen Anlagenmodellen ermöglicht das Metamodell der xPPU, die Erweiterungen des KAMP-Frameworks anhand eines konkreten Systems zu erproben. Um die Implementierung zu testen, müssen die Auswirkungen der Änderungen bereits bekannt sein. Somit ist eine gute Möglichkeit gegeben, das KAMP4aPS mittels eines komplexeren Falls zu evaluieren. Außerdem bietet die xPPU die Möglichkeit, da diese als Fallstudie angelegt ist, verschiedene Verfahren in einer kontrollier- und wiederholbaren Umgebung zu testen.

Ziel ist es, ein Metamodell zu entwickeln, das nicht nur eine gute Repräsentation der xPPU darstellt, sondern ein allgemeingültiges Metamodell einer Anlage zu haben. Dadurch wird es Herstellern von automatisierten Produktionsanlagen ermöglicht, KAMP4aPS auch für ihre Anlagen zu verwenden. Der Fokus der ersten Umsetzung liegt aber auf der xPPU. Die Verallgemeinerung kann und wird erfolgen, wenn KAMP4aPS für die xPPU vollständig ist und hinreichend getestet wurde.

5.1.1. Anlage

Die Produktionsanlage entspricht der Klasse *Anlage*. Sie ist das Kernstück eines automatisierten Produktionssystems. Das Metamodell wurde so erstellt, dass es möglich ist, mehrere Anlagen innerhalb einer Datei zu definieren. Somit kann eine Modellinstanz *1 bis n* Anlagen enthalten. Dadurch ist es möglich, für jede Anlage Änderungen zu spezifizieren und die Änderungsausbreitung zusammenzufassen. Wie in Abb. 2.4 gezeigt, verfügt eine Anlage über beliebig viele Strukturen (siehe Abschnitt 5.1.2). Außerdem war es notwendig, für die Berechnung der Änderungsausbreitung das Metamodell um eine Referenz sowohl auf die Komponentenablage sowie auf die Schnittstellenablage zu erweitern. Abb. 5.1 zeigt die Klasse *Anlage* und die entsprechenden Relationen.

Um eine bessere Übersicht zu erhalten, wird die Klasse Kennzeichnung in keiner der kommenden Grafiken dargestellt. Die Kennzeichnung dient dazu, einem Element eine eindeutige Bezeichnung zuweisen zu können. Sie ist aber für das Metamodell einer Anlage

[1] Für einen besseren Lesefluss wurde entschieden, hauptsächlich deutsche Begriffe der Modellelemente zu verwenden, auch wenn diese im Quelltext und den Metamodellen sowie Modellen auf Englisch enthalten sind. Im Metamodell des IEC 61131-3 wurden die englischen Begriffe beibehalten, ebenso Begriffe wie „Master/Slave“, die im deutschen IT-Sprachgebrauch mittlerweile etabliert sind.

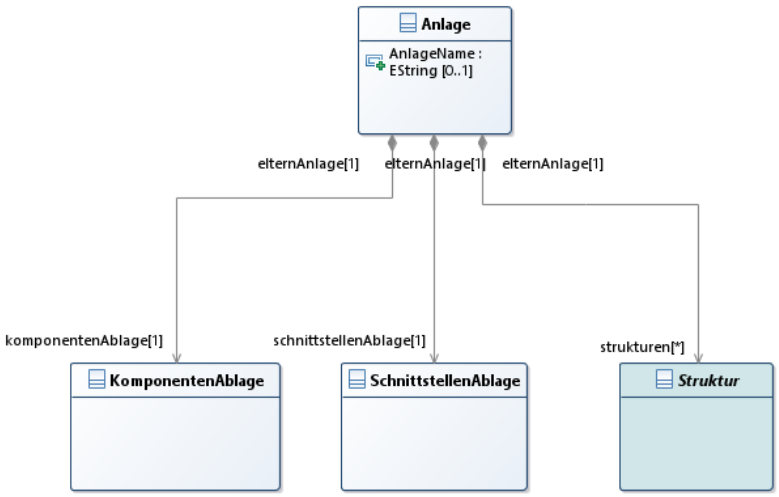

Abbildung 5.1.: Aufbau und Relationen der Klasse *Anlage* [77]

ansonsten von keiner Relevanz. Die Referenzen an zentraler Stelle dienen dazu, alle in der Anlage vorkommenden Schnittstellen bzw. Komponenten zu bündeln.

Aktuell muss der Nutzer auf Instanzebene die Ablagen manuell um neu hinzugekommene Schnittstellen ergänzen. In einer der nächsten Iterationen ist geplant, den Schritt zu automatisieren. Neben der eindeutigen Kennzeichnung, die im Nachhinein noch geändert werden kann, ist es ebenfalls möglich, der Produktionsanlage einen Namen zu geben. Die Unterscheidung wird getroffen, da zwar auf der Ebene der Anlage die Kennung und der Name zur eindeutigen Bestimmung denselben Zweck erfüllt. Aber auf Komponentenebene, könnte es mehrere Elemente mit demselben Namen geben. Somit würde die Bestimmung mittels des Namens nicht mehr funktionieren bzw. der Name müsste auf Kosten der Lesbarkeit erweitert werden.

5.1.2. Struktur/-ablage

Das Strukturelement einer Anlage ist als ein Zusammenschluss mehrerer Elemente realisiert. Neben Komponenten (5.1.4) sind auch Module (5.1.3) Teil einer Struktur. Die Anlage ist nicht beschränkt auf nur eine Struktur, da in einer solchen funktionale Systemteile untergebracht sind. Diese stellen für sich allein gesehen noch keine ganze Anlage dar. Eine Struktur dient also primär zur Gruppierung von Komponenten und Modulen. Im Metamodell der xPPU gehören Netzwerke wie das Kommunikationsnetzwerk oder das Stromnetz zu einer Struktur, aber auch Kräne und Förderbänder sind als Struktur anzusehen. Abb. 5.2 bietet eine Darstellung der eben beschriebenen Struktur. Neben der Gruppierung von Modulen und Komponenten zu funktionalen Einheiten und der daraus resultierenden Übersichtlichkeit können Strukturen gebündelt und in anderen Anlagen wiederverwendet werden. Das reduziert den Modellierungsaufwand einer Anlage, wenn Strukturen bereits modelliert wurden. Es können beliebig viele Komponenten und Module in einer Struktur zusammengefasst werden. Es ist aber ebenfalls möglich, dass nur eine der beiden Klassen in einer Struktur zusammengefasst werden.

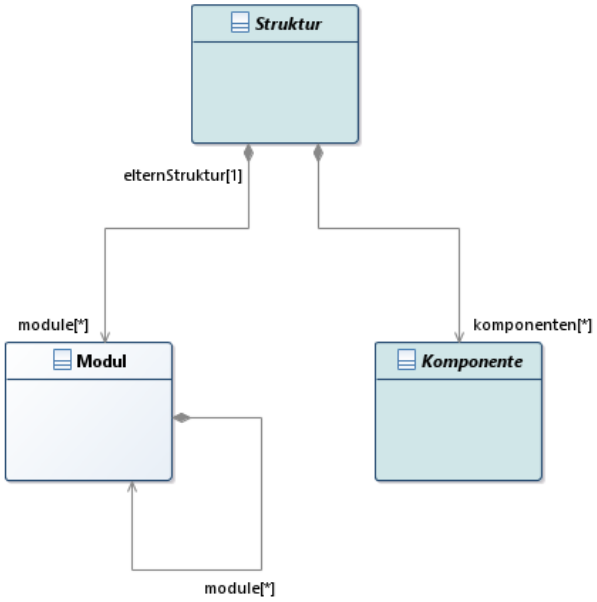

Abbildung 5.2.: Aufbau und Relationen der Klasse *Struktur* [77]

5.1.3. Modul/-ablage

Module innerhalb einer Anlage sind von den Ingenieuren aus Komponenten zusammengesetzte Einheiten. Ein Motormodul der xPPU z.B. besteht neben der Motoreinheit, welche die essentielle Komponente des Moduls darstellt, noch aus Komponenten, die notwendig sind, das Modul mit dem Bus zu verbinden. Im Beispiel in Abb. 5.3 enthält das Motormodul neben dem Motor eine Bus Box, ein Bus Kabel und einen Bus Slave. Der Zusammenschluss einzelner Komponenten ist nicht nur den kombinierten Komponenten geschuldet. Auch die logische Gruppierung von Komponenten, ebenfalls wie bei den Strukturen, bietet die Möglichkeit, Module in einer Ablage zu sammeln und im Modell wiederverwenden zu können.

Außerdem wird durch den Einsatz von Modulen das Metamodell besser strukturiert und übersichtlicher. Im Gegensatz zur Gruppierung einer Struktur dient die Gruppierung innerhalb eines Moduls dazu, auf einer feingranulareren Ebene Elemente zusammenzufassen. Aber nicht nur Komponenten können in Modulen enthalten sein. Ein Modul kann wiederum als ein Teil von einem oder mehreren anderen Modulen dienen. Ein Fixierungsmodul bestehend aus Schrauben und Streben kann sowohl einen Motor als auch einen Zylinder fixieren. Um nicht eine Fixierung für den Motor und den Zylinder erstellen zu müssen, ermöglicht das Metamodell, dass sie beiden Elementen zugewiesen werden kann.

5.1.4. Komponenten/-ablage

Das Paket Komponentenablage enthält die im Metamodell vorkommenden Komponenten. Dazu zählen alle Teile, die als fertige Bauteile eingekauft werden und nicht selbst gefertigt werden. Teil davon sind mechanische Bauteile wie Einhausungen, Schrauben und Gummibänder, aber auch elektronische Bauteile wie Motoren und LEDs. Komponenten stellen

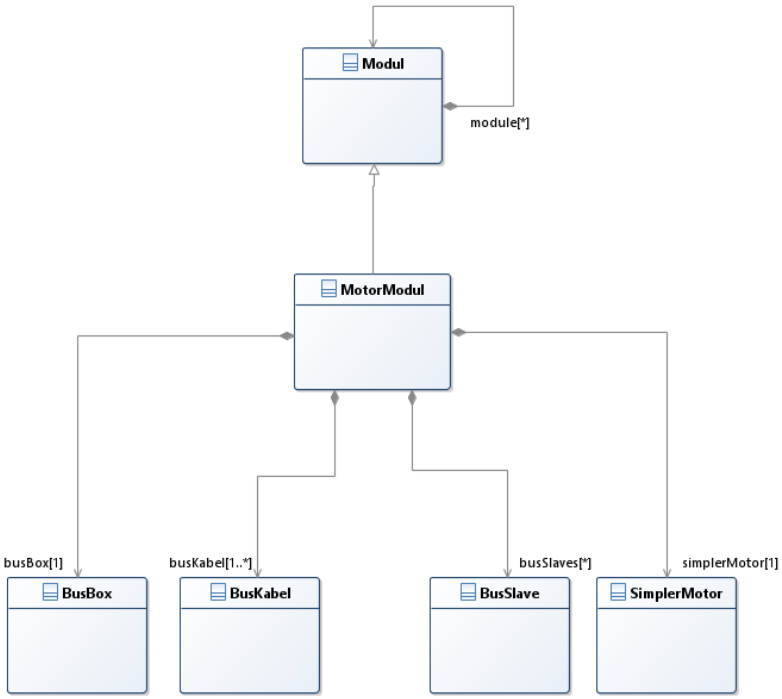

Abbildung 5.3.: Struktur eines Motormoduls [77]

demnach das kleinste Bauteil im physischen wie im übertragenden Sinn, einer Fertigungsanlage dar. Abb. 5.4 zeigt eine Zusammenfassung der in den Komponenten enthaltenen Elemente. Zur besseren Übersicht werden nur die Oberklassen in der Grafik abgebildet. Die Menge aller möglichen Komponenten ist derart umfassend, dass die Komponenten in weitere logische Gruppen unterteilt werden. Das bisher kleinste Paket stellen die Bus Komponenten dar. Darin sind alle Komponenten enthalten, die zur Kommunikation auf dem Bus notwendig sind. Da beim Bau einer Produktionsanlage verschiedene Bussysteme zur Auswahl stehen, bot es sich an, diese zu gruppieren.

Wie bei der Abbildung 2.1 bereits geschildert, bestehen automatisierte Produktionsanlagen aus drei Elementen. Daher entspricht die Unterteilung der Pakete eben dieser. Folglich gibt es neben dem Paket der Buskomponenten noch ein Paket für mechanische Komponenten und eines für elektronische Komponenten.

5.1.4.1. Buskomponenten

Komponenten, die das Bussystem einer Anlage beschreiben, finden sich in der logischen Gruppe Buskomponenten (Bus Components). Unter der Kategorie sind alle Elemente untergebracht, die notwendig sind damit eine Kommunikation mit dem in der Produktionsanlage verwendeten Bussystem möglich ist. Sie sind immer Teil anderer Elemente innerhalb einer Anlage.

Damit z. B. ein Kran oder ein Sensor eingebunden werden kann, ist es notwendig, dieser Komponente einen Bus Slave zuzuweisen und sie mit einem entsprechenden Kabel an den Bus anzuschließen. In dem Paket sind neben den Elternklassen Bus Box, Bus Master, Bus

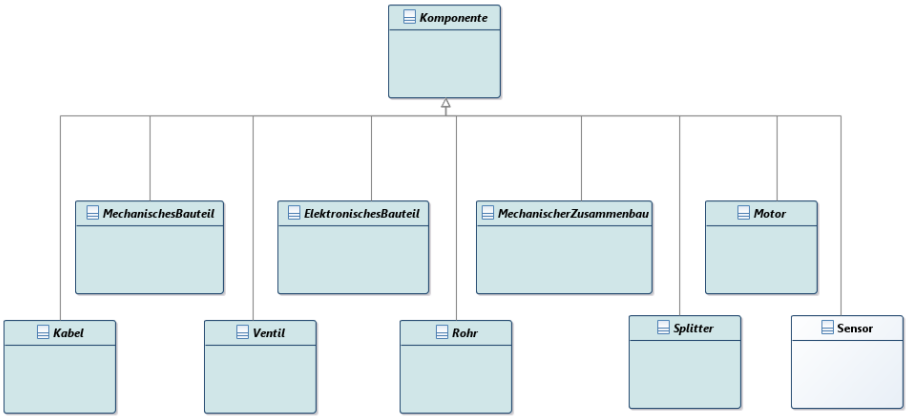

Abbildung 5.4.: Übersicht über die Komponenten [77]

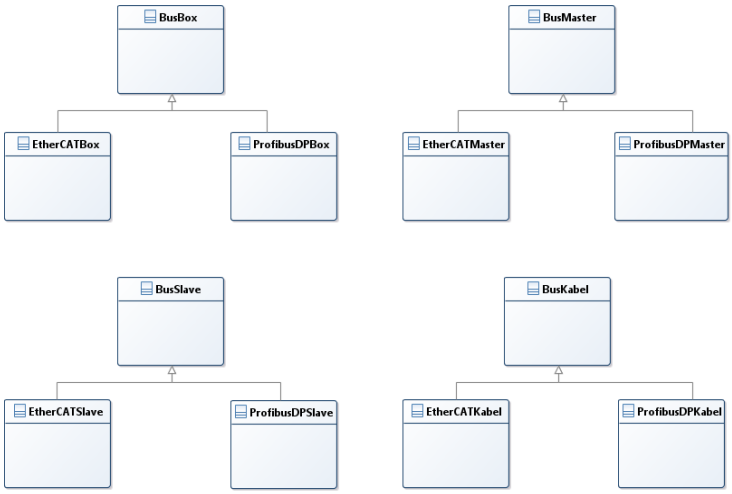

Abbildung 5.5.: Inhalt der Buskomponenten [77]

Slave und Bus Kabel auch Spezialisierungen der jeweiligen Klassen zu finden. Abb. 5.5 zeigt die bereits beschriebene Struktur der Buskomponenten.

In der xPPU-Anlage werden zum aktuellen Zeitpunkt die Busse EtherCAT und ProfibusDP eingesetzt. Daher sind die zwei genannten Busse im Repository „Bus Components“ enthalten. Die Bus Box, auch Buskoppler genannt, ist die zentrale Einheit des Busses. Sie verfügt über *n* Ports, an die die Busteilnehmer mittels Klemmen angeschlossen werden können. Bei den Teilnehmern wird zwischen Bus Mastern und Bus Slaves unterschieden. Verbunden werden die Komponenten mittels der Komponente Bus Kabel.

5.1.4.2. Elektronische Komponenten

Auch elektronische Komponenten werden in einem Repository zusammengefasst. In dieser Gruppe befinden sich rein elektrische Elemente. Ein Elektromotor wird elektronisch betrieben, erfüllt aber primär einen mechanischen Zweck. Daher wird er nicht den elektronischen Komponenten zugeordnet. Abb. 5.6 zeigt die Hauptgruppen der elektronischen

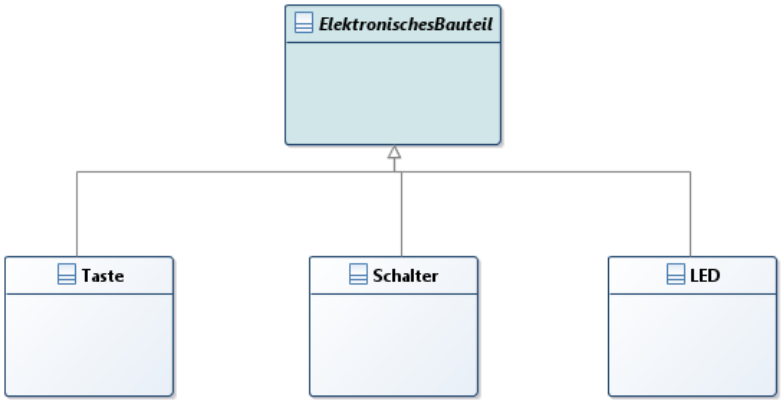

Abbildung 5.6.: Inhalt der elektronischen Komponenten [77]

Bauteile einer Anlage. Die Möglichkeit, weitere elektronische Bauteile hinzuzufügen, ist natürlich gegeben. Dazu kann auf Modellebene eine neue Klasse hinzugefügt werden. Außerdem kann das Metamodell immer noch erweitert werden, falls das nötig werden sollte. Vorgesehen sind momentan Knöpfe (Buttons), Schalter (Switches) und LED's in verschiedenen Ausführungen.

5.1.4.3. Mechanischen Komponenten

Zu der Kategorie der mechanischen Komponenten gehören Bauteile, die primär eine mechanische Aufgabe erfüllen. Wie im vorherigen Abschnitt bereits beschrieben, gehören Elektromotoren zu dieser Gruppe. Auch ist das Band eines Förderbandes oder eine Feder Teil davon. Größere Komponenten, die zwar aus mehreren Teilen zusammengesetzt sind, aber als Ganzes eingekauft werden, gilt es ebenfalls einzuordnen. Der Arm eines Krans oder ein beliebig anderer Aktuator fällt in dasselbe Muster wie der Elektromotor. Es gibt aber auch Komponenten, die intuitiv erst einmal als mechanische Komponente eingeordnet werden. Z. B. wäre ein Getriebe oder eine Fixierung hierunter einzuordnen. Hier wurden Schnittstellen, nichts Anderes ist ein Getriebe oder eine Fixierung, in ein eigenes Paket ausgelagert.

5.1.5. Schnittstellen/-ablage

In der Schnittstellenablage sind alle Komponenten enthalten, die in die Kategorie der Schnittstelle fallen. Dazu zählen sowohl physikalische Schnittstellen wie ein Getriebe oder eine Halterung bzw. Fixierung. Das Getriebe ist eine Verbindung, die zur Kraftübertragung dient und eine Fixierung ist die Verbindung einer Komponente zu einer Basis. Aber auch logische und somit nicht unbedingt in der physischen Welt vorhandene Schnittstellen gehören in dieses Paket. Daher wurde entschlossen, eine eigene Ablage dafür bereitzustellen. Die Abbildung 5.7 liefert einen Überblick über die im Metamodell der xPPU enthaltenen Schnittstellen. Zwei der wichtigsten Schnittstellen für die xPPU sind zum einen die physische Verbindung (Physical Connection), also die Fixierung, und zum ande-

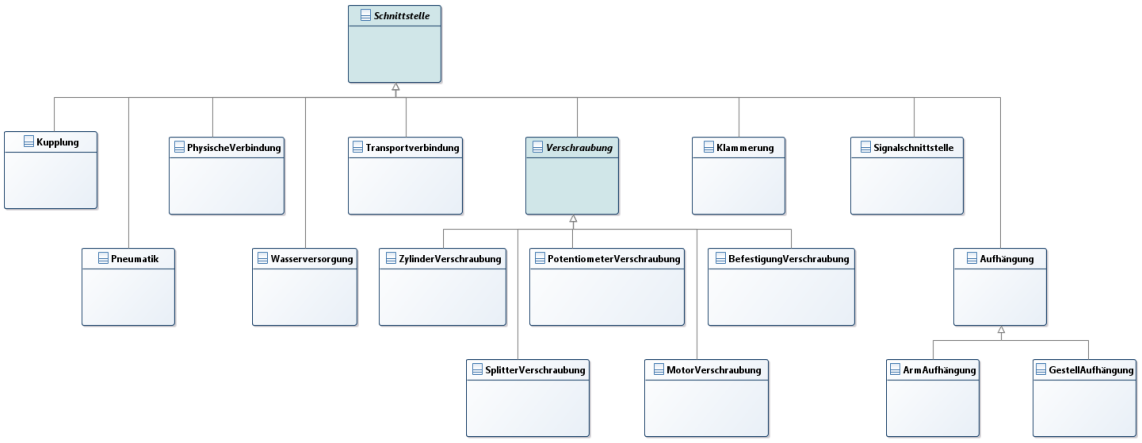

Abbildung 5.7.: Übersicht über die Schnittstellen [77]

ren die Signalschnittstelle (Signal Interface), das für den Aufbau des Busses von tragender Bedeutung ist.

5.2. IEC 61131-3

Das Metamodell für IEC 61131-3 basiert auf einem Referenzschema der Universität Koblenz-Landau vom Institut für Softwaretechnik[31] und einem Metamodell der Firma Codesys und deren Interpretation des Standards. Codesys hat uns ihre Daten bereitgestellt. Die Metamodelle lagen im XML-Format vor und mussten in das Ecore-Format überführt werden. Manuell wäre der Aufwand zu groß gewesen, daher wurde entschlossen einen Parser zu verwenden, um die Modelltransformation durchzuführen. In diesem Abschnitt wird die Struktur des Metamodells grob beschrieben, indem die drei Hauptpakete dargestellt werden. Anschließend wird auf besondere Zusammenhänge gesondert eingegangen.

5.2.1. Die Common und Configuration Pakete

Auf der ersten Hierarchieebene der Pakete befinden sich drei Gruppen. Das Common Paket bildet häufig genutzte bzw. essentielle Elemente wie Variablen und Typen ab. Aber auch Objektorientierung ist darin abgebildet. Im Configuration Paket sind Elemente zur Konfiguration des Programmes enthalten. Im letzten Paket, den Sprachen (Languages), sind die fünf Sprachen modelliert, mit denen Programme nach dem IEC 61131-3 Standard programmiert werden können.

5.2.1.1. Common

Im Paket Common sind die Grundfunktionalitäten des Standards abgebildet. Abbildung 2.6 zeigt die drei Hauptelemente, die allen Sprachen des IEC 61131-3 gemein sind. Das Programm, die Funktion sowie der Funktionsblock sind in einem weiteren Unterpaket

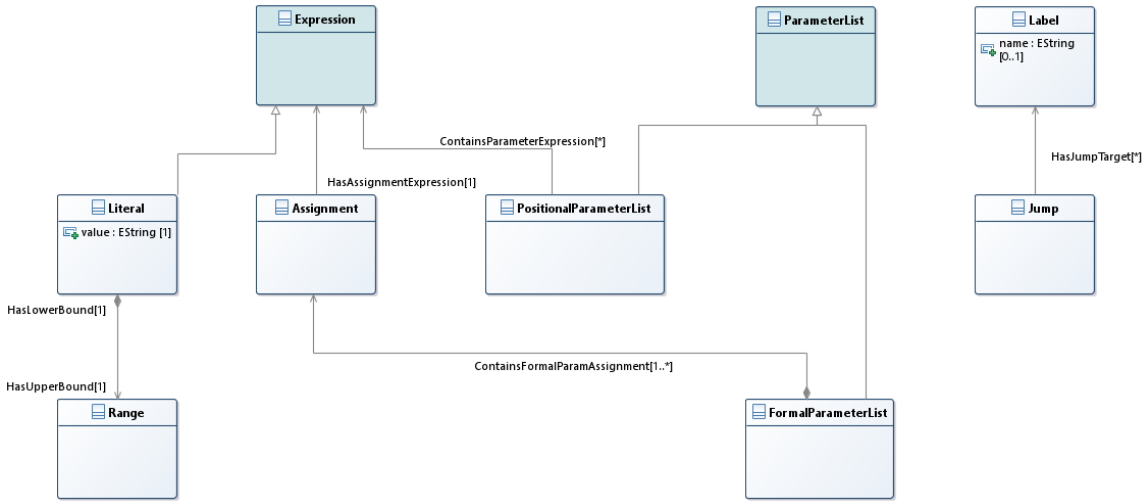

Abbildung 5.8.: Inhalt des Paketes Common [31]

definiert. Aber auch Typen und Variablen werden in Unterpaketen dargestellt. Neben dem in Abschnitt 2.2.3 vorgestellten Aufbau, ist im Commons Paket weiterhin abgebildet, dass die Objektorientierung in IEC 61131-3 möglich ist.

Für die Nutzung der fünf Sprachen ist die Nutzung der Objektorientierung unerheblich. Daher sei die Funktionalität hier zwar erwähnt, aber bei mehr Interesse auf [34] verwiesen. Wie Abbildung 5.8 zeigt, sind neben den Paketen die Grundlagen auf der Metaebene definiert, d.h. Elemente, die von allen Sprachen verwendet werden, sind hier enthalten.

5.2.1.2. Configuration

Abbildung 5.9.: Inhalt des Paketes Configuration [31]

Die Konfiguration ist notwendig, um Anwendungen zu strukturieren und die Interaktion zwischen den POUs zu definieren[34]. In einer Konfiguration werden Laufzeitinformationen beschrieben und auch die Zuweisung zur jeweiligen SPS gemacht. Innerhalb einer SPS wird zuerst die Konfiguration geladen, damit z.B. Variablen einer konkreten Adresse zugewiesen werden.

Abbildung 5.9 zeigt, dass eine Konfiguration aus Ressourcen und Tasks besteht. Eine Ressource ist eine CPU in einer SPS. In einem Task werden Laufzeiteigenschaften für ein Programm oder Funktionsblock hinterlegt. Ist ein Programm aufgeteilt auf mehrere SPSen, kann in der Konfiguration der Datenfluss ebenfalls definiert werden.

5.2.2. Das Language Paket

Die Sprachen des Standards IEC 61131-3 bilden die Kernschnittstelle zur Integration in das KAMP-Framework, um die Änderungsausbreitung und deren Auswirkung auf die Software analysieren zu können. In diesem Abschnitt wird erläutert, wie die fünf Sprachen modelliert sind. Da die Möglichkeit besteht, jede der Fünf in einer Anlage vorzufinden, ist es notwendig auf diese einzugehen.

Die Reihenfolge in der die Metamodelle vorgestellt werden, entspricht der aus Kapitel 2 Abschnitt 2.2.3. Der Fokus wird dabei auf die prägnanten Elemente der Sprachen und auf die Repräsentation im Metamodell gelegt wird. Begonnen wird mit der Beschreibung der visuellen Sprache des Kontaktplans, gefolgt von der textuellen Sprache der Anweisungsliste, der visuellen Sprachen des Funktionsplans und der Ablaufsprache. Abschließend wird auf die letzte Sprache, den strukturierten Text, eingegangen.

5.2.2.1. Kontaktplan

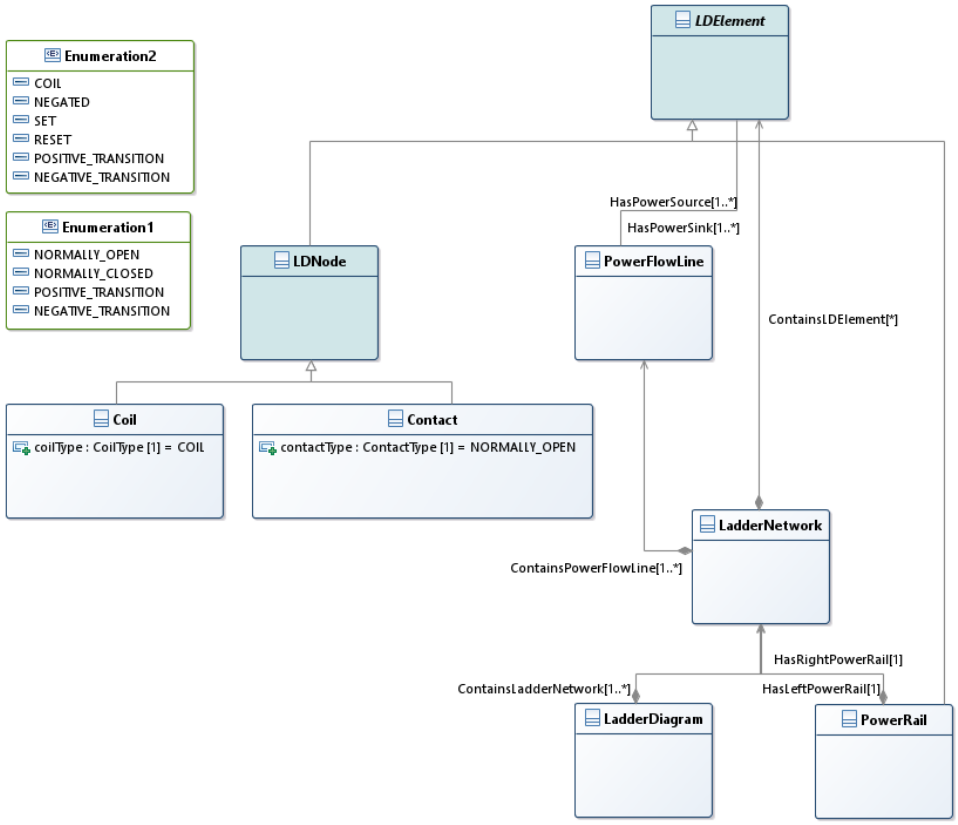

Abbildung 5.10.: Metamodell des Kontaktplans [31]

Der Kontaktplan, im englischen Ladder Diagram genannt, bietet die Möglichkeit ein Programm für eine SPS visuell zu programmieren. Dazu kann der Anwender Kontakte definieren, die die Schnittstelle der SPS abbilden. Die Kontakte können verbunden werden.

Außerdem ist es möglich, Aktuatoren in den Kontaktplan aufzunehmen und diese ebenfalls zu verbinden.

Die Aufzählung Contact Type definiert die Form, die ein Kontakt annehmen kann. Im Coil Type sind die Arten eines Aktuators definiert. Weiterhin sind neben den Typen auch konkrete Klassen für Kontakte und Aktuatoren definiert, die jeweils das passende Enum als Attribut besitzen. Von diesen beiden Klassen abgesehen, ist noch die Struktur in Form der Stromversorgung und Informationsübertragung im Metamodell enthalten.

In Abbildung 5.10 ist der Aufbau visuell dargestellt. Die Sprache eines Kontaktplans ist sehr eingeschränkt. Das spiegelt sich auch im Metamodell wieder. Auch ist die Sprache dazu geeignet, den Aufbau bzw. die Verdrahtung einer SPS darzustellen.

5.2.2.2. Anweisungsliste

Das Metamodell der Anweisungsliste ist in Abbildung 5.11 dargestellt. Die Anweisungsliste ist eine der zwei textuellen Sprachen nach IEC 61131-3. Sie ist vergleichbar mit einer Assemblersprache. Es werden Anweisungen für die SPS in der Anweisungsliste definiert. Abhängig vom Hersteller kann sich die Syntax unterscheiden. Diese Tatsache spiegelt sich auch im Metamodell wieder.

Eine Anweisungsliste enthält Anweisungen, die sich wiederum in vier Kategorien einteilen lassen. Neben Sprunganweisungen und Rückgabeanweisungen, die über keine Attribute verfügen, gibt es noch einfache und komplexe Anweisungen. Wie bereits beschrieben, ist die Syntax abhängig vom Hersteller. Daher sind die Attribute der einfachen und komplexen Anweisungen Strings. Somit können beliebige herstellerspezifische Anweisungen enthalten sein, ohne das Metamodell ändern zu müssen.

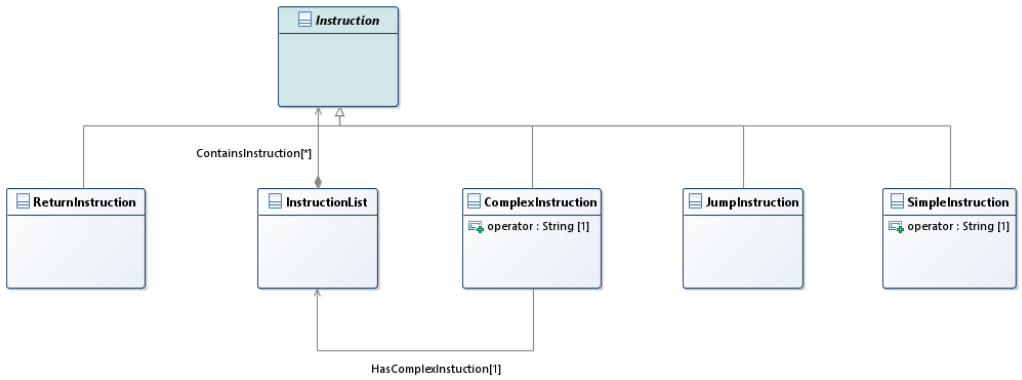

Abbildung 5.11.: Metamodell der Anweisungsliste [31]

5.2.2.3. Funktionsplan

Der Funktionsplan ist ebenfalls eine grafische Sprache nach IEC 61131-3. Im Gegensatz zum Kontaktplan, in dem nur die Verdrahtung der Ein- und Ausgänge abgebildet werden, ist es beim Funktionsplan möglich, Logikschaltungen zu realisieren. Die Darstellung orientiert sich an dem Standard in der Digitaltechnik (mehr dazu im Grundlagenkapitel in Abschnitt 2.2.3). Abbildung 5.12 zeigt die Struktur des Metamodells für einen Funktionsplan. Darin

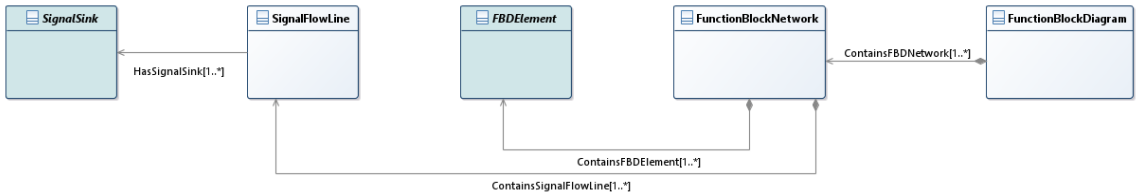

Abbildung 5.12.: Metamodell des Funktionsplans [31]

ist die Verdrahtung und der Signalfluss definiert. Weitere, konkrete Elemente wie z.B. Logikgatter und Negationen sind nicht explizit modelliert.

5.2.2.4. Ablaufsprache

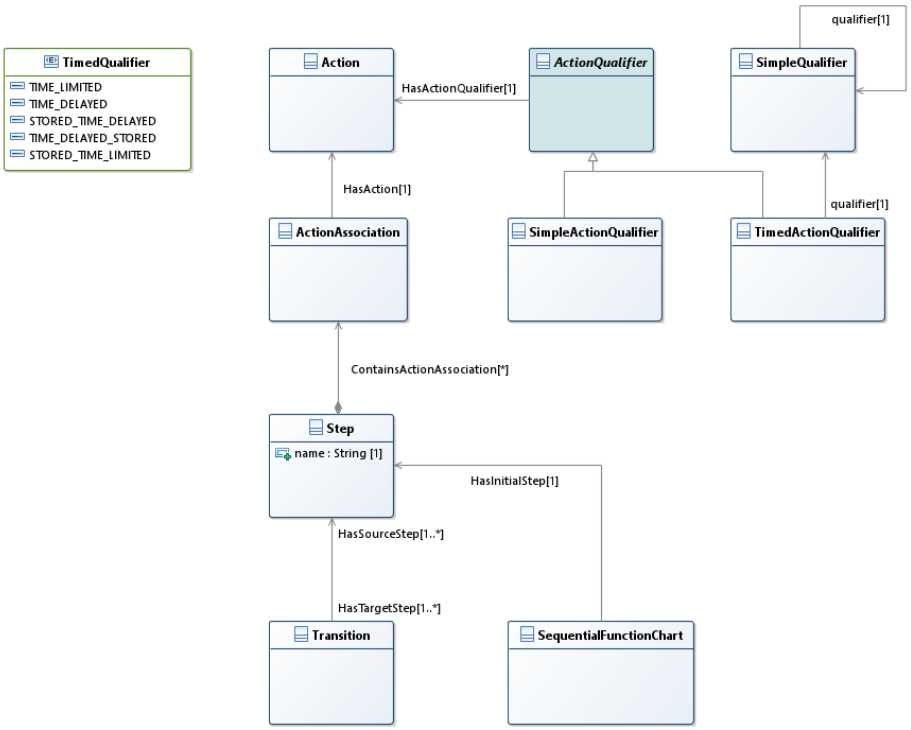

Abbildung 5.13.: Metamodell der Ablaufsprache [31]

Die Ablaufsprache ist die letzte der grafischen Sprachen im IEC 61131-3 Standard. Im Vergleich zum Kontaktplan oder zum Funktionsplan ist die Ablaufsprache die mächtigste. Damit können nicht nur Verdrahtungen modelliert und logische Abläufe implementiert werden, sondern auch die Abbildung komplexer Abläufe wie in einem Petri-Netz sind möglich.

In Abschnitt 2.2.3 im Grundlagenkapitel wird auf die Ablaufsprache genauer eingegangen. Wie in Abbildung 5.13 zu sehen, ist das Metamodell im Vergleich zu den bisher vorgestellten Sprachen etwas umfangreicher. Im Metamodell sind verschiedene Arten

von Timings modelliert, die an Bedingungen gekoppelt sind. Nicht alle Bedingungen sind abhängig von den Timings. Diese Bedingungen gehören zu Aktionen, die einem oder auch mehreren Schritten zugeordnet werden können. Um von einem Schritt zum nächsten zu kommen, sind Übergänge definiert. Dabei ist es möglich, dass ein Zustand mehrere Übergänge haben kann.

5.2.2.5. Strukturierter Text

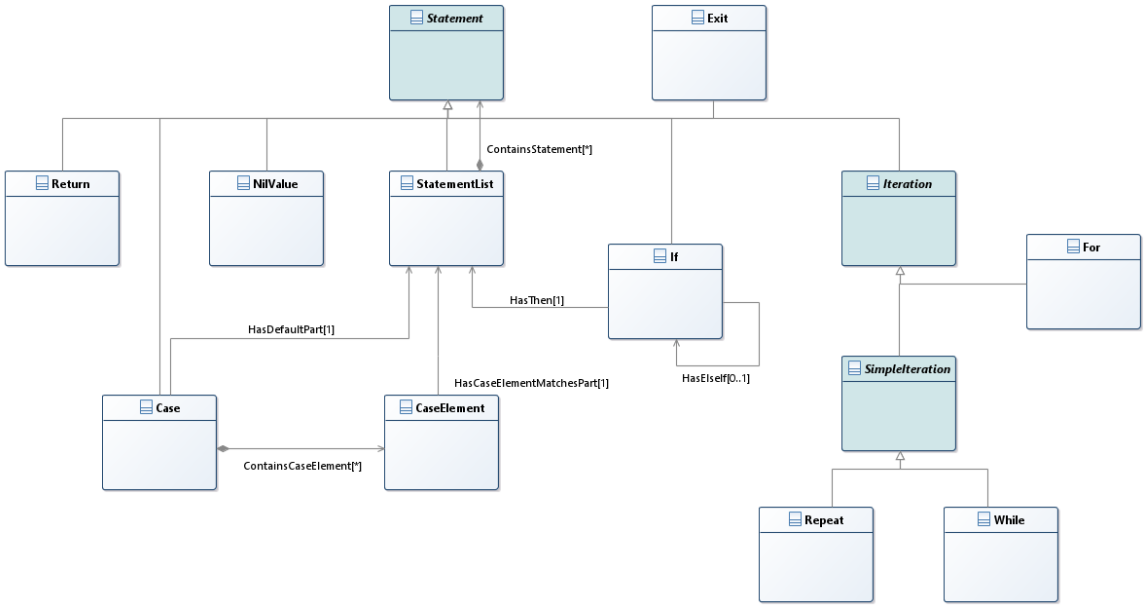

Abbildung 5.14.: Metamodell des strukturierten Textes [31]

Der strukturierte Text ist nicht nur die mächtigste Sprache im IEC 61131-3 Standard, sondern auch die zweite textuelle Sprache. Resultierend aus der Mächtigkeit des strukturierten Textes, auch bezogen auf das Metamodell, ist es die umfangreichste Sprache im Standard. Wie in anderen Programmiersprachen sind Kontrollstrukturen in Form von if-Abfragen möglich, aber auch Iterationen mithilfe von Schleifenkonstrukten. Ergänzt werden die Kontrollstrukturen von Case Elementen. Auch Rückgabewerte sind definiert sowie Ausstiegspunkte.

Mehr Details zum strukturierten Text gibt es im Abschnitt 2.2.3 des Grundlagenkapitels. Die Abbildung 5.14 zeigt das Metamodell der Sprache des strukturierten Textes.

6. Hardware-Software Korrelation

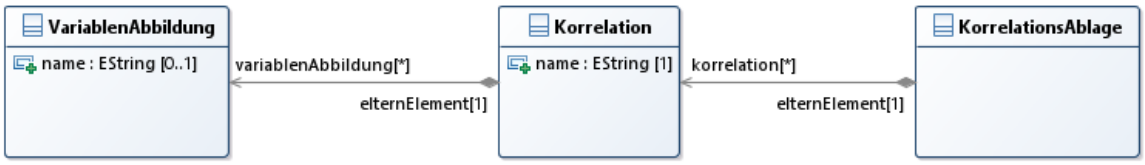

Abbildung 6.1.: Metamodell der Abbildung von Hardware auf Software

Wie Eingangs bereits beschrieben wird eine Hardware-Software Korrelation dafür benötigt, eine Relation zwischen der Hardware und der Software herzustellen. Das Metamodell der xPPU sowie das Metamodell des IEC 61131-3-Standards stehen in keiner direkten Verbindung. Das Metamodell der Hardware-Software Korrelation stellt eben diese Verbindung zwischen den beiden Metamodellen her, ohne sie zwangsweise zu koppeln. Damit ist die Möglichkeit gegeben, dass die beiden Metamodelle unabhängig von einander entwickelt werden können.

Um die Verbindung herstellen zu können, muss dem Metamodell der Hardware-Software Korrelation das Metamodell einer Anlage sowie das Metamodell der darauf eingesetzten Software zur Verfügung stehen. Dabei besteht die Möglichkeit, dass nicht nur ein Modell einer Anlage oder einer Software verwendet werden kann. Prinzipiell ist es aber empfehlenswert, dass nur eine Anlage pro Korrelationsmodell verwendet wird. Die Elemente des Hardware-Software Korrelationsmetamodelle werden in den folgenden Abschnitten erläutert.

6.1. Korrelationsablage

In der Korrelationsablage werden alle Korrelationen gespeichert, die der Nutzer im Modell anlegt. Dabei ist zu beachten, dass je Modellinstanz nur eine Korrelationsablage vorgesehen ist. Entsprechend sollte der Nutzer beim Erstellen des Modells die Korrelationsablage als Wurzelelement anlegen, um dieses im Modell vorzufinden.

6.2. Korrelation

Eine Korrelation beschreibt die Verbindungen eines Anlagenelements mit der Software. Im besten Fall legt der Entwickler ein Korrelationselement pro Anlagenelement an. Damit ist gewährleistet, dass alle entsprechenden Verbindungen eines Anlagenelements an

einer Stelle im Modell zu finden sind. Bei realen Projekten fördert dies die Übersichtlichkeit und erleichtert somit die Wartung. Ebenfalls kann die Wartbarkeit durch weitere Hardware-Software Korrelationsmodelle erhöht werden. Dies ist projektspezifisch und sollte entsprechend des Umfangs angelegt werden.

6.3. Variablen Abbildung

Wie in Abbildung 6.1 zu sehen, können in einer Korrelation s.g. Variablenabbildungen angelegt werden. In einer Abbildung wird eine Variable aus dem Programm einer Schnittstelle eines Anlagenelements zugewiesen. Die zugewiesene Variable muss global sein, ansonsten ist die Abbildung nicht möglich. Bei einer SPS werden so die Ein- und Ausgänge den entsprechenden Variablen im Code zugewiesen. Somit werden die möglichen Einstiegspunkte für die Änderungspropagation innerhalb der Software definiert. Bei der Änderungsausbreitungsanalyse besteht nun die Möglichkeit, anhand der Abbildung sowohl die entsprechende Variable und die dazu passende Repräsentation innerhalb der Anlage zu bestimmen, als auch die verschiedenen Programme zu identifizieren, die geändert werden müssen.

7. Metamodelle der Änderungspropagation

Dieses Kapitel stellt die Metamodelle vor, welche erstellt werden müssen, um das Framework dahingehend zu erweitern, damit die Änderungspropagation für aPS berechnet werden kann. Zum einen wird das Metamodell der Änderungsanfragen benötigt, um die initialen Änderungen im System zur Verfügung zu stellen. Somit dient es als Einstiegspunkt für den Anwender, die Änderungspropagation zu initiieren. Zum anderen werden im Architekturmodellanreicherungs-Metamodell die Artefakte definiert, die nicht direkt in der Anlage vorhanden sind, aber dennoch von einer Änderung betroffen sein können.

7.1. Änderungsanfragen

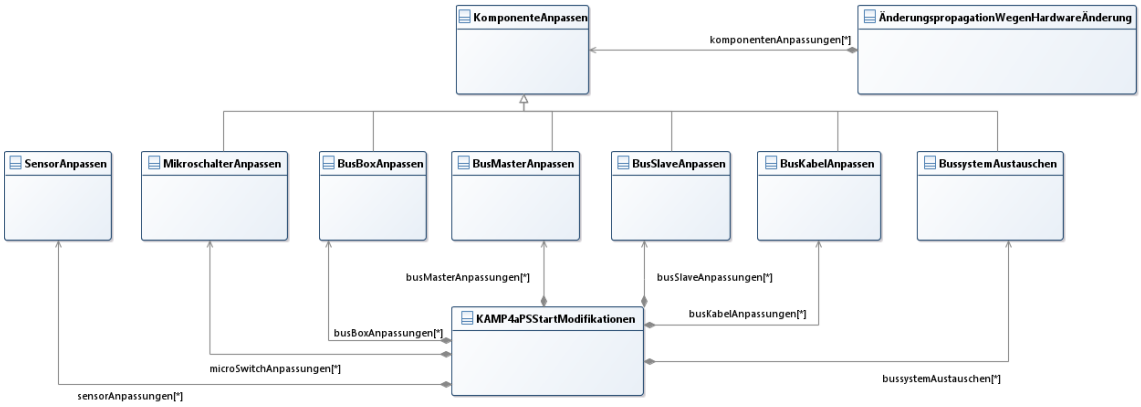

Abbildung 7.1.: Metamodell der Änderungsanfragen

Im Metamodell der Änderungsanfragen werden die Änderungen angegeben, die durchgeführt werden können. Dabei ist zu beachten, dass damit nur die Elemente der initialen Änderungen gemeint sind. Das bedeutet, der Entwickler muss die Elemente identifizieren, die als initiale Änderung in Frage kommen können. Im Falle der xPPU wären es prinzipiell erst einmal alle Komponenten, Module, Strukturen und Schnittstellen. Da wir uns aber zunächst auf zwei Szenarien beschränkt haben, ist die Menge der initialen Änderungen entsprechend geringer. Abbildung 7.1 liefert einen Überblick über die Struktur des von uns erstellten Änderungsanfragen-Metamodells. Zu sehen ist, dass als initiale Änderung zunächst Elemente des Bussystems und ein Mikroschalter[1] enthalten sind.

[1] ein binärer Schalter

Die Entität „Änderungspropagation wegen Hardwareänderungen“ dient als Container für die bei der Änderungspropagation gefundenen Elemente. Dabei ist zu beachten, dass bei dem Modell die Modifikationen nur bei bereits vorhandenen Elementen stattfinden können. Eine Modifikation im Sinne von Hinzufügen oder Entfernen eines Elements in der Anlage kann damit nicht abgebildet werden.

Auch wenn das über den Aufbau des Metamodells suggeriert wird, ist der Vorgang und das Hinzufügen oder Entfernen abzubilden wie folgt zu realisieren. Der Anwender muss von dem System, das analysiert werden soll, ein Modell erstellen. Die Basis dafür liefert das xPPU Metamodell. Das erstellte Modell stellt den Zustand der Anlage vor der Änderung dar. Daraufhin erzeugt der Anwender ein zweites Modell der Anlage. Im Idealfall wird das erste Modell kopiert, um Arbeit zu sparen. Das Modell stellt den Zustand nach dem Hinzufügen und Entfernen dar. Das Framework wird diese beiden Modelle abgleichen und anhand der Änderungen, die als initiale Änderungen registriert werden, eine Analyse der Änderungspropagation durchführen.

7.1.1. Initiale Änderungen

Eine Anlage besteht aus fünf Hauptklassen die modifiziert werden können. Aus diesem Grund existiert für jede dieser Klassen eine äquivalente Klasse als „*Klassenname* Anpassen“ im Metamodell der Änderungsanfragen. Diese Klassen dienen nicht dazu eine initiale Änderung anzugeben, sondern im Falle einer vorgesehenen Änderung, das entsprechende Element der Anlage zur Änderung zu markieren. Somit ist sichergestellt, dass grundsätzlich jedes Element einer Anlage zur Änderung markiert werden kann.

Zur initialen Änderung sind im vorliegenden Modell drei Klassen vorgesehen. Die Modifikation eines Sensors sowie eines Schalters (MikroSchalter) werden durch die Klassen „Sensor Anpassen“ sowie „Mikroschalter Anpassen“ repräsentiert. Beim Anlegen dieser Modifikationen kann der Nutzer aus der Menge der Sensoren bzw. Mikroschalter einen in der entsprechenden Modifikation als von der Änderung betroffenes Element angeben. Die passende Ausbreitungsregel vorausgesetzt, kann auf der Basis der initialen Änderung die Änderungspropagation berechnet werden.

„Bussystem Austauschen“ ist ebenfalls eine initiale Änderung, obwohl diese vom bisherigen Namensschema („Klassenname“ Anpassen) abweicht. Würde das „Austauschen“ ersetzt werden durch „Anpassen“ wäre die Art der Modifikation am Bussystem nicht eindeutig. Bei der Änderung „Bussystem Austauschen“ sollen die Änderungen berechnet werden, die notwendig sind, wenn ein Bussystem innerhalb einer Anlage durch ein anderes Bussystem ausgetauscht werden soll. Der Nutzer muss beim Anlegen dieser Änderung die Bus Box als zu änderndes Element angeben.

7.1.2. Änderungspropagation wegen Hardwareänderungen

Wie in der Einführung zu diesem Abschnitt bereits geschrieben, dient die Klasse *Änderungspropagation wegen Hardwareänderungen* zur Speicherung der zur Änderung vorgesehenen Elemente. Dabei ist zu beachten, dass die Elemente nicht sortiert werden und entsprechend in der Reihenfolge enthalten sind, wie diese abgespeichert werden. Duplikate werden nicht

automatisch entfernt, ist das gewünscht, muss der Entwickler entsprechende Vorkehrungen treffen.

Nach der Berechnung der Änderungspropagation dient die *Änderungspropagation wegen Hardwareänderungen* zum einen als Basis für die Erstellung der Aktivitäten und zum anderen werden auf der Basis der enthaltenen Elemente die Artefakte abgeleitet, die nicht in der Anlage enthalten sind aber dennoch von den Änderungen betroffen sind. Wie die Artefakte im Framework repräsentiert werden, wird im folgenden Abschnitt der *Architekturmodellanreicherung* dargestellt.

7.2. Architekturmodellanreicherung

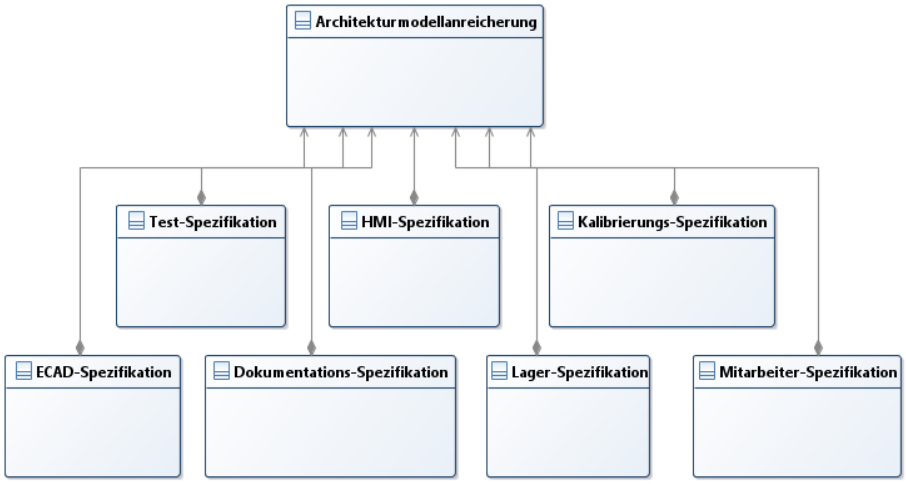

Abbildung 7.2.: Metamodell der Architekturmodellanreicherung

Die bisher skizzierten Metamodelle behandeln die Elemente, die direkt bei einer automatisierten Produktionsanlage zum Einsatz kommen. Somit ist die eingesetzte Hardware und die darauf laufende Software in den Prozess der Änderungen und Änderungsausbreitung eingebunden. Was noch fehlt sind die Artefakte, die nun nicht unmittelbar bei einer solchen Anlage zum Einsatz kommen, aber dennoch von Änderungen betroffen sein können. Abbildung 7.2 zeigt das Zusammenspiel der direkt betroffenen Artefakte und derer, die nur indirekt beeinflusst werden. Zu den indirekt betroffenen Artefakten zählen die Tests, die an der Anlage nach einer oder mehrerer Änderungen durchgeführt werden müssen, um die Funktionalität zu gewährleisten. Doch bevor Tests durchgeführt werden können, werden u. U. Komponenten kalibriert, da die Tests ansonsten fehlerhafte Werte liefern könnten.

Aber auch Änderungen an der Schnittstelle zum Benutzer der Anlage, die z. B. aus einer Änderung der Struktur der Anlage resultieren würde, ist Teil der Architekturmodellanreicherung. Ein wichtiger Teil, der bei fast jeder Änderung angepasst werden muss, ist die Dokumentation, sei es von der ganzen Anlage oder nur von einem Bauteil. Wird eine Struktur oder ein Modul geändert, besteht die Möglichkeit, dass die entsprechenden ECAD

Zeichnungen angepasst werden müssen. Auch die Lagerhaltung der im Moment nicht verbauten Elemente spielt ebenso mit in die Annotationen wie die Mitarbeiter, die die Änderungen durchführen müssen.

7.2.1. Test-Spezifikation

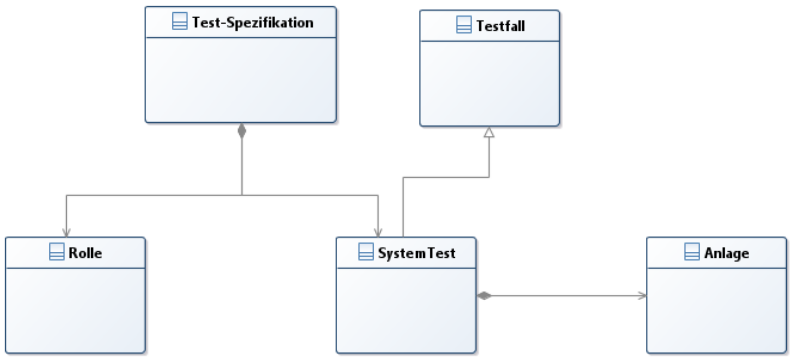

Abbildung 7.3.: Metamodell der Test-Spezifikation

Im Vergleich zur Softwareentwicklung sind in der Domäne der Automatisierungstechnik Tests weniger ausgereift. Das bedeutet, dass Unit Tests, Integration Tests usw. noch kaum Einzug in die Domäne der automatisierten Produktionssysteme gehalten haben. Nichtsdestotrotz sind Tests ein integraler Bestandteil. Die Hardwarekomponenten, die direkt vom Hersteller bezogen werden, bedürfen keiner weiteren Tests, bevor diese eingebaut werden. In dieser Hinsicht wird der Qualitätssicherung des Herstellers vertraut. Bei einem Defekt werden die Komponenten dann direkt ausgetauscht und nicht repariert.

Eigens hergestellte Module werden vor dem Einbau ebenfalls nicht auf Funktionalität überprüft, da dies bei der Fertigung geschieht und somit nicht nochmals durchgeführt werden muss. Daraus resultiert, dass während des Umbaus aufgrund von Änderungen, keine Tests durchgeführt werden. Nach dem Abschluss der Umbauarbeiten ist ein Systemtest vorgesehen, der die ganze Funktionalität der Anlage prüft. Wichtig ist daher, dass die Anlage wieder komplett zusammengebaut wurde und alle für diesen Umbau geplanten Änderungen erfolgreich durchgeführt werden konnten.

Abbildung 7.3 zeigt die Struktur der Test-Spezifikation. Jedem Test ist neben dem System, das getestet werden soll, zusätzlich noch eine Rolle zugeteilt. Diese gibt an, welcher Mitarbeitertyp, sei es nun ein Mitarbeiter aus der Fertigung oder ein Ingenieur, den Test durchführt. Ebenfalls ist es möglich, eine Mitarbeiterliste mit zugewiesenen Rollen vorausgesetzt, einem konkreten Mitarbeiter diese Aufgabe direkt zuzuweisen.

7.2.2. Kalibrierungs-Spezifikation

Wie bereits in der Einführung zu diesem Abschnitt geschrieben, kann die Notwendigkeit bestehen, dass eine modifizierte Komponente kalibriert werden muss. Am Beispiel der xPPU wäre der Kran zu nennen, der nach einer Modifikation, die daran durchgeführt wurde,

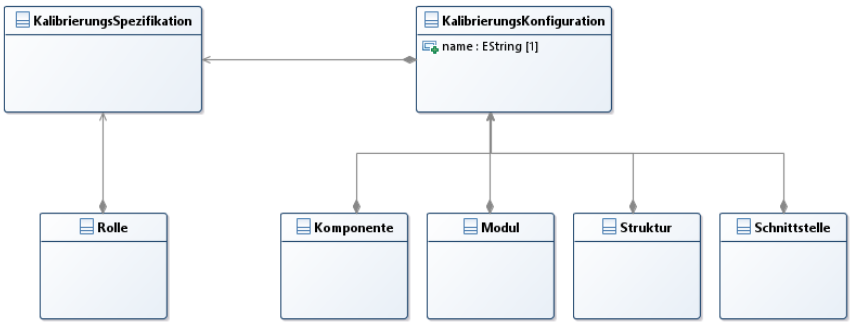

Abbildung 7.4.: Metamodell der Kalibrierungs-Spezifikation

an den anliegenden Förderbändern neu ausgerichtet werden muss. Die Notwendigkeit besteht, damit der Kran die Werkstücke ideal ansaugen und transportieren kann.

Abbildung 7.4 zeigt die Struktur der Kalibrierungs-Spezifikation. Neben der Rollenzuweisung wie bei der Test-Spezifikation enthält die Kalibrierungs-Spezifikation nun noch eine Konfigurationsklasse, in der die zu kalibrierenden Elemente hinterlegt werden. Somit ist gewährleistet, dass alle Elemente und für jede Rolle separate Aufgaben abgebildet werden können. Zur Differenzierung kann jeder Kalibrierungs-Spezifikation ein Name zugeteilt werden.

7.2.3. HMI-Spezifikation

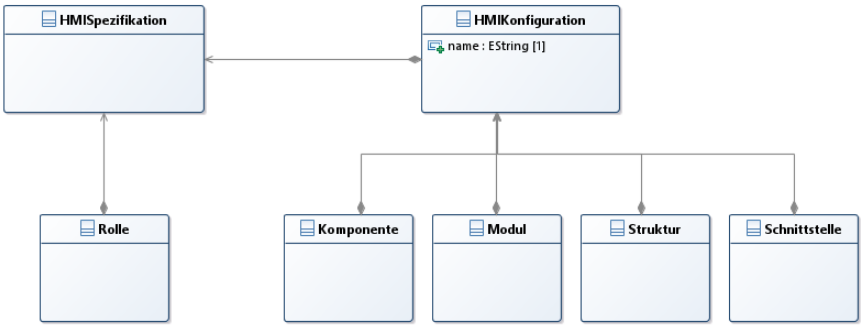

Abbildung 7.5.: Metamodell der HMI-Spezifikation

Die HMI-Spezifikation legt fest, welche Elemente eines automatisierten Produktionssystems eine Änderung an der Mensch-Maschinen-Schnittstelle provozieren. Der Zusammenhang wird in der HMI-Spezifikation abgebildet. Ebenso wie bei der Kalibrierungs-Spezifikation verfügt diese über eine Konfiguration, in der die Elemente, die die HMI beeinflussen, gekoppelt werden können.

Der Aufbau ist in Abbildung 7.5 dargestellt. Um die Änderung feingranular einzelnen Rollen und Mitarbeitern zuweisen zu können, ist es möglich, mehrere Spezifikationen mit dedizierten Rollen anzuglegen.

7.2.4. ECAD-Spezifikation

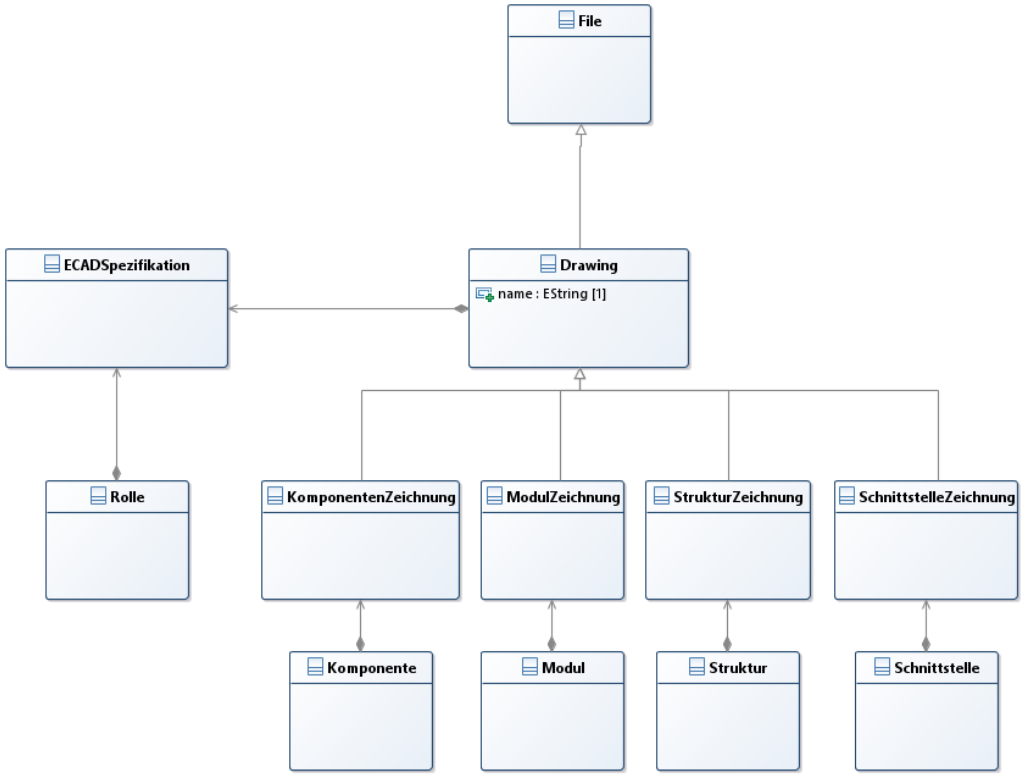

Abbildung 7.6.: Metamodell der ECAD-Spezifikation

In der ECAD-Spezifikation werden die Elemente definiert, deren Zeichnung(en) bei einer Änderung anzupassen sind. Dazu muss der Anwender im Modell festlegen, welche Komponenten über eine oder auch mehrere Zeichnungen verfügen. Das legt fest, dass bei jeglicher Art von Änderung die entsprechende(n) Zeichnung(en) zur Änderung angegeben wird bzw. werden. Da das Verfahren überapproximiert ist, werden auch die Zeichnungen mit einbezogen, die u. U. nicht von einer Änderung betroffen sind. Dem kann der Anwender entgegensteuern, indem entweder das Modell feingranular aufgebaut wird oder nicht relevante Elemente nicht mit einer Annotation versehen werden.

Abbildung 7.6 zeigt das Metamodell der ECAD-Spezifikation. Dabei ist zu beachten, dass generell wieder eine Rollenzuweisung vorhanden ist und dass es auch mehrere Spezifikationen geben kann. Innerhalb einer ECAD-Spezifikation werden einzelne Zeichnungen definiert, die wiederum an ein Element der Anlage gekoppelt sind. Somit kann der Anwender den Detailgrad abhängig von den Anforderungen bestimmen.

7.2.5. Dokumentations-Spezifikation

Die Dokumentation einer Anlage ist in vier Bereiche gegliedert. Aufgeteilt wird sie in die Dokumentation für die Komponenten, Module, Schnittstellen sowie Strukturen. Diese werden in den jeweiligen Gruppen abgelegt. Wird so in einer Gruppe für ein Element die

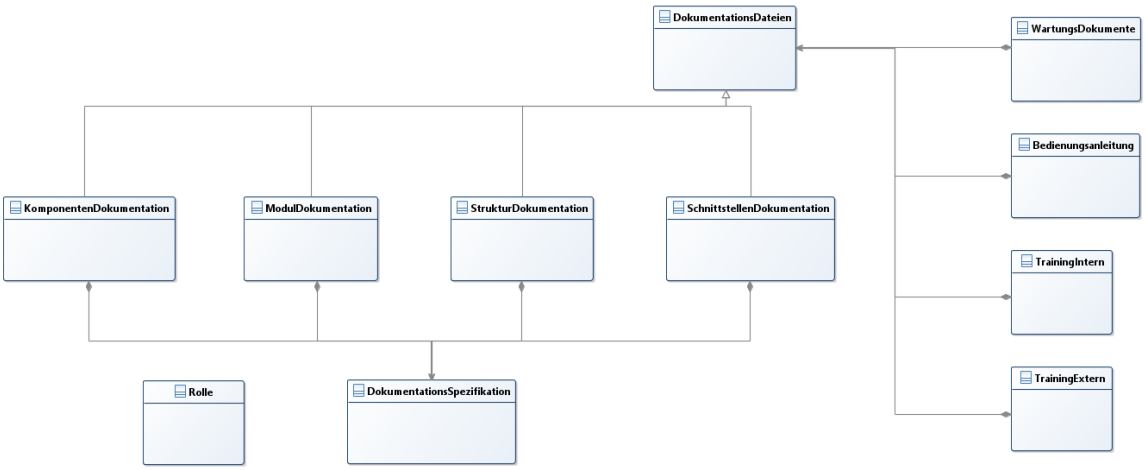

Abbildung 7.7.: Metamodell der Dokumentations-Spezifikation

Dokumentation angelegt, kann diese wiederum aus bis zu vier Teilen bestehen. Dabei ist zu beachten, dass nicht für jedes Element, jeder Dokumentationstyp vorhanden sein kann. Das entsprechend zu modellieren liegt in der Verantwortung des Domänenexperten, da dieser über die nötigen Kenntnisse verfügt.

Die Dokumentations-Dateien bestehen aus den Wartungsdokumenten, die alle nötigen Informationen zur Verfügung stellen, die notwendig sind, um das entsprechende Element zu warten. In der Bedienungsanleitung ist hinterlegt, wie das beschriebene Element zu handhaben ist. Darüber hinaus gibt es noch Trainingsdokumente. Diese ist unterteilt in firmeninterne Unterlagen sowie firmenexterne Unterlagen. Für die Dokumentation bzw. deren Aktuellhaltung wird eine Rolle definiert. Der eben geschilderte Aufbau ist in Abbildung 7.6 skizziert.

7.2.6. Lager-Spezifikation

Die Lager-Spezifikation repräsentiert die Lagerhaltung, die an die Anlage gekoppelt ist. Dabei ist zu beachten, dass nicht das komplette Lager abgebildet wird, sondern nur die Elemente, die in der Anlage vorkommen. Es dient dazu, nicht die Lagerhaltung abzubilden, sondern nur die Arbeit, die durch eine Änderung resultiert, aufzuzeigen. Verschiedene Lager können durch separate Spezifikationen abgebildet werden. Jede Spezifikation kann prinzipiell jeden Typus (Komponente, Modul, Stuktur, Schnittstelle) enthalten. Das ist aber nicht zwingend notwendig. Auch wird jeder Spezifikation eine Rolle zugewiesen, die für die Lagerhaltung zuständig ist. Abbildung 7.9 zeigt das Metamodell der Lager-Spezifikation mit dem eben beschriebenen Aufbau.

7.2.7. Mitarbeiter-Spezifikation

In erster Linie geht es bei der Änderungsvorhersage um das Abschätzen der Kosten einer Änderung. Neben den Tätigkeiten selbst, die in dieser Arbeit behandelt werden, spielt der

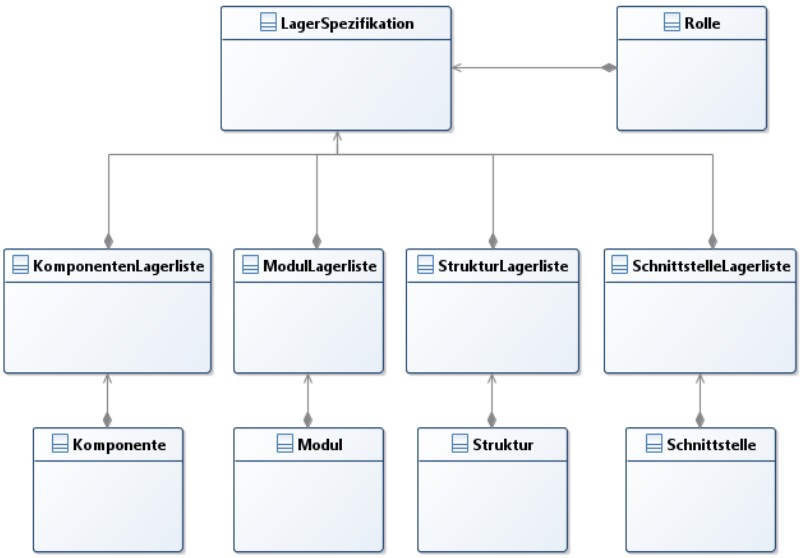

Abbildung 7.8.: Metamodell der Lager-Spezifikation

Aufwand, der für die Änderungen anfällt, ebenfalls eine entscheidende Rolle. Mit dieser Beschreibung wird der Grundstein zur Kostenschätzung gelegt, die jedoch nicht Gegenstand dieser Arbeit ist. In jedem Modell der Architekturmodellanreicherung wurde eine Rolle einer Spezifikation zugewiesen. In der Mitarbeiter-Spezifikation werden alle Mitarbeiter und deren Rollen definiert. Somit ist es möglich, später anhand der Aufwandsschätzung und den Rollen die Kosten für die Änderungen zu bestimmen.

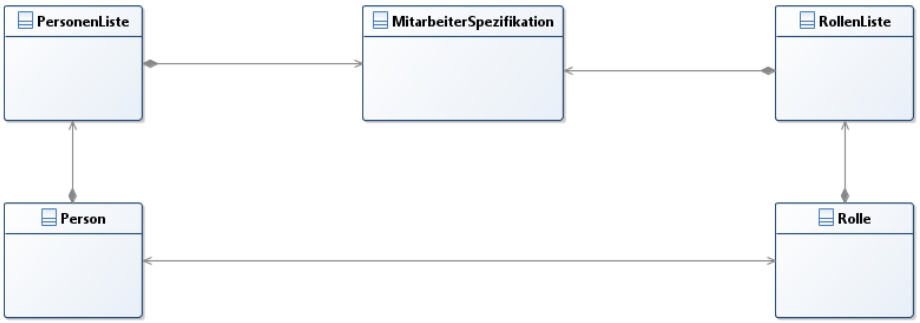

Abbildung 7.9.: Metamodell der Mitarbeiter-Spezifikation [67]

8. Regelwerk der Änderungsausbreitung

Neben den Modellen die bereits vorgestellt wurden, gilt es die Änderungsausbreitung zu definieren und umzusetzen. Als Basis dienen die Elemente, die im Änderungsanfragen-Metamodell unter „Anpassen“ eingetragen sind. Dazu zählt die Modifikation der Mikroschalter sowie die der Buskomponenten. Neben den Regeln zur Ausbreitung innerhalb der Hardware einer Anlage gilt es ebenfalls, die beeinflussten Artefakte zu bestimmen, die nicht direkt in der Anlage vorhanden sind. Die Regeln, wie die Änderungen durch das System propagiert werden, sind in den folgenden Abschnitten zu den Änderungen der jeweiligen Komponenten erläutert. Neben der Definition der Ausbreitungsregeln wird auch die Implementierung der Regeln behandelt. Dabei wird zusätzlich zu den Algorithmen auch deren Ablauf erörtert.

8.1. Änderung eines Sensors

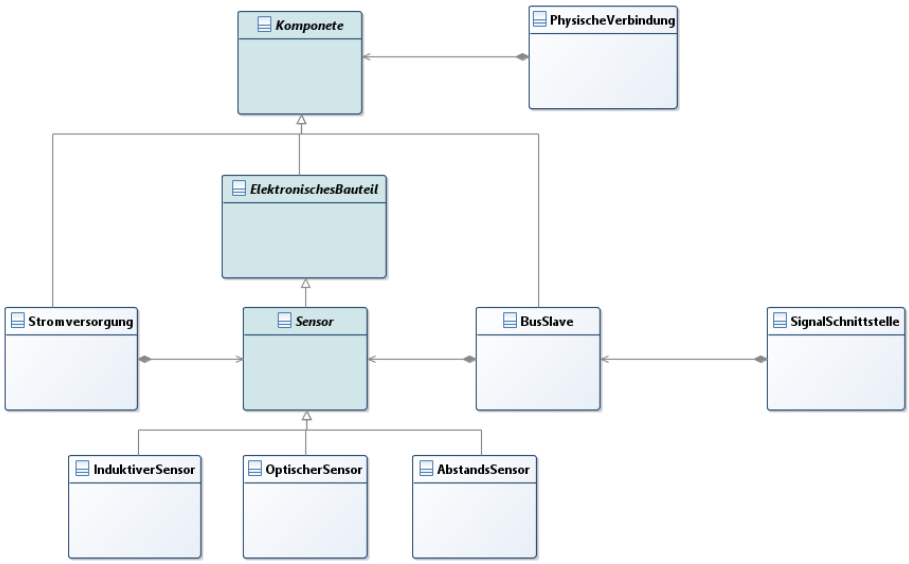

Abbildung 8.1.: Metamodell des Sensors [77]

Wie auch ein Mikroschalter ist ein Sensor ein Element, das einbaufertig bezogen wird und daher per Definition bei den Komponenten eingeordnet wird. Die Ausbreitungsregel bei einem Sensor unterscheidet sich dahingehend von denen des Mikroschalters, dass ein Sensor die Oberklasse verschiedener Sensortypen ist. Die Struktur eines Sensors ähnelt der des Mikroschalters. Daher sind die Sensoren Teil der elektronischen Komponenten.

Auch verfügen Sensoren über eine physische Verbindung, die eine Verbindung zur Basis darstellt, an der ein Sensor montiert werden kann. Zusätzlich verfügt ein Sensor über eine dedizierte Stromversorgung in Form eines Netzteils.

Die Relation zu einem Bus Slave stellt sicher, dass die Sensoren in das Bussystem eingebunden werden können. Abbildung 8.1 zeigt das Metamodell des Sensors und dessen Unterarten. Die Fixierung des Sensors an einer Basis ist implizit im Arbeitsschritt der Modifikation des Sensors bereits enthalten, wodurch eine Modifikation der Basis nicht zu den Arbeitsschritten hinzugefügt wird. Die Modifikation des Bus Slaves ist ebenfalls nicht enthalten, da davon ausgegangen wird, dass der korrekte Bus Typ bereits angebracht ist.

Hinzu kommt der Arbeitsschritt, bei dem der Sensor mit Strom versorgt werden muss. Unter Umständen muss die Stromversorgung angepasst werden. Im einfachsten Fall genügt es, dass der Sensor an bestehende Verbindungen gekoppelt werden kann. Der Ablauf ist im Algorithmus 8.1 dargestellt.

Algorithmus 8.1 : Änderungsausbreitung für Sensoren

Result : Create all Tasks based on changed Sensors

```
1 Load all ModifySensors;
2 while Sensors to modify do
3     create ModifySensor;
4     set Affected- and Causing-Elements;
5     create ModifyPhysicalInterface;
6     set Affected- and Causing-Elements;
7     create ModifyPowerSupply;
8     set Affected- and Causing-Elements;
```

8.2. Änderung eines Mikroschalters

Der Mikroschalter ist ein binärer Schalter. Da der Schalter als fertiges Element eingekauft wird, zählt er zu den Komponenten. Im Metamodell der xPPU erbt der Mikroschalter von der abstrakten Klasse Schalter, die wiederum ein elektronisches Bauteil darstellt. Somit verfügt der Mikroschalter über eine Signalschnittstelle das in einer Bus Slave Relation untergebracht ist. Die Signalschnittstelle dient dazu, die Komponente an den Bus anzuschließen. Außerdem muss der Mikroschalter noch an der Anlage befestigt werden. Dies geschieht mittels der physischen Verbindung.

Die Besonderheiten des Mikroschalters sind in Abbildung 8.2 illustriert. In der Grafik ist zu erkennen, dass es zwei Möglichkeiten gibt, über die Änderungen propagiert werden können. Nämlich sowohl über die physische Schnittstelle als auch über die Signalschnittstelle des Bus Slaves. Beim Mikroschalter wird vorausgesetzt, dass bereits die passende Schnittstelle verbaut ist, so dass die Bus Slave Komponente also zum Bussystem passt.

Die physische Schnittstelle des Mikroschalters stellt die Fixierung dar. Das bedeutet zwar, dass der Mikroschalter an einer anderen Komponente fixiert werden kann, der Arbeitsschritt aber bereits implizit in der Modifikation bzw. beim Hinzufügen/Entfernen

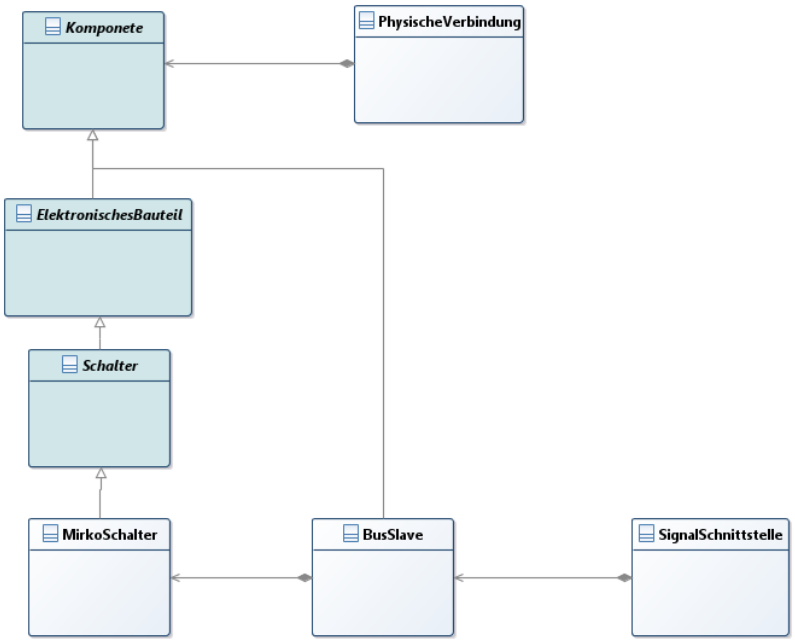

Abbildung 8.2.: Metamodell des Mikroschalters [77]

der Schnittstelle enthalten ist. Somit beschränkt sich die Änderung auf den Schalter und die entsprechende Fixierung. Der Algorithmus 8.2 zeigt den Ablauf zum Bestimmen und Erzeugen der Änderungen für alle Mikroschalters, die in einer Anlage geändert werden.

Algorithmus 8.2 : Änderungsausbreitung für Mikroschalters

```
   Result : Create all Tasks based on changed Microswitches
1  Load all ModifyMicroswitches;
2  while Microswitches to modify do
3      create ModifyMicroswitch;
4      set Affected- and Causing-Elements;
5      if Microswitch got a PhysicalInterface then
6          create ModifyPhysicalInterface;
7          set Affected- and Causing-Elements;
8      else
9          continue;
```

8.3. Änderung einer Buskomponente

Zur Kategorie der Buskomponenten gehört die Bus Box, der Bus Master, der Bus Slave sowie das Bus Kabel. Alle aufgezählten Komponenten sind immer Teil einer anderen Komponente. Es ist nicht vorgesehen, dass eine Buskomponente als Hauptelement instanziiert wird. Das Bus Kabel stellt hierbei eine Ausnahme dar, da es immer als eigenständige Komponente

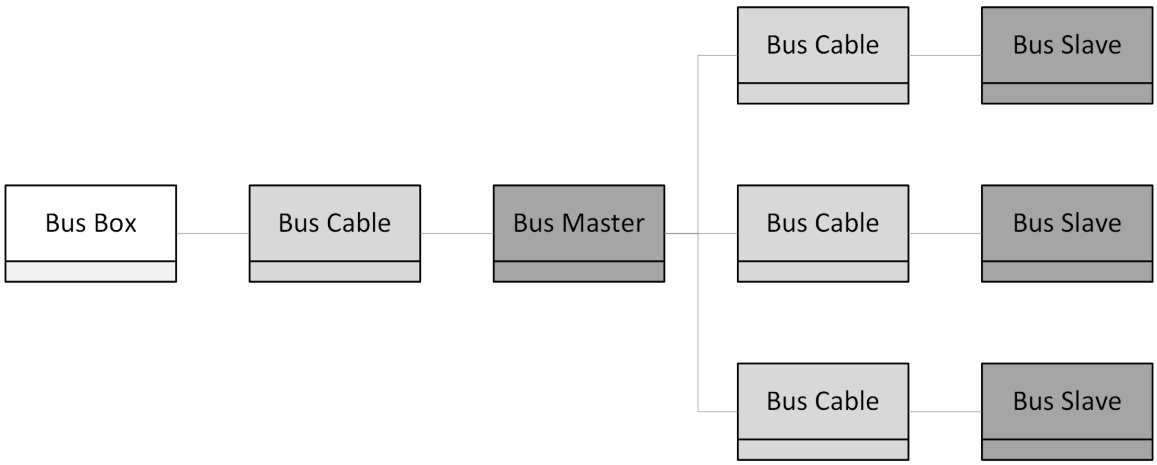

Abbildung 8.3.: Aufbau des Bussystems

instanziiert wird. Für die xPPU sind momentan zwei Bussysteme vorgesehen die zum Einsatz kommen können: Ether CAT und Profibus DP. Es besteht auch die Möglichkeit, dass innerhalb einer Anlage beide Systeme verwendet werden. Die Bus Box ist der zentrale Verteilerkasten an den die Busteilnehmer mithilfe der Bus Kabel verbunden sind. Eine Bus Box verfügt über eine begrenzte Anzahl an Anschlüssen. Im Moment spielt die Anzahl der maximal möglich angeschlossenen Busteilnehmer bei der Änderungsausbreitung noch keine Rolle. Es besteht auch die Möglichkeit, mehr als eine Bus Box zu verwenden, dabei werden die Boxen miteinander verbunden.

Der Bus Master, meist eine SPS, ist mit anderen Busteilnehmern verbunden. Er ist in der Lage Werte wie bei den zuvor gezeigten Sensoren und Schaltern abzufragen, aber auch Elemente der Anlage zu steuern. Diese sind als Bus Slaves realisiert, da sie entweder Informationen zum Zustand des Systems liefern oder den Zustand des Systems verändern.

Abbildung 8.3 illustriert vereinfacht die Zusammenhänge des Bussystems. Im Folgenden werden die Metamodelle der Buskomponenten vorgestellt und auf Basis dessen die Ausbreitungsregeln vorgestellt. Bei den Regeln ist zu beachten, dass davon ausgegangen wird, dass innerhalb der Anlage das Bussystem bzw. Subsystem ausgetauscht wird.

8.3.1. Änderung - Bus Box

Die Bus Box ist eine Art Verteilerkasten, an die über diverse Signalschnittstellen Buskomponenten angeschlossen werden können. Es wird differenziert zwischen der Verbindung zu einer weiteren Bus Box, einem Bus Master oder einem Bus Slave. Die Bus Box erbt von der Klasse „Component“ und ist im Paket der Buskomponenten untergebracht. Alle Buskomponenten sind mittels der Bus Kabel an der Bus Box angeschlossen. Zusätzlich zu den Signalschnittstellen verfügt eine Bus Box noch über eine dedizierte Stromversorgung.

Wie in Abbildung 8.4 zu erkennen, wird eine Änderung über die Signalschnittstellen propagiert. Änderungen an der Stromversorgung sind zu vernachlässigen und somit für die Änderungsausbreitung nicht relevant.

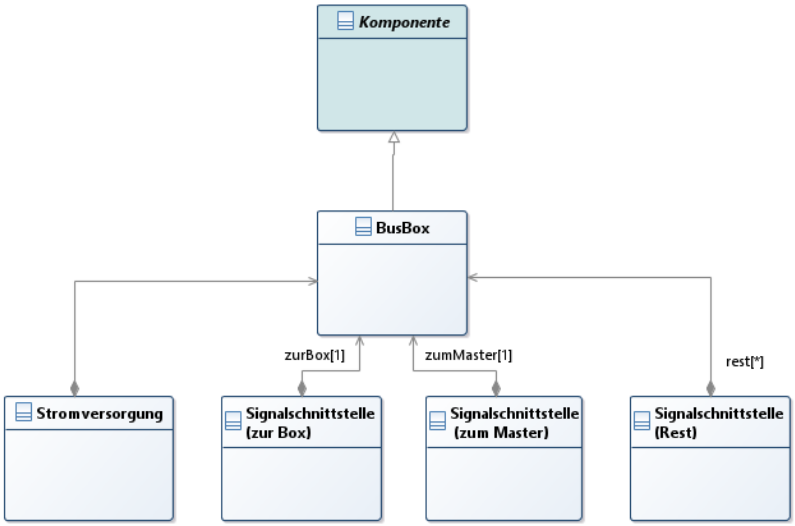

Abbildung 8.4.: Metamodell der Bus Box [77]

8.3.2. Änderung - Bus Master

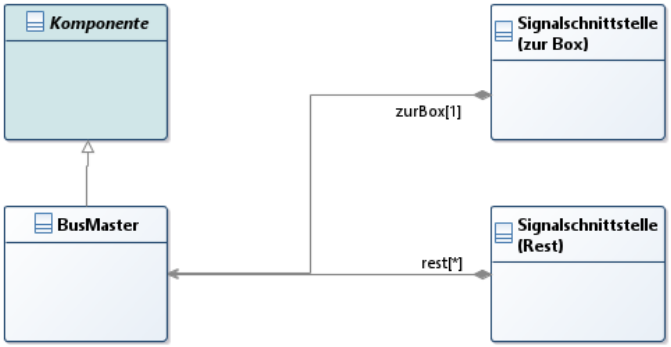

Abbildung 8.5.: Metamodell des Bus Master [77]

Der Bus mit seinen angeschlossenen Komponenten wird von einem Bus Master verwaltet. Dieser ist in der Lage Statuswerte anzufordern oder zu steuern. Aus diesem Grund sind Buselemente immer Teil anderer Komponenten und Module. Der Master ist über eine Signalschnittstelle mit der Bus Box verbunden. Über die restlichen Signalschnittstellen des Bus Masters sind Busteilnehmer verbunden.

Pro Signalschnittstelle kann nur ein Busteilnehmer verbunden werden. Das Hintereinanderschalten ist nicht möglich. Somit ist die maximale Anzahl an Elementen, die mit einem Master verbunden sein können, limitiert auf die maximale Anzahl an Signalschnittstellen. Bei einer Änderung sind also alle an den Master angeschlossenen Komponenten betroffen. Abbildung 8.5 zeigt das Metamodell des Bus Masters.

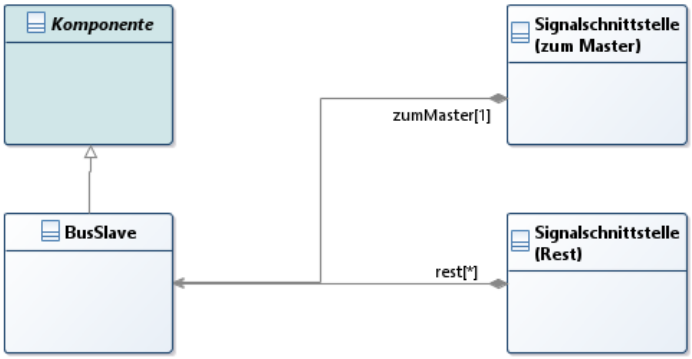

Abbildung 8.6.: Modell des Bus Slave [77]

8.3.3. Änderung - Bus Slave

Alle Teile einer Anlage müssen über den Bus angesteuert werden können. Das wird dadurch realisiert, dass eine Komponente oder ein Modul die Bus Slave Funktionalität implementiert. Eine Komponente oder ein Modul, das diese Funktionalität nicht implementiert, wäre vom System isoliert und somit nicht über den Bus steuerbar. Wäre das der Fall, ist dieses Element auch nicht von einer Änderung am Bussystem betroffen. Als Beispiel seien hier mechanische Komponenten genannt, die Teil eines Moduls sein können, aber für sich selbst gesehen keine Busanbindung benötigen.

Der Bus Slave kann von einem Bus Master über den Bus angesprochen werden, um z. B. aktuelle Statuswerte zu liefern oder um Befehle auszuführen. Ein Bus Slave ist über eine Signalschnittstelle mit einem Bus Master verbunden. Über die restlichen Signalschnittstellen können weitere Buskomponenten verbunden werden. Wie auch beim Bus Master sind bei einer Änderung des Bus Slaves alle nachfolgenden Elemente die über die Signalschnittstellen verbunden sind, betroffen. Bei der Ausbreitung der Änderungen spielt der Signalfluss keine Rolle.

8.3.4. Bestimmung der Aufgaben bei einer Änderung am Bussystem

Ziel des Algorithmus ist es, alle Änderungen innerhalb einer Anlage zu finden die durch eine Initialänderung durch eine Buskomponente erfolgen können. Dabei werden alle Komponenten aus der Anlage benötigt, um daraus alle Buskomponenten herausfiltern zu können. Die Komponenten sind in der Komponentenablage hinterlegt (siehe dazu Abschnitt 5.1.4). Ist die Initialisierung abgeschlossen, werden die Schnittstellen der initial markierten Buskomponenten geladen und es wird überprüft, welche Kabel mit dieser Schnittstelle verbunden sind. Die passenden Kabel werden zwischengespeichert. Daraufhin wird über jede Bus Box iteriert und es wird überprüft, ob die Bus Box mit einem der Kabel verbunden ist.

Ist dies der Fall, wird die Bus Box zur Liste der zu modifizierenden Komponenten hinzugefügt und der Marker gesetzt, dass mindestens ein neues Element in der zu modifizierenden Liste vorhanden ist. Das selbe Prozedere wird dann für jeden Bus Master und jeden Bus Slave im System durchgeführt. Nachdem alle Buskomponenten überprüft wurden, werden

Algorithmus 8.3 : Änderungsausbreitung für Buskomponenten

```
Result : Look Up Changes Based On Bus Modification
Load all Componentes of Plant;
Filter all Bus Components;
while there are new modifications do
    Load all Interfaces of the Bus Cables that are
    connected to the marked Bus Components;
    for all bus boxes in the system do
        if bus box shares interface with cable then
            add box to modify list;
            set marker for new modifications;
        else
            continue;
    for all bus masters in the system do
        if bus master shares interface with cable then
            add master to modify list;
            set marker for new modifications;
        else
            continue;
    for all bus slaves in the system do
        if bus slave shares interface with cable then
            add slave to modify list;
            set marker for new modifications;
        else
            continue;
    remove duplicates;
```

mehrfach vorkommende Elemente auf einen Eintrag reduziert. Das ist notwendig, da z. B. mit einem Bus Master mehrere andere Busteilnehmer verbunden sein können und der Master somit für jede der angeschlossenen Komponenten als Treffer markiert wird. Der Prozess wird so lange wiederholt, bis keine neuen Komponenten zur Modifikation hinzukommen.

Ausgehend von einer Komplexität des EMF Frameworks von $O(1)$ ergibt sich eine generelle Komplexität von $O(n)$ für das Überprüfen einer jeden Komponente, bei der n gleich die Anzahl der Elemente in der Anlage ist. Die Komplexität für das Betrachten der modifizierten Komponente ist $O(m)$, wobei m gleich alle modifizierten Komponenten ist. Im Worst Case wird jede Komponente modifiziert, wodurch sich eine Gesamtkomplexität von $O(m * n)$ bzw. $O(n^2)$ ergibt.

9. Evaluation

In diesem Kapitel wird die Evaluation des Verfahrens zur architekturbasierten Bewertung und Planung von Änderungsanfragen in aPS behandelt. Dabei wurde ein szenariobasierter Ansatz gewählt, der es ermöglichen soll, die Umsetzung von KAMP4aPS zu evaluieren. Bevor die Modelle sowie Ergebnisse behandelt werden, wird die gewählte Evaluationsmethode aufgezeigt. Abschließend werden die Ergebnisse interpretiert und die möglichen Einschränkungen durch die getroffenen Annahmen erörtert.

9.1. Evaluationsmethode

Dieser Abschnitt beschäftigt sich mit der Evaluationsmethode der vorgestellten Konzepte. Dazu verfolgen wir den Ansatz von Böhme und Reussner[12], bei dem drei Typen der Validierung für Methoden der Vorhersage behandelt werden. Die Validierung ist so aufgebaut, dass zuerst die Machbarkeit (Typ 1) untersucht wird. Dazu wird das Ergebnis, im vorliegenden Fall die automatisch erzeugte Aufgabenliste, mit einer manuell erstellten Aufgabenliste verglichen. Anschließend soll die Durchführbarkeit (Typ 2) überprüft werden, um abschließend den Aufwand/Nutzen (Typ 3) zu analysieren. Die zu Beginn dieser Arbeit gestellte Frage, ob das KAMP-Framework auf Automatisierungssysteme angewendet werden kann, ist mit der Analyse der Machbarkeit zu überprüfen. Die Analyse zur Durchführbarkeit und zum Aufwand/Nutzen ist nicht Teil dieser Arbeit, wird aber im Rahmen der KAMP4aPS Entwicklung zu einem späteren Zeitpunkt durchgeführt, und zwar, wenn eine Grundmenge an Szenarien umgesetzt wurde.

Um die Machbarkeit zu untersuchen, haben wir aus einer Sammlung von Änderungsszenarien zwei ausgewählt. Die Sammlung wurde von Vogel-Heuser et al. angefertigt[78]. Darin werden insgesamt 13 Szenarien beschrieben, die die Evolution der xPPU darstellt. Dabei bauen die Szenarien aufeinander auf, sind in der Ausführung also nicht in beliebiger Reihenfolge anwendbar. Abbildung 9.1 zeigt den Ablauf der Szenarien wie er in [78] beschrieben wird. Über die Vollständigkeit der Evolution der Produktionsanlage lässt sich keine Aussage treffen. Die verschiedenen Iterationen spielen für diese Arbeit keine Rolle. Die Änderungen können u. U. einen anderen Verlauf durch das System nehmen, wenn Komponenten ausgetauscht oder hinzugefügt wurden. KAMP4aPS betrachtet den Ist-Zustand des Systems bzw. die Modellierung des Systems. Dadurch ist es nicht möglich vorherige Iterationen zu betrachten. Soll dies dennoch geschehen, muss für jede Iteration ein Modell erstellt werden, so dass die Änderungspropagation auch dort betrachtet werden kann. Im nachfolgenden Abschnitt 9.2 sollen die beiden Szenarien vorgestellt werden.

Auf Basis der Szenarien wird manuell je eine Aufgabenliste erzeugt. Mit KAMP4aPS wird dann ebenfalls für die Szenarien jeweils eine Aufgabenliste erzeugt. Diese beiden werden

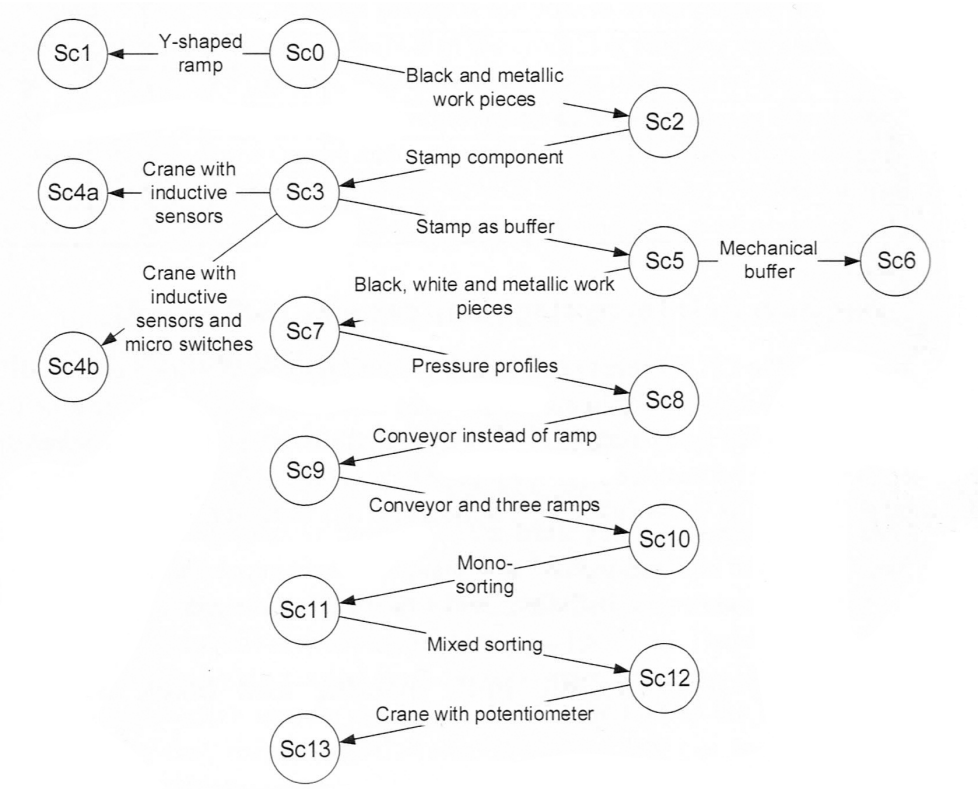

Abbildung 9.1.: Abfolge der Szenarien [78]

verglichen, um eine Aussage über die Qualität der automatisch erzeugten Aufgabenliste treffen zu können. Evaluiert wird das Vorhaben mithilfe des Goal Question Metric (GQM) Plans. Wie bereits beschrieben, ist das Ziel (Goal) die Qualität der automatisch erzeugten Aufgabenliste zu bestimmen. Die Frage (Question) ist zum einen, ob die erzeugte Liste komplett ist, und zum anderen, wie genau sie ist. Als Metrik (Metric) haben wir das harmonische Mittel (F_1) auf Basis der Recall- und Precision- Werte gewählt. Diese zwei Metriken setzen auf dem Vergleich einer automatisch erzeugten und der entsprechenden manuell erzeugten Aufgabe auf. Sind die Aufgaben identisch, ist es ein True Positive (TP); wird eine Aufgabe generiert obwohl diese nicht existieren sollte, entspricht das einem False Positive (FP). Ein False Negative (FN) ist, wenn eine Aufgabe hätte generiert werden sollen, aber in der Aufgabenliste fehlt. Die positive Rückmeldung (Recall) beschreibt den Anteil der als korrekt positiv erkannten Aufgaben. Daher wird der Recall auch True Positive Rate (TPR) genannt.

$$TPR = Recall = \frac{TP}{TP + FN}$$

Um nicht nur die richtig erkannten Aufgaben in die Bewertung einfließen zu lassen, wird ebenfalls noch die Genauigkeit (Precission) der Aufgabenliste betrachtet. Dazu wird ermittelt, wie viele korrekt benannte Aufgaben aus der Menge aller richtig erkannten Aufgaben es gibt.

$$Precision = \frac{TP}{TP + FP}$$

Die F_1-Metrik kombiniert Precision und Recall mit Hilfe des harmonischen Mittels wie folgt, um die Güte mittels einer Kennzahl zu bestimmen:

$$F_1 = 2 \cdot (Recall \cdot Precision)/(Recall \cdot Precision)$$

9.2. Szenarioentwürfe

Zunächst wird aufgezeigt, wie die zwei Szenarien im Kontext der xPPU definiert wurden. Dazu ist es notwendig, den Zustand der xPPU zu Beginn des Szenarios zu beschreiben. Dadurch werden Zusammenhänge zwischen den Komponenten besser verständlich. Die Vorstellung der Szenarien geschieht in den Abschnitten 9.2.1 und 9.2.2. Daraufhin werden Komponenten, die nicht für das Szenario benötigt werden, entfernt. Das vereinfachte Modell dient als Basis für die Evaluation des jeweiligen Szenarios. Zusätzlich muss zum Modell der Anlage ein Modifikationsmodell, ein Annotationsmodell, ein Modell der Software sowie ein Verteilungsmodell erstellt werden.

9.2.1. Schaltertausch-Szenario

Das hier beschriebene Szenario 1 entspricht dem 13. Szenario in [78]. Die Produktionsanlage enthält für dieses Szenario sowohl einen Stapel (Stack), der dazu dient, die Werkstücke zu lagern, als auch ein Förderband (Conveyor), um die Werkstücke zu transportieren. Außerdem verfügt die Anlage noch über einen Stempel (Stamp), um Werkstücke zu stempeln und einen Kran (Crane), der die Werkstücke zwischen den Arbeitsstationen bewegen kann. Die Bewegung der Werkstücke erfolgt mittels anheben und platzieren (pick and place). Abbildung 9.2 zeigt den Aufbau und die Anordnung der eben beschriebenen Elemente.

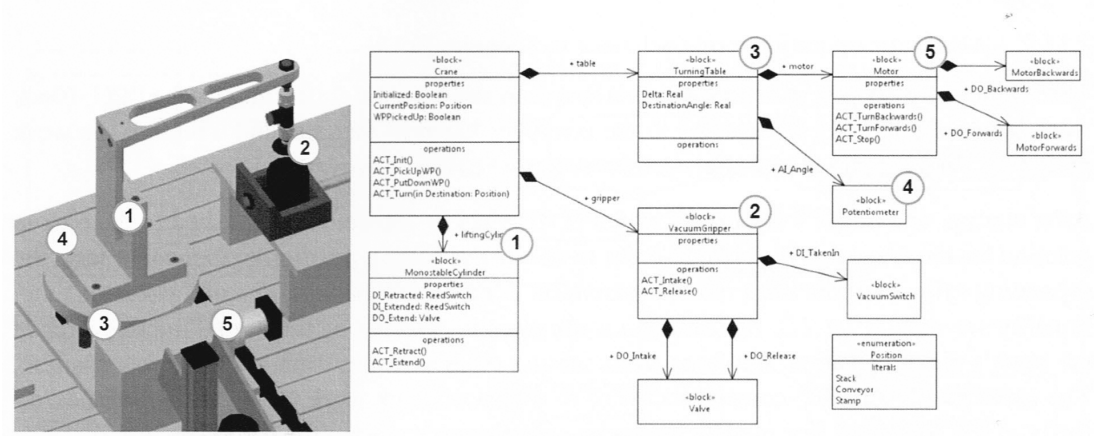

Abbildung 9.2.: Modell der PPU für Szenario 1 [78]

Als Schnittstelle zum Menschen (HMI) verfügt die Produktionsanlage über einen Start- und Notausschalter (Button). Die PPU wird schwarze und weiße Werkstücke aus Plastik sowie metallische Werkstücke verarbeiten. Schwarze Werkstücke sollen direkt auf das Förderband gelegt werden, wohingegen die weißen und metallenen Werkstücke zuerst

zum Stempel gebracht werden, um nach der Stempelung auf das Förderband gelegt zu werden. Abhängig vom Material wird ein passendes Stempelungsprofil für den Druck ausgewählt.

Der Kran ist auf einem Drehtisch (Turning Table) angebracht. Darunter befinden sich drei binäre Schalter (Micro Switch). Die Schalter dienen dazu, die Position des Krans zu bestimmen. Für die drei möglichen Positionen (Stapel, Förderband und Stempel) ist ein Schalter angebracht, der ausgelöst wird, wenn sich der Kran im richtigen Winkel zur jeweiligen Position befindet. Diese drei Schalter sollen nun durch ein Potentiometer ersetzt werden. Durch das Potentiometer wird die Position des Krans nun nicht mehr durch drei diskrete Werte repräsentiert. Nach dem Einbau des Potentiometers wird die Position durch einen kontinuierlichen Wert dargestellt.

Der Austausch der Schalter durch das Potentiometer erfolgt in zwei Schritten. Zunächst müssen die Schalter entfernt werden. Davon können andere Komponenten betroffen sein. Ebenfalls hat der Wegfall der Komponenten Auswirkungen auf die SPS und die darauf laufende Software. Im zweiten Schritt wird das Potentiometer eingebaut und angeschlossen. Das kann wiederum Auswirkungen auf andere Komponenten haben. Da das Potentiometer an den Bus angeschlossen wird, hat die Änderung auch Auswirkungen auf die SPS und die Software.

9.2.1.1. Änderungen an der Hardware

Die Änderung betrifft also die drei Schalter, die entfernt werden müssen, sowie die Komponente, an der sie angebracht sind. Gesetz dem Fall, dass die Schalter gut zugänglich sind, werden der Drehtisch sowie der Kran nicht von der Änderung betroffen. Im schlechtesten Fall aber muss der Drehtisch samt Kran abmontiert werden. Da die Schalter komplett entfernt werden sollen, ist deren Fixierung (Physical Interface) auch von der Änderung betroffen und muss ebenfalls entfernt werden. Der Stapel sowie der Stempel und das Förderband müssen nicht angepasst werden, wenn die Schalter entfernt werden. Nachdem die Schalter entfernt wurden, muss das Potentiometer angebracht und fixiert werden. Ist das Potentiometer eingebaut muss u.U. wie bei den Schaltern auch, der Drehtisch samt Kran wieder montiert werden. Tabelle 9.1 zeigt die durchgeführten Operationen auf Basis der Komponenten. Außerdem werden die ebenfalls betroffenen Komponenten aufgelistet.

Operation	**Komponente**	**Betr. Elemente**
Entfernen	Schalter (Stapel)	Fixierung, ggf. Drehtisch und Kran
	Schalter (Förderband)	Fixierung, ggf. Drehtisch und Kran
	Schalter (Stempel)	Fixierung, ggf. Drehtisch und Kran
Hinzufügen	Potentiometer	Fixierung, ggf. Drehtisch und Kran

Tabelle 9.1.: Änderungen in Szenario 1

9.2.1.2. Sonstige Änderungen

Neben den offensichtlichen Änderungen der Komponenten der Anlage sind noch weitere Elemente von den Änderungen betroffen. Dazu zählen die in den Annotationen ange-

gebenen Relationen, die vom Domänenexperten modelliert werden müssen. Tabelle 9.2 zeigt anhand eines Schalters, welche Annotationen zugewiesen werden können und die im Falle einer Änderung angepasst werden müssen. Unterteilt sind die Annotationen in komponentenspezifische, z. B. die Bedienungsanleitung und allgemeine Informationen, z. B. zu den Mitarbeitern und deren Rollen. Komponentenspezifische Annotationen sind nur für eine bestimmte Komponente von Bedeutung, wohingegen allgemeine Annotationen nur einmal definiert werden müssen. Einer Rolle können mehrere Aufgaben, auch Komponentenübergreifend, zugewiesen werden. Die Rolle des Ingenieurs z. B. ist zuständig für das Erstellen und Aktualisieren der Dokumentation sowie auch für das Erzeugen und Warten der ECAD Zeichnungen.

Gekoppelte Komponente	**Annotation**
Schalter	> Test Spezifikation
	=> System Test
	> Dokumentation
	=> Wartung
	=> Bedienungsanleitung
	=> Trainingsdokumente (intern)
	=> Trainingsdokumente (extern)
	> Kalibrierung
	=> Kalibrierung der Komponente
	> ECAD Zeichnungen
	=> Zeichnung der Komponente
	> Software
	=> Update Software
Allgemein	> Mitarbeiter
	=> Rollenliste
	==> Ingenieur
	==> Einkauf
	==> Montage
	=> Mitarbeiterliste

Tabelle 9.2.: Annotationen für die Szenarien

Da die Änderung der Schalter auch in der Software eine Änderung provoziert, werden zusätzlich die Modelle für das Programm sowie das Hardware-Software Korrelation betrachtet. Wie in Abschnitt 5.2 bereits dargelegt, kann ein Programm ganzheitlich modelliert werden. Die Analyse betrachtet die öffentlichen Variablen, die den Ein- und Ausgängen der SPS entsprechen. Im Modell der Hardware-Software Korrelation werden die physischen Ein- und Ausgänge den Variablen im Modell der Programme zugewiesen. Pro Schalter wird eine Signalschnittstelle verwendet, die mit dem Bus verbunden ist. Die Schnittstelle wird auf eine Variable im Programm abgebildet. Somit wird das Programm zur Änderung vorgemerkt, da diese drei Variablen beim Entfernen der Schalter nicht mehr verwendet

werden. Durch das Hinzukommen des Potentiometers, das ebenfalls über eine Signalschnittstelle mit dem Bus verbunden ist, muss als Aktivität auch das Programm abgeändert werden. Abbildung 9.3 zeigt beispielhaft wie der Kontext zwischen einem Schalter und einem Programm realisiert ist.

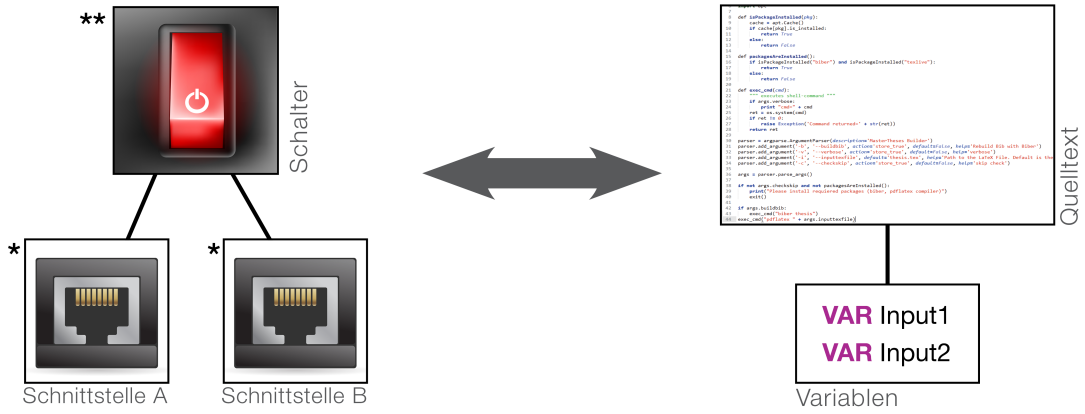

Abbildung 9.3.: Hardware-Software Korrelation für Szenario 1

9.2.2. Buswechsel-Szenario

Im zweiten Szenario wird der Wechsel des Bussystems thematisiert. Innerhalb einer Anlage können mehrere Busse zum Einsatz kommen. Auch ist es möglich, dass es verschiedene Bussysteme sein können. So kann zum einen der Profibus und zum anderen EtherCAT verwendet werden. Das vorliegende Szenario sieht vor, dass in einem Busnetz das Bussystem gewechselt wird. Dabei müssen im schlechtesten Fall alle Buselemente getauscht werden. Dazu zählen die Bus Boxen, die Bus Master und die Bus Slaves. Auch kann es sein, dass die Buskabel ersetzt werden müssen. Es besteht aber auch die Möglichkeit, dass die Komponenten bestehen bleiben können und nur die Software angepasst werden muss. Abhängig ist das von der aktuellen Struktur im System, aber auch vom eingesetzten Bussystem und dem Bussystem, auf das gewechselt werden soll. Der Aufwand ist somit schwer abzuschätzen und ist in allen Nuancen nicht mit vertretbarem Aufwand umzusetzen. Aus diesem Grund haben wir beschlossen, dass wenn ein Buswechsel erfolgen soll, alle Buselemente in die Aufgabenliste eingetragenen werden und der Anwender im zweiten Schritt dann festlegen muss, welche Aufgaben nicht nötig sind.

* © puruan / Fotolia.com
** © SM Web / Fotolia.com

9.2.2.1. Änderungen an der Hardware

Buskomponenten sind Teil eines Moduls oder einer Struktur. Bei der Änderungsausbreitung werden diese nicht markiert, da bereits das innere Element zur Änderung hinterlegt wird. In diesem Fall ist es die Buskomponente, also entweder die Bus Box, der Bus Master oder der entsprechende Bus Slave. Die Szenarien in [78] und in [80] können alle implizit auf den Buswechsel übertragen werden, da jedes Modul und jede Struktur davon betroffen sein können.

Es besteht die Notwendigkeit, dass bei einem Buswechsel alle Buskomponenten, die an den zu ändernden Bus angeschlossen sind, identifiziert werden. Der genaue Vorgang ist in 8.3.4 beschrieben. Der Algorithmus 8.3 beschreibt wie alle Buskomponenten gefunden werden können. Im folgenden Abschnitt 9.3.2 stellen wir das der Evaluation für Szenario 2 zugrundeliegende Modell vor.

9.2.2.2. Sonstige Änderungen

Die Annotationen sind wie im vorherigen Szenario vom Domänenexperten zu bestimmen. Die möglichen Annotationen stimmen mit denen aus Szenario 1 überein. Dazu zählt die Anpassung der Dokumentation, die Kalibrierung der geänderten Komponenten, die Anpassungen der ECAD Zeichnungen sowie der abschließende Systemtest, der gestartet wird, nachdem alle Änderungen durchgeführt wurden. Die Liste der Mitarbeiter und deren Rollen muss nicht verändert werden, da wir davon ausgehen, dass es sich um eine Anlage im selben Unternehmen handelt. Die Annotationen sind in Tabelle 9.2 zu finden.

9.3. Modellentwürfe

In diesem Abschnitt werden die Modelle vorgestellt, die wir für die Evaluation erstellt haben. Zu jedem Szenario werden die sechs Modelle besprochen, die notwendig sind, um die Änderungsausbreitungsanalyse durchführen zu können. Dazu zählen zwei Modelle einer Anlage, die nur hinreichend vollständig sein müssen. Komponenten die sicher nicht von der Änderung betroffen sein werden, können zum besseren Verständnis ausgespart werden. Weiterhin wird ein Modell der Programme sowie der Hardware-Software Korrelation benötigt. Wie schon beim Modell der Anlage gilt auch hier, dass die Modelle nicht alle Programme und Kontexte abbilden müssen. Das Annotationsmodell muss, sofern vorgesehen, jedes Element aus der Anlage behandeln. Initiale Änderungen werden im Modifikationsmodell hinterlegt. Aber auch wenn keine initialen Änderungen vorliegen, ist dieses Modell obligatorisch, da zum einen der Analyseprozess über dieses Modell gestartet werden muss und zum anderen sich Differenzen zwischen den zwei Anlagenmodellen ergeben können. Das ist gerade im ersten Szenario der Fall. Zur besseren Übersicht werden die Szenarien getrennt behandelt.

9.3.1. Testanlage - Schaltertausch

Wie in 9.3 bereits besprochen, enthält dieser Abschnitt die Modelle für das erste Szenario. Der Stapel sowie der Stempel und das Förderband sind nicht von den Änderungen am

Kran betroffen. Aus diesem Grund wurde beschlossen, diese Elemente nicht in das Modell aufzunehmen. Auch ist der Kran nicht vollständig modelliert, da z.B. der Arm und der Greifer keine Abhängigkeiten haben zur Positionsbestimmung die mittels der Schalter bzw. nach der Änderung mit dem Potentiometer erfolgt. Die Beschreibung beginnt mit dem Modell der Anlage (9.3.1.1) und dem Annotationsmodell (9.3.1.2). Dabei wird der Aufbau der Anlage erläutert sowie die Annotationen dargestellt. Im Modifikationsmodell (9.3.1.5) wird auf die möglichen Initialmodifikationen eingegangen. Das Programm- (9.3.1.3) und Auslieferungsmodell (9.3.1.4) beschreibt die Programme und den dazu nötigen Auslieferungskontext.

9.3.1.1. Anlagenmodell

Für das erste Szenario wird mindestens einen Kran, an dem die Änderung vorgenommen wird benötigt. Da die Schalter nicht direkt mit dem Kran verbunden sind, sondern ein Teil des Drehtisches sind, muss auch dieser modelliert werden. Das Szenario sieht vor, dass der Drehtisch über drei Schalter verfügt, die jeweils in Richtung des Stapels, des Stempels und des Förderbandes ausgerichtet sind. Abbildung 9.4 illustriert schematisch den Aufbau und die Anordnung. Da es keine Abhängigkeit gibt zwischen dem Kran und den drei Stationen, ist es nicht notwendig, diese zu modellieren. Die Abhängigkeit bezieht sich nur auf die Änderungsausbreitung in diesem speziellen Szenario und gilt nicht für die reale Anwendung. Dort muss die ganze Anlage modelliert werden, um die bestmögliche Abschätzung zu erhalten.

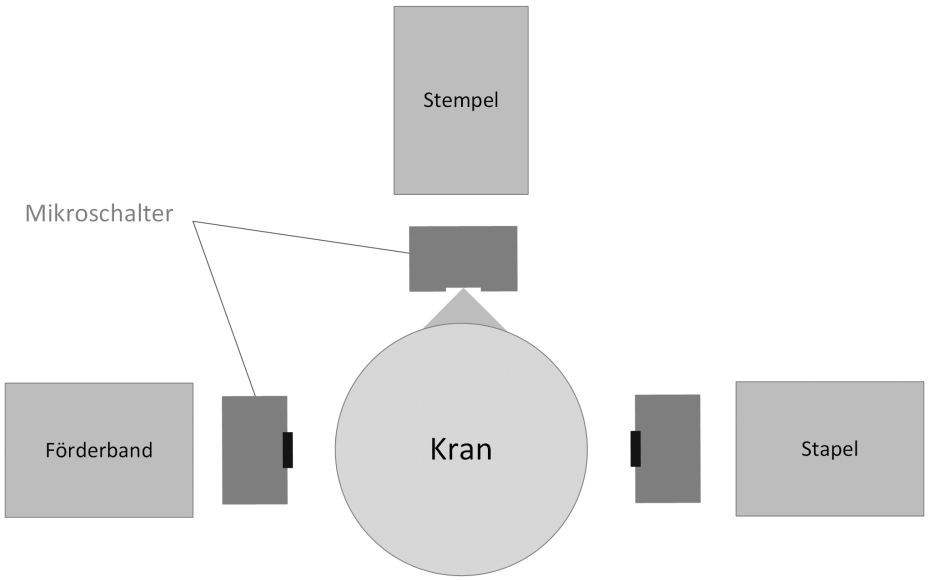

Abbildung 9.4.: Illustration der fiktiven Anlage für Szenario 1

Abbildung 9.5 zeigt das Modell der Anlage. Es wurde im xPPU-Baum-Editor modelliert, der auf dem Metamodell der xPPU aus Abschnitt 5.1 basiert. Wie bereits beschrieben, sind alle Elemente enthalten, die in 9.3.1.1 aufgezählt wurden. Zusätzlich zu dem Kran „Production Crane“, der für die Änderungen vorgesehen ist, haben wir noch einen weiteren

Kran „Replacement Crane“ hinzugefügt. Damit soll gezeigt werden, dass sich die Änderung nur auf diesen einen Kran beschränkt und nicht jeden im Modell vorhandenen zur Liste hinzufügt.

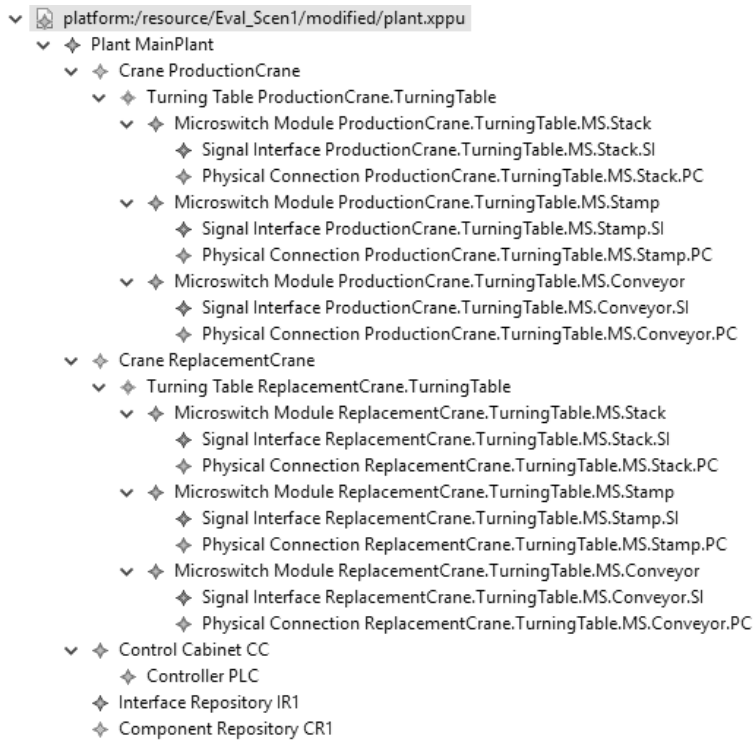

Abbildung 9.5.: Modell einer fiktiven Anlage für Szenario 1 vor der Änderung

Neben dem Modell der Anlage mit den drei Schaltern, die noch entfernt werden müssen, existiert ein zweites Modell, das den Zustand nach dem Entfernen der Schalter sowie dem Hinzufügen des Potentiometers darstellt. Im KAMP4aPS-Framework werden die beiden Modelle verglichen und entfernte, sowie hinzugefügte Elemente erkannt und entsprechend zur Aktivitätsliste hinzugefügt. Abbildung 9.6 zeigt das Modell der Anlage nach den Änderungen. Beim Drehtisch des Krans „Production Crane“ wurden die drei Schalter „Micro Switch“ für den Stapel, den Stempel sowie das Förderband entfernt. Teil der Schalter ist jeweils eine Signalschnittstelle sowie die Befestigung. Diese wurden mit den Schaltern ebenfalls entfernt.

Dem Drehtisch wurde ein Potentiometer hinzugefügt, das wie jedes andere elektrische Bauteil über eine Signalschnittstelle sowie eine Befestigung verfügt. Der zweite Kran „ReplacementCrane“ bleibt unangetastet. Um eine korrekte Berechnung der Ausbreitung zu gewährleisten, muss das Interface Repository angepasst werden. Hinzugefügt wurde die Signalschnittstelle sowie die Befestigung. Das Entfernen der Schnittstellen, wenn diese vorhanden sind im Repository, geschieht automatisch und bedarf keines Eingreifens. Für dieses Szenario ist es zwar nicht notwendig auch das Component Repository anzupassen, da die Überprüfung durch einen Vergleich geschieht. Dennoch ist es für ein konsistentes Modell erforderlich.

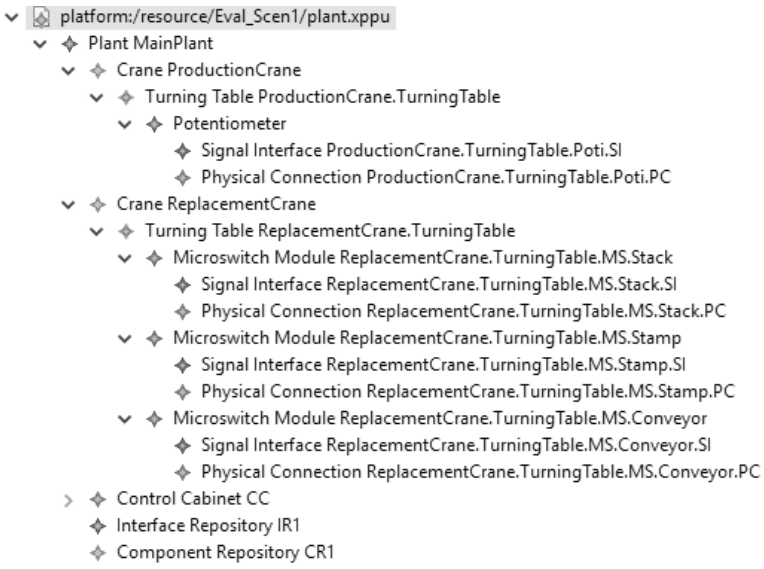

Abbildung 9.6.: Modell der Anlage aus Szenario 1 nach den Änderungen

9.3.1.2. Annotationsmodell

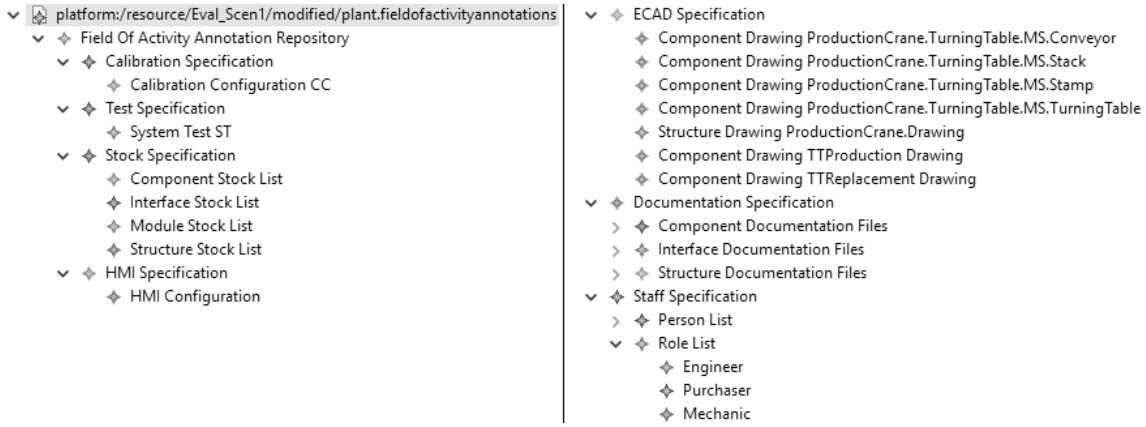

Abbildung 9.7.: Modell der Annotationen für Szenario 1

Um eine adequate Einbindung der Artefakte zu testen, die nicht direkt Teil der Anlage sind, haben wir alle sieben Klassen der Annotationen direkt in das Modell eingebunden. Zu sehen ist das Modell in der Abbildung 9.7. Das Szenario sieht drei Rollen vor: der Ingenieur, der für die Dokumentation, die ECAD Zeichnungen sowie das HMI zuständig ist, der Einkäufer, der für das Lager zuständig ist, und der Mechaniker, der für das Testen, Modifizieren und Kalibrieren verantwortlich ist.

Im Rahmen der entworfenen Anlage ist für jede Komponente und Struktur eine ECAD Zeichnung vorgesehen. Module sind in der Anlage nicht verbaut, daher werden sie in diesem Szenario nicht betrachtet. Da jedes Element mögliche Auswirkungen auf das HMI

haben könnte, werden alle dort hinzugefügt. Außerdem muss jede Komponente in der Lagerhaltung erfasst sowie Teile, die kalibriert werden, zur Kalibrierungsspezifikation hinzugefügt werden. Im vorliegenden Fall wird davon ausgegangen, dass jedes Element bei einer Änderung einer Kalibrierung bedarf.

9.3.1.3. Programmmodell

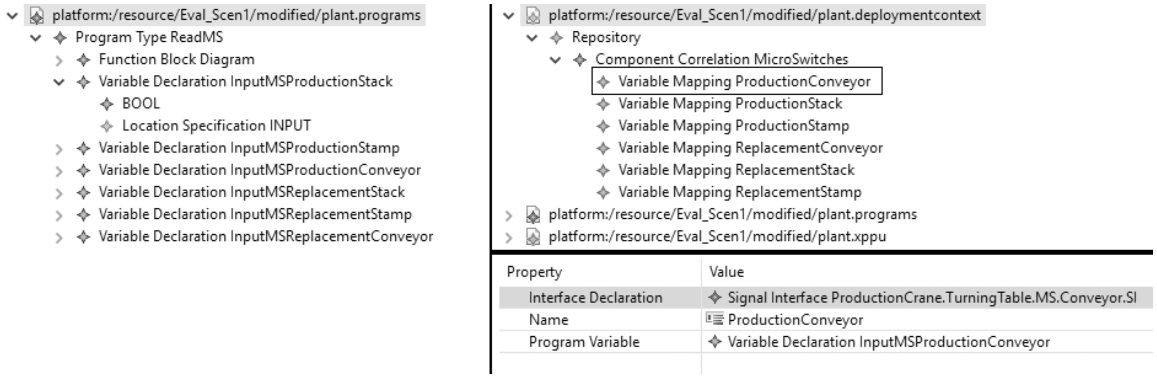

Abbildung 9.8.: Modell der Programme und Variablen Abbildung für Szenario 1

Im Programmmodell werden die Programme modelliert, die auf den Komponenten der Anlage zum Einsatz kommen. Es ist aber auch möglich ein generelles Modell mit allen Programmen anzulegen. Da es keine direkte Relation zu den Komponenten gibt, auf denen die Software eingesetzt wird, können z. B. verschiedene Varianten innerhalb des Modells existieren, die je nach Bedarf in Relation zu einer Komponente gesetzt werden können. Außerdem ist es möglich, das selbe Programm auf mehreren Komponenten auszuliefern. In der Version von KAMP4aPS, die zur Evaluation eingesetzt wird, werden nur die globalen Variablen eines Programms betrachtet. Daher genügt es, ein Programm mit den Variablen zu modellieren.

Abbildung 9.8 zeigt das Programm „ReadMS“ das über sechs Eingabevariablen verfügt. Jede dieser Variablen entspricht einem Schalter innerhalb der Anlage. Alle Variablendeklarationen sind identisch, daher wird exemplarisch nur die Variable „InputMSProductionStack“ vorgestellt.

9.3.1.4. Auslieferungsmodell

Im Auslieferungsmodell wird die Relation zwischen der Software und den Komponenten hergestellt. Die Relation zwischen den beiden Modellen ist intrinsisch nicht vorhanden und muss daher extrinsisch definiert werden. Dies geschieht im Auslieferungsmodell. Wie auf der rechten Seite von Abbildung 9.8 können Korrelationen zusammengefasst werden. Im vorliegenden Fall werden alle Abbildungen auf die Schalter in der Korrelation „Micro Switches“ zusammengefasst.

Innerhalb einer Korrelation werden Abbildungen von einer Schnittstelle einer Komponente auf eine Variable definiert. Ob es sich nun um eine Eingabe- oder Ausgabevariable

handelt, ist dabei nicht relevant. Die Korrektheit der Abbildung liegt in der Verantwortung des Domänenexperten, der das Modell erstellt. Im unteren Teil der Grafik 9.8 ist exemplarisch die Abbildung einer Signalschnittstelle auf eine Variable gezeigt.

9.3.1.5. Modifikationsmodell

Der Vollständigkeit halber sei das Modifikationsmodell hier noch erwähnt. Es muss erstellt werden, um die Analyse zu starten, enthält aber für das erste Szenario keine Modifikationen. Das wäre notwendig, wenn eine Komponente erhalten bleibt, aber modifiziert werden muss. Das Hinzufügen und Entfernen ist nicht direkt im Modell abgebildet, wird aber implizit durch die Analyse hergeleitet. Die Darstellung eines Modifikationsmodells ist in Abschnitt 9.3.2.3 zu finden.

9.3.2. Testanlage - Buswechsel

In diesem Abschnitt werden die Modelle vorgestellt, die wir für die Evaluation des zweiten Szenarios erstellt haben. Im Vergleich zum ersten Szenario sind die nun gezeigten Modelle umfangreicher. Das hat zum einen den Grund, dass mehr Komponenten involviert sind, und zum anderen ist es notwendig, ein ausreichend komplexes Modell zu erstellen. Damit wird sichergestellt, dass die Propagation der Änderungen korrekt abläuft. Zunächst werden das Modell der Anlage (9.3.2.1) und das Annotationsmodell (9.3.2.2), welche einen deutlich größeren Umfang aufweisen als die vorherigen Modelle vorgestellt. Anschließend werden das Modifikationsmodell (9.3.2.3) sowie das Programm- und Auslieferungsmodell (9.3.2.4) ebenfalls im nachfolgenden Abschnitt behandelt.

9.3.2.1. Anlagenmodell

Die Anlage ist so modelliert, dass zwei Bussysteme enthalten sind. Diese sind voneinander entkoppelt, damit später festgestellt wird, ob die Änderungsausbreitung nur auf einen Bus beschränkt ist. Die beiden Bussysteme verwenden den Profibus DP. Für die Änderungsausbreitung ist es unerheblich, welches Bussystem für die Modellierung genommen wird. Die Eigenheiten des eingesetzten Bussystems, sei es nun ProfibusDP[74], EtherCAT[8] oder ein beliebig anderes, sind vom Anwender zu beachten und entsprechend zu modellieren. Dazu zählt z.B. die mögliche Topologie des Busnetzes.

Die modellierte Anlage ist bestückt mit zwei Kränen, die jeweils zwei Förderbänder bedienen. Ein weiteres Förderband dient der Zulieferung und ist nicht hinter einen Kran geschalten. Diese beiden Kräne und das Zulieferungsförderband bilden mit der ersten SPS den ersten Bus. Im zweiten Bus sind sowohl die zweite SPS sowie ein Kran mit zwei Förderbändern enthalten. Jeder Bus verfügt zusätzlich über jeweils eine Bus Box und die entsprechenden Kabel, um die Komponenten miteinander zu verbinden. Abbildung 9.9 zeigt den Aufbau der eben beschriebenen Anlage.

9.3.2.2. Annotationsmodell

Der Grundaufbau der Annotationen ist wie in Szenario 1 gestaltet. Zur möglichen Kalibrierung sind alle Elemente der Anlage eingetragen. Ebenso ist das zu testende System

Abbildung 9.9.: Modell der Anlage für Szenario 2

die modellierte Anlage. Zur Mitarbeiter-Spezifikation gehören ebenfalls alle möglichen Komponenten, Strukturen und Schnittstellen. Module sind im Modell keine vorhanden, daher werden diese ausgespart.

ECAD Zeichnungen sind angelegt für alle Bus Master, Bus Slaves und Bus Boxen. Für jede der genannten Komponenten jeweils eine Zeichnung. Ebenso wurden Dokumentationen für diese angelegt. Damit das Modell nicht unnötig groß wird, haben wir die Dokumentation auf die Wartungsdokumente beschränkt. Dabei entsprechen die Rollen und Mitarbeiterlisten denen aus Szenario 1. Die Annotationen können je nach Projekt gewählt werden. Es ist notwendig, dass der Nutzer das Modell der Annotationen hinreichend zum realen Anwendungsfall modelliert.

Unnötige Elemente, die, selbst wenn sie als Änderung markiert wurden, nicht angepasst werden bzw. nicht angepasst werden müssen, können ausgespart werden. Abbildung 9.10 zeigt den Aufbau des Anreicherungsmetamodells.

Abbildung 9.10.: Modell der Annotationen für Szenario 2

9.3.2.3. Änderungsanfragen-Modell

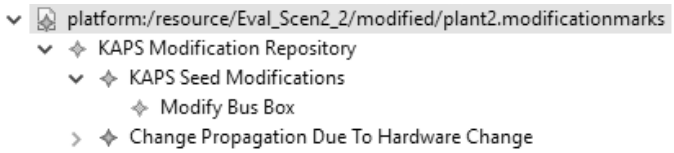

Abbildung 9.11.: Modell der Änderungsanfragen für Szenario 2

Im ersten Szenario mussten keine Modifikationen im Modifikationsmodell definiert werden, da Elemente entfernt bzw. hinzugefügt wurden. Im vorliegenden Fall gestaltet sich die Sachlage nun anders. Es werden weder Elemente hinzugefügt noch entfernt. Es wird initial ein Buselement zur Modifikation markiert. Um das Szenario von anderen möglichen Modifikationen abzugrenzen, muss der Buswechsel als initiale Änderung die Modifikation der Bus Box Komponente aufweisen. Somit wird für Szenario 2 die Bus Box Komponente für den ersten Bus als initiale Änderung markiert. Siehe dazu Abbildung

9.11. Im Modellelement „Modify Bus Box“ ist die Bus Box, die geändert werden soll, als Eigenschaft zu hinterlegen.

9.3.2.4. Programm- und Auslieferungsmodell

Ein Modell jeweils für die Software sowie die Variablen Abbildung wurde nicht erstellt. Im besprochenen Szenario wird die Auswirkung auf die Software nicht betrachtet. Aus diesem Grund wäre es nicht möglich, das Ergebnis dahingehend zu evaluieren. Die Grundfunktionalität bezüglich des Programm- und Auslieferungsmodells wurde zum ersten Szenario bereits evaluiert.

9.4. Ergebnisse der Änderungspropagation

Die Methode für die Auswertung der erzeugten Aufgabenliste wurde bereits im Kapitel der Einführung beschrieben. Zur besseren Übersicht sind die Szenarien in Abschnitte unterteilt. Zuerst werden alle True Positives, False Negatives und False Positives einer Aufgabenliste erfasst. Die kompletten Listen sind zum Vergleich im Anhang unter A.2 und A.3 zu finden. Die Einfärbung geschah manuell und ist nicht Teil des Frameworks. Grün bedeutet, dass die Aufgabe sowohl in der händisch erstellten als auch der automatisch erstellten Liste vorkommen (True Positive). Rot in Abbildung A.2 bedeutet, dass die Einträge in der generierten Liste fehlen (False Negative). Dementsprechend sind die roten Einträge in A.3 die zuviel oder falsch erzeugten Aufgaben (False Positive).

9.4.1. Auswertung des Schaltertausch-Szenarios

Die Ergebnis-Tabelle 9.3 zeigt, dass von den insgesamt automatisch erzeugten 35 Aufgaben (True Positive + False Positive) 31 korrekt erstellt wurden. Die vier Ausreißer sind damit begründet, dass die Signalschnittstellen quasi über zwei Enden verfügen, die beide angepasst werden müssten. Im vorliegenden Fall ist das aber nicht notwendig, daher genügt jeweils eine Aufgabe pro Schnittstelle. Die vier False Positves sind also das Resultat der Überschätzung (FPIS), siehe dazu Abschnitt 2.3.2.1. Ausgelassen wurden keine Aufgaben, daher ist der False Negative Wert gleich null. Das wiederum ist ebenfalls auf das Überschätzen der möglichen Aufgaben zurückzuführen.

Typ	**Menge**
True Positive	31
False Negative	0
False Positive	4

Tabelle 9.3.: Ergebnis von Szenario 1

Im Resultat ist die Menge aller zu findenden Aufgaben in der erzeugten Aufgabenliste enthalten. Der Recall liegt somit bei 1,00 und ist für dieses Szenario ideal. Bei der Präzision weist KAMP4aPS ebenfalls einen guten Wert auf, erreicht aber aufgrund der Problematik

der Überapproximation nicht das Ideal. Im Mittel erreicht die Umsetzung des ersten Szenarios einen Wert von 0,94. Im folgenden Abschnitt betrachten wir das Ergebnis für das zweite Szenario und überprüfen, ob ähnliche Werte auch bei einem komplexeren Aufbau erreicht werden können.

Recall	**1,00**
Precision	**0,89**
F_1	**0,94**

Tabelle 9.4.: Recall und Precision von Szenario 1

9.4.2. Auswertung des Buswechsel-Szenarios

Wie in Tabelle 9.5 zu sehen, wird das Ergebnis aus der Auswertung für das zweite Szenario gezeigt. In der Liste der automatisch erzeugten Aufgaben sind 54 Einträge enthalten. Diese bestehen aus der Summe der True Positives und False Positives. Alle 35 Aufgaben wurden korrekt erstellt. Im Gegensatz zum ersten Szenario sind keine Ausreißer in der Aufgabenliste enthalten. Weder False Positives noch False Negatives wurden generiert. Somit ist erwiesen, dass der zweite Bus, repräsentiert durch die zweite Bus Box und die daran angeschlossenen Komponenten, vom Algorithmus nicht als Aufgaben in Betracht gezogen werden. Auch dass die gestellte Aufgabe, das Finden aller Aufgaben, gelöst wird, ist somit erwiesen.

Typ	**Menge**
True Positive	54
False Negative	0
False Positive	0

Tabelle 9.5.: Ergebnis von Szenario 2

Aufgrund des positiven Ergebnisses weißt das Resultat einen Recall-Wert von 1,00 auf. Auch der Precission-Wert liegt bei 1,00. Daraus ergibt sich für das Mittel ebenfalls ein Wert von 1,00. Somit wurde gezeigt, dass zumindest für dieses Szenario mit diesem Aufbau und den gewählten Annotationen das Framework alle Aufgaben generiert, die bei der Durchführung der Änderungen von Szenario 2 notwendig sind.

Recall	**1,00**
Precision	**1,00**
F_1	**1,00**

Tabelle 9.6.: Recall und Precision von Szenario 2

9.5. Einschränkungen durch Vorannahmen

Die Evaluationsszenarien wurden unter einer bestimmten Prämisse entworfen und umgesetzt. In diesem Abschnitt wird erörtert, wie die Ergebnisse der Evaluation unter eben dieser Prämisse zu interpretieren sind. Wie bereits beschrieben, basiert das Modell der Hardware, also der mechanischen und elektronischen Komponenten, auf die xPPU der TUM. Somit sind die Ergebnisse nur für eine Instanz eines aPS gültig. Aber für die xPPU sind die Ergebniswerte als positiv zu bewerten.

Zwar wurde gezeigt, dass die Änderungsausbreitungsanalyse mittels dem erweiterten KAMP-Framework für aPS angewendet werden kann. Aber die Menge der untersuchten Szenarien ist noch nicht ausreichend, um eine allgemeingültige Aussage zur Qualität der Ergebnisse treffen zu können. Außerdem ist das Ergebnis auch von der Qualität der Modelle, sowie der Änderungsausbreitungsregeln und deren Umsetzung abhängig. So können die Ergebnisse für ein vermeintlich komplexeres Projekt negativ ausfallen, was aber der Komplexität geschuldet sein kann, und nicht unbedingt auf das Framework zurückzuführen ist. Eine Tendenz lässt sich durch die bisher geringe Anzahl an Szenarien noch nicht erkennen, dazu ist die Umsetzung weiterer Szenarien notwendig.

Wie auch CoCoME dient die xPPU als Plattform, um verschiedene Ansätze zu testen und diese untereinander zu vergleichen. Aus diesem Grund ist es notwendig, ein Modell der xPPU auf der Basis eines allgemeingültigen Metamodells der Hardware zu erstellen. Somit können alle Szenarien künftig auf diesem einen Grundsystem getestet werden. Verschiedene Implementierungen der Änderungsausbreitungsregeln lassen sich somit vergleichen. Der Vergleich verschiedener Verfahren wäre bereits jetzt möglich, dazu müsste ein vollständiges Modell der xPPU erstellt werden. Auf dieser Basis könnten allgemeine Änderungsausbreitungsregeln mit den bereits bestehenden verglichen werden.

10. Ausblick

Im Rahmen dieser Arbeit wurde auf der Basis zweier Szenarien aufgezeigt, dass es prinzipiell möglich ist, automatisierte Produktionssysteme in das KAMP-Framework zu integrieren. Also dass mithilfe des Frameworks automatisiert Änderungsvorhersagen in einem aPS getroffen werden können. In der Evaluation haben wir gezeigt, dass mit entsprechendem Aufwand und speziellem Domänenwissen die Änderungsausbreitung in den Szenarien automatisiert berechnet werden kann. Die Abweichungen im ersten Szenario ist eine Ursache der Überapproximation. Diese ist ein generelles Problem des Frameworks, das u. U. mit feingranulareren Regeln, zumindest für das erste Szenario, in den Griff zu bekommen sein sollte.

Zum aktuellen Zeitpunkt wird im ersten Szenario von einer allgemeinen Änderung der Komponente Mikroschalter ausgegangen. Eine Differenzierung zwischen den Änderungsarten könnte die überflüssigen Änderungen filtern. Die Differenzierung erhöht den Implementierungsaufwand der jeweiligen Regeln, da Sonderfälle und Wechselwirkungen stärker in den Vordergrund rücken. Abhängig davon, bei wie vielen der noch möglichen Szenarien dieser Aufwand betrieben werden muss, stellt sich die Frage, ob das Ergebnis der Überapproximation vielleicht doch ausreichend sein kann.

Vor allem ist zu beachten, dass je allgemeingültiger die Regeln umgesetzt sind, ein breiteres Feld an Automatisierungssystemen direkt abbildbar und somit verwendbar wären. Außerdem bilden die implementierten Szenarien nur eine Teilmenge aller von der TUM definierten Szenarien ab. Die dazu nötigen Regeln sind in KAMP4aPS stark an die xPPU gekoppelt und nicht unbedingt allgemeingültig auf alle möglichen Automatisierungssysteme bzw. automatisierte Produktionssysteme anwendbar.

10.1. Generalisierung der KAMP4aPS Implementierung

Eine weiterführende Forschungsfrage könnte sein, ob es ein hinreichendes Metamodell gibt, mit dessen Hilfe sich alle automatisierten Produktionssysteme oder gar alle Automatisierungssysteme modellieren lassen. Zum einen bestünde der Vorteil für den Anwender darin, dass dieser kein Metamodell für die zu modellierende Anlage erstellen muss. Dadurch wäre die Einstiegshürde geringer, da die Anwender bezüglich der Metamodellierung keine Kenntnisse benötigen, um KAMP4aPS verwenden zu können. Zum anderen kann das Unternehmen somit Zeit sparen, die dann in die Modellierung der zu analysierenden Anlage eingesetzt werden könnte.

Auf Basis der Szenarien und den daraus resultierenden Regeln besteht u. U. die Möglichkeit, das Regelset soweit herunterzubrechen bzw. auf einen gemeinsamen Nenner zu bringen, so dass dafür allgemeingültige Regeln erstellt werden können. Dazu wird es aber notwendig sein, weitere Szenarien zu implementieren, um diese auf der Regelebene

analysieren zu können. Auch ist für die Zukunft interessant, wie viele der möglichen Änderungen mithilfe der Szenarien umgesetzt werden. Diese Erkenntnis korreliert mit der Menge der testbaren Änderungen für das Framework.

10.2. Domänenspezifische Sprache zum Erstellen der Ausbreitungsregeln

Eine Idee, die während der Bearbeitung dieser Arbeit aufkam, war, die Regeln, die im Moment noch in Java implementiert werden müssen, mittels einer domänenspezifischen Sprache (DSL) zu realisieren. Im Idealfall entspricht die Syntax der DSL einer einfachen Abfolge von Anlagenelementen, die z. B. mit einer Pfeilnotation verbunden werden. Die Ausbreitungsregel für das zweite Szenario könnte aussehen wie in Listing 10.3. Diese

```
BusBox <-> SignalInterface <-> BusCable <-> SignalInterface <-> BusMaster
BusBox <-> SignalInterface <-> BusCable <-> SignalInterface <-> BusSlave
BusBox <-> SignalInterface <-> BusCable <-> SignalInterface <-> BusBox
BusMaster <-> SignalInterface <-> BusCable <-> SignalInterface <-> BusSlave
BusSlave <-> SignalInterface <-> BusCable <-> SignalInterface <-> BusSlave
```

Listing 10.3: DSL - Ausbreitungsregeln

Pfeile suggerieren die Änderungsausbreitung in die jeweilige Richtung, in die diese zeigen. Damit wäre es möglich, dass der Domänenexperte ohne großen Aufwand und vor allem ohne den Technologieexperten in der Lage wäre, die Regeln selbst zu implementieren. Natürlich müsste dieser noch bezüglich der Implementierung geschult werden. Bei speziellen Regeln, die nicht unbedingt per DSL abgebildet werden können, würde weiterhin ein Technologieexperte benötigt.

Aber im Ergebnis würde durch die Simplifizierung der Anwendung die Akzeptanz in der Zielgruppe gefördert werden. Ein weiterer Vorteil wäre das Entkoppeln der konkreten Regeln von der Codebasis des Frameworks. Diese könnten somit getrennt gewartet werden und es bestünde die Möglichkeit, dass Regeln aus anderen Quellen ohne die Notwendigkeit der aufwändigen Integration verwendet werden könnten.

10.3. Notwendige Schritte bis zur Einsetzbarkeit

Wie bereits dargelegt, bestehen bei der Entwicklung von KAMP4aPS noch einige Hürden, die es zu überwinden gilt, bis KAMP4aPS produktiv eingesetzt werden kann. Dazu zählt die Umsetzung aller Szenarien und die Beantwortung der Frage, wie viele mögliche Änderungen dadurch abgebildet werden können. Die Entkopplung von der xPPU ist ebenfalls noch zu realisieren, bevor es auch für andere Automatisierungssysteme verwendet werden kann. Dann gilt die Einsetzbarkeit nur für KAMP4aPS. Die anderen bereits realisierten Domänen im KAMP-Framework sind dabei noch nicht beachtet.

Außerdem sind im Moment alle Elemente des KAMP-Frameworks Insellösungen. Wenn die Arbeiten an den Framework Fragmenten abgeschlossen sind, ist es Ziel, diese zusammenzuführen, so dass die Analyse von Software, Geschäftsprozessen und Automatisierungssystemen verknüpft werden. Im Idealfall kann der Anwender aus einer Menge an Domänen wählen, für die dieser in der Folge entsprechenden Modelle erstellen kann. Auf der Basis dieser Modelle werden die nötigen Änderungen berechnet. Im Idealfall wird nicht nur eine Domäne für sich betrachtet, sondern auch die Auswirkungen auf die anderen gewählten Domänen. Natürlich nur soweit diese Verbindungen auch vorhanden sind.

Das Potential für ein revolutionäres Framework zur Optimierung langlebiger Systeme ist gegeben. Bis es marktreif ist und eingesetzt werden wird, müssen noch einige Grundfunktionalitäten erweitert und des System nutzerfreundlicher gestaltet werden.

11. Abkürzungsverzeichnis

AIS Actual Impact Set

aPS Automatisierte Produktionssysteme

AS Ablaufsprache

AWL Anweisungsliste

CHF Schweizer Franken

CIS Candidate Impact Set

CoCoME Common Component Modelling Example

CoCoMo Comprehensive Cost Model

DIS Discovered Impact Set

DSL Domänenspezifische Sprache

ECAD Electronic Computer Aided Design

EMF Eclipse Modeling Framework

EN Europäische Norm

FPIS False Positive Impact Set

FUP Funktionsplan

GoF Gang of Four

HMI Human Machine Interface

HTA Hierarchical Task Analysis

IDE Integrierte Entwicklungsumgebung, engl. Integrated Development Environment

IEC 61131-3 IEC Stardard Nummer 61131-3

IEC International Electrotechnical Commission

KAMP Karlsruhe Architectural Maintainability Prediction

KAMP4aPS KAMP für automatisierte Produktionssysteme

KOP Kontaktplan

LED Light-emitting diode

MDSD Modellgetriebene Softwareentwicklung, engl. Model-Driven Software Development

OMG Object Management Group

POU Program Operation Unit

PPU Pick and Place Unit

SIS Straight Impact Set

SPS Speicherprogrammierbare Steuerung

ST Strukturierter Text

SVN Subversion

TUM Technische Universität München

UML Unified Modeling Language

xPPU Extended Pick and Place Unit

Literatur

[1] *Ablaufsprache*. In: *Wikipedia*. Page Version ID: 160714297. 16. Dez. 2016. URL: https://de.wikipedia.org/w/index.php?title=Ablaufsprache&oldid=160714297 (besucht am 01. 02. 2017).

[2] Nadine Amsel u. a. „Toward Sustainable Software Engineering (NIER Track)“. In: *Proceedings of the 33rd International Conference on Software Engineering*. ICSE '11. Waikiki, Honolulu, HI, USA: ACM, 2011, S. 976–979. ISBN: 978-1-4503-0445-0. DOI: 10.1145/1985793.1985964. URL: http://doi.acm.org/10.1145/1985793.1985964.

[3] Ermeson C. Andrade u. a. „Modeling and analyzing server system with rejuvenation through SysML and stochastic reward nets“. In: *Availability, Reliability and Security (ARES), 2011 Sixth International Conference on*. IEEE, 2011, S. 161–168. URL: http://ieeexplore.ieee.org/abstract/document/6045928/ (besucht am 31. 03. 2017).

[4] John Annett. „Hierarchical task analysis“. In: *Handbook of cognitive task design* 2 (2003), S. 17–35. URL: https://books.google.de/books?hl=de&lr=&id=dElPH0ruRsC&oi=fnd&pg=PA17&dq=Hierarchical+Task+Analysis&ots=EGnTn-2KAE&sig=UTXazOebA8SPHzLxo6gqwczzRFc (besucht am 04. 02. 2017).

[5] John Annett. „TASK ANALYSIS AND TRAINING DESIGN.“ In: (1967). URL: https://eric.ed.gov/?id=ED019566 (besucht am 04. 02. 2017).

[6] *Anweisungsliste*. In: *Wikipedia*. Page Version ID: 147998150. 13. Nov. 2015. URL: https://de.wikipedia.org/w/index.php?title=Anweisungsliste&oldid=147998150 (besucht am 01. 02. 2017).

[7] *AutomationML*. In: *Wikipedia*. Page Version ID: 152367425. 10. März 2016. URL: https://de.wikipedia.org/w/index.php?title=AutomationML&oldid=152367425 (besucht am 03. 10. 2016).

[8] Beckhoff. *EtherCAT System Documentation*. 18. Aug. 2016. URL: https://www.google.de/url?sa=t&rct=j&q=&esrc=s&source=web&cd=4&cad=rja&uact=8&ved=0ahUKEwiG1pSnv_vSAhWGxxQKHe3pBdcQFggzMAM&url=https://download.beckhoff.com/download/document/io/ethercat-terminals/ethercatsystem_en.pdf&usg=AFQjCNHL546sKXXWtRGIKYtc99mHOBVhjw&sig2=Im1Tmb_-Bmec1R9DW6zMPQ (besucht am 29. 03. 2017).

[9] P. Bengtsson und Jan Bosch. „Architecture level prediction of software maintenance“. In: *Software Maintenance and Reengineering, 1999. Proceedings of the Third European Conference on*. IEEE, 1999, S. 139–147. URL: http://ieeexplore.ieee.org/abstract/document/756691/ (besucht am 29. 03. 2017).

[10] PerOlof Bengtsson u. a. „Architecture-level modifiability analysis (ALMA)“. In: *Journal of Systems and Software* 69.1 (2004), S. 129–147. URL: http://www.sciencedirect.com/science/article/pii/S0164121203000803 (besucht am 29. 03. 2017).

[11] Barry Boehm u. a. „COCOMO suite methodology and evolution“. In: *CrossTalk* 18.4 (2005), S. 20–25. URL: https://pdfs.semanticscholar.org/745b/123fdffc944f6e9%2024f4aec446dd86fed1e01.pdf (besucht am 04. 02. 2017).

[12] Rainer Böhme und Ralf Reussner. „Validation of predictions with measurements“. In: *Dependability metrics*. Springer, 2008, S. 14–18.

[13] Shawn A. Bohner. „Extending software change impact analysis into cots components“. In: *Software Engineering Workshop, 2002. Proceedings. 27th Annual NASA Goddard/IEEE*. IEEE, 2002, S. 175–182.

[14] Shawn A. Bohner. „Impact analysis in the software change process: a year 2000 perspective“. In: *, International Conference on Software Maintenance 1996, Proceedings.* , International Conference on Software Maintenance 1996, Proceedings. Nov. 1996, S. 42–51. DOI: 10.1109/ICSM.1996.564987.

[15] Ralf Carbon u. a. *Architecture-Centric Software Producibility Analysis*. Fraunhofer IRB Verlag, 2012. URL: http://dl.acm.org/citation.cfm?id=2341150 (besucht am 04. 02. 2017).

[16] Vittorio Castelli u. a. „Proactive management of software aging“. In: *IBM Journal of Research and Development* 45.2 (2001), S. 311–332. URL: http://ieeexplore.ieee.org/abstract/document/5389090/ (besucht am 31. 03. 2017).

[17] Scott Chacon und Ben Straub. *Pro git*. Apress, 2014. URL: https://books.google.de/books?hl=de&lr=&id=jVYnCgAAQBAJ&oi=fnd&pg=PP3&dq=pro+git&ots=vN63o1eu96&sig=U3gW_aVTLVtV5yCQxWsABO8sgjE (besucht am 03. 04. 2017).

[18] P. Clements und L. Northrop. „Software Product Lines: Practices and Patterns, SEI Series in Software Engineering, Addison W esley“. In: *Reading, MA, USA* (2001).

[19] Paul Clements, Rick Kazman und Mark Klein. *Evaluating Software Architectures*. Pearson Education, 2002. ISBN: 978-81-317-1592-5. URL: https://books.google.de/books?id=Isne75m_ULgC.

[20] Ben Collins-Sussman, Brian W. Fitzpatrick und C. Michael Pilato. *Version Control with Subversion*. URL: http://svnbook.red-bean.com/en/1.7/svn-book.html (besucht am 03. 04. 2017).

[21] Melvin E. Conway. „How do committees invent“. In: *Datamation* 14.4 (1968), S. 28–31. URL: http://michaelsaunders.com.au/wp-content/uploads/2016/10/Conway-Man.pdf (besucht am 08. 04. 2017).

[22] Ferruccio Damiani, Ina Schaefer und Tim Winkelmann. „Delta-oriented multi software product lines“. In: *Proceedings of the 18th International Software Product Line Conference-Volume 1*. ACM, 2014, S. 232–236. URL: http://dl.acm.org/citation.cfm?id=2648536 (besucht am 09. 04. 2017).

[23] ZVEI Automation Division. *Life-Cycle-Management für Produkte und Systeme der Automation*. 2012. URL: http://www.zvei.org/Publikationen/Leitfaden_LifeCycle.pdf (besucht am 17. 09. 2016).

[24] R. Drath. *Datenaustausch in der Anlagenplanung mit AutomationML: Integration von CAEX, PLCopen XML und COLLADA*. VDI-Buch. Springer Berlin Heidelberg, 2009. ISBN: 9783642046735.

[25] Rainer Drath und Alexander Horch. „Industrie 4.0: Hit or hype?[industry forum]". In: *IEEE industrial electronics magazine* 8.2 (2014), S. 56–58. (Besucht am 01. 10. 2016).

[26] *EN 61131*. Page Version ID: 161909022. 23. Jan. 2017. URL: https://de.wikipedia.org/wiki/EN_61131 (besucht am 31. 01. 2017).

[27] *Funktionsbausteinsprache*. In: *Wikipedia*. Page Version ID: 146021840. 14. Sep. 2015. URL: https://de.wikipedia.org/w/index.php?title=Funktionsbausteinsprache&oldid=146021840 (besucht am 01. 02. 2017).

[28] E. Gamma u. a. *Design Patterns: Elements of Reusable Object-Oriented Software (Adobe Reader)*. Pearson Education, 1994. ISBN: 978-0-321-70069-8. URL: https://books.google.de/books?id=6oHuKQe3TjQC.

[29] D. Garlan u. a. „Evolution styles: Foundations and tool support for software architecture evolution". In: *2009 Joint Working IEEE/IFIP Conference on Software Architecture European Conference on Software Architecture*. 2009 Joint Working IEEE/IFIP Conference on Software Architecture European Conference on Software Architecture. Sep. 2009, S. 131–140. DOI: 10.1109/WICSA.2009.5290799.

[30] golem.de. *Wenn der Kühlschrank mitdenkt*. 2016. URL: http://www.golem.de/news/liebherr-wenn-der-kuehlschrank-mitdenkt-1609-123066.html (besucht am 07. 09. 2016).

[31] Tassilo Horn und Jürgen Ebert. *Ein Referenzschema für die Sprachen der IEC 61131*. URL: https://www.uni-koblenz.de/~fb4reports/2008/2008_13_Arbeitsberichte.pdf (besucht am 04. 11. 2017).

[32] Y. Huang u. a. „Software rejuvenation: analysis, module and applications". In: *Twenty-Fifth International Symposium on Fault-Tolerant Computing. Digest of Papers*. Twenty-Fifth International Symposium on Fault-Tolerant Computing. Digest of Papers. Juni 1995, S. 381–390. DOI: 10.1109/FTCS.1995.466961.

[33] Eclipse Foundation Inc. *Eclipse - The Eclipse Foundation open source community website*. URL: http://www.eclipse.org/ (besucht am 28. 03. 2017).

[34] Karl-Heinz John und Michael Tiegelkamp. *IEC 61131-3: programming industrial automation systems: concepts and programming languages, requirements for programming systems, decision-making aids*. Springer Science & Business Media, 2010.

[35] Frédéric Jouault u. a. „ATL: a QVT-like transformation language". In: *Companion to the 21st ACM SIGPLAN symposium on Object-oriented programming systems, languages, and applications*. ACM, 2006, S. 719–720. URL: http://dl.acm.org/citation.cfm?id=1176691 (besucht am 28. 03. 2017).

[36] Henning Kagermann, Wolf-Dieter Lukas und Wolfgang Wahlster. „Industrie 4.0: Mit dem Internet der Dinge auf dem Weg zur 4. industriellen Revolution“. In: *VDI nachrichten* 13 (2011), S. 2011. URL: https://pdfs.semanticscholar.org/5fbc/8571686c74516b4da38f6780c30fb31da2f0.pdf (besucht am 01. 10. 2016).

[37] *Kontaktplan*. In: *Wikipedia*. Page Version ID: 161353742. 6. Jan. 2017. URL: https://de.wikipedia.org/w/index.php?title=Kontaktplan&oldid=161353742 (besucht am 01. 02. 2017).

[38] Heiko Koziolek. „Sustainability Evaluation of Software Architectures: A Systematic Review“. In: *Proceedings of the Joint ACM SIGSOFT Conference – QoSA and ACM SIGSOFT Symposium – ISARCS on Quality of Software Architectures – QoSA and Architecting Critical Systems – ISARCS*. QoSA-ISARCS '11. New York, NY, USA: ACM, 2011, S. 3–12. ISBN: 978-1-4503-0724-6. DOI: 10.1145/2000259.2000263. URL: http://doi.acm.org/10.1145/2000259.2000263 (besucht am 04. 02. 2017).

[39] Irwin Kwan, Marcelo Cataldo und Daniela Damian. „Conway's law revisited: The evidence for a task-based perspective“. In: *IEEE software* 29.1 (2012), S. 90–93. URL: http://ieeexplore.ieee.org/abstract/document/6111370/ (besucht am 08. 04. 2017).

[40] C. Larman. *Applying UML and Patterns: An Introduction to Object-oriented Analysis and Design and the Unified Process*. Prentice Hall PTR, 2002. ISBN: 9780130925695.

[41] C. Legat, J. Folmer und B. Vogel-Heuser. „Evolution in industrial plant automation: A case study“. In: *IECON 2013 - 39th Annual Conference of the IEEE Industrial Electronics Society*. IECON 2013 - 39th Annual Conference of the IEEE Industrial Electronics Society. Nov. 2013, S. 4386–4391. DOI: 10.1109/IECON.2013.6699841.

[42] Meir M. Lehman u. a. „Metrics and laws of software evolution-the nineties view“. In: *Software Metrics Symposium, 1997. Proceedings., Fourth International*. IEEE, 1997, S. 20–32. URL: http://ieeexplore.ieee.org/abstract/document/637156/ (besucht am 31. 03. 2017).

[43] Fang Li u. a. „Specification of the requirements to support information technology-cycles in the machine and plant manufacturing industry“. In: *IFAC Proceedings Volumes* 45.6 (2012), S. 1077–1082.

[44] Robert C. Martin. *Clean Code-Refactoring, Patterns, Testen und Techniken für sauberen Code: Deutsche Ausgabe*. MITP, 2013. URL: https://books.google.de/books?hl=de&lr=&id=7pVbAgAAQBAJ&oi=fnd&pg=PA1&dq=clean+code&ots=3j7_h6sEzz&sig=_nlRulIvbfE-vZ_YHrWAm2dvPtg (besucht am 31. 03. 2017).

[45] M. Naab. „Improving the Flexibility of SOA-Based Information Systems by Adopting Practices from Product Line Engineering“. In: *Doctoral Symposium of SPLC*. Bd. 3. 2009.

[46] Matthias Naab. „Enhancing Architecture Design Methods for Improved Flexibility in Long-Living Information Systems“. In: *Software Architecture*. European Conference on Software Architecture. DOI: 10.1007/978-3-642-23798-0_19. Springer, Berlin, Heidelberg, 13. Sep. 2011, S. 194–198. URL: http://link.springer.com/chapter/10.1007/978-3-642-23798-0_19 (besucht am 04. 02. 2017).

[47] David Lorge Parnas. „Software Aging“. In: *Proceedings of the 16th International Conference on Software Engineering*. ICSE '94. Sorrento, Italy: IEEE Computer Society Press, 1994, S. 279–287. ISBN: 0-8186-5855-X. URL: http://dl.acm.org/citation.cfm?id=257734.257788.

[48] Daniel J. Paulish und Len Bass. *Architecture-centric software project management: A practical guide*. Addison-Wesley Longman Publishing Co., Inc., 2001. URL: http://dl.acm.org/citation.cfm?id=560012 (besucht am 04. 02. 2017).

[49] H. Prähofer u. a. „Feature-oriented development in industrial automation software ecosystems: Development scenarios and tool support“. In: *2016 IEEE 14th International Conference on Industrial Informatics (INDIN)*. 2016 IEEE 14th International Conference on Industrial Informatics (INDIN). Juli 2016, S. 1218–1223. DOI: 10.1109/INDIN.2016.7819353.

[50] *QVT 1.2*. URL: http://www.omg.org/spec/QVT/1.2/ (besucht am 28. 03. 2017).

[51] Christian Ramsauer. „Industrie 4.0-Die Produktion der Zukunft“. In: *WINGbusiness* 3.2013 (2013), S. 6–12. (Besucht am 01. 10. 2016).

[52] Koichiro Rinsaka und Tadashi Dohi. „A faster estimation algorithm for periodic preventive rejuvenation schedule maximizing system availability“. In: *International Service Availability Symposium*. Springer, 2007, S. 94–109. URL: http://link.springer.com/chapter/10.1007/978-3-540-72736-1_9 (besucht am 31. 03. 2017).

[53] Kiana Rostami. „Domain-spanning Maintainability Analysis for Software-intensive Systems.“ In: *Software Engineering (Workshops)*. 2015, S. 106–108. URL: https://pdfs.semanticscholar.org/a239/7d6974b830bb8a9d4b7da72e48b9ab029aba.pdf (besucht am 04. 02. 2017).

[54] Kiana Rostami u. a. „Architecture-based assessment and planning of change requests“. In: *Proceedings of the 11th International ACM SIGSOFT Conference on Quality of Software Architectures*. ACM, 2015, S. 21–30. URL: http://dl.acm.org/citation.cfm?id=2737198 (besucht am 29. 03. 2017).

[55] Felix Salfner, Maren Lenk und Miroslaw Malek. „A survey of online failure prediction methods“. In: *ACM Computing Surveys (CSUR)* 42.3 (2010), S. 10. URL: http://dl.acm.org/citation.cfm?id=1670680 (besucht am 31. 03. 2017).

[56] M. Schleipen. „A concept for conformance testing of AutomationML models by means of formal proof using OCL“. In: *2010 IEEE Conference on Emerging Technologies and Factory Automation (ETFA)*. 2010 IEEE Conference on Emerging Technologies and Factory Automation (ETFA). Sep. 2010, S. 1–5. DOI: 10.1109/ETFA.2010.5641270.

[57] Ulrich Sendler. „Industrie 4.0“. In: *Berlin/Heidelberg* (2013). URL: http://link.springer.com/content/pdf/10.1007/978-3-642-36917-9.pdf (besucht am 01. 10. 2016).

[58] Hisham El-Shishiny, Sally Deraz und Omar Bahy. „Mining software aging patterns by artificial neural networks“. In: *IAPR Workshop on Artificial Neural Networks in Pattern Recognition*. Springer, 2008, S. 252–262. URL: http://link.springer.com/chapter/10.1007/978-3-540-69939-2_24 (besucht am 31. 03. 2017).

[59] Siemens. *SEQUENTIAL FUNCTION CHART*. Industry Automation and Drive Technologies - SCE. Dez. 2010. URL: https://w3.siemens.com/mcms/sce/de/fortbildungen/ausbildungsunterlagen/hochschul-module/tabcardseiten/Documents/V7.0/eP01-07_sequential_function_chart_RC1012.pdf.

[60] Siemens. *SIMATIC - Function Block Diagram (FBD) for S7-300 and S7-400 Programming - Reference Manual*. 6ES7810-4CA10-8BW1. Mai 2010. URL: https://cache.industry.siemens.com/dl/files/487/45522487/att_61497/v1/s7fup__b.pdf.

[61] Siemens. *SIMATIC - Instuction list for S7-300 CPUs and ET 200 CPUs - Reference Manual*. 6ES7398-8FA10-8BA0. Juni 2011. URL: https://cache.industry.siemens.com/dl/files/679/31977679/att_81622/v1/s7300_parameter_manual_en-US_en-US.pdf.

[62] Siemens. *SIMATIC - Ladder Logic (LAD) for S7-300 and S7-400 Programming - Reference Manual*. 6ES7810-4CA08-8BW1. März 2006. URL: http://www.kntu.ac.ir/dorsapax/userfiles/file/Electrical/az-plc/Ladder_Logic.pdf.

[63] Siemens. *SIMATIC - Structured Control Language (SCL) for S7-300/S7-400 Programming - Manual*. 6ES7811-1CA02-8BA0. URL: https://cache.industry.siemens.com/dl/files/188/1137188/att_27471/v1/SCLV4_e.pdf.

[64] Dieter Spath u. a. *Produktionsarbeit der Zukunft-Industrie 4.0*. Fraunhofer Verlag Stuttgart, 2013. (Besucht am 01. 10. 2016).

[65] H. Stachowiak. *Allgemeine Modelltheorie*. Springer-Verlag, 1973. ISBN: 9783211811061.

[66] T. Stahl und M. Völter. *Modellgetriebene Softwareentwicklung: Techniken, Engineering, Management*. dpunkt-Verlag, 2005. ISBN: 9783898643108.

[67] Johannes Josef Stammel und R. Reussner. „Architekturbasierte Bewertung und Planung von Änderungsanfragen“. Diss. Karlsruhe: KIT-Bibliothek, 2015.

[68] Neville A. Stanton. „Hierarchical task analysis: Developments, applications, and extensions“. In: *Applied Ergonomics*. Special Issue: Fundamental Reviews 37.1 (Jan. 2006), S. 55–79. ISSN: 0003-6870. DOI: 10.1016/j.apergo.2005.06.003. URL: https://www.sciencedirect.com/science/article/pii/S0003687005000980 (besucht am 04. 02. 2017).

[69] Dave Steinberg u. a. *EMF: Eclipse Modeling Framework*. Pearson Education, 2008. (Besucht am 28. 03. 2017).

[70] *Strukturierter Text*. In: *Wikipedia*. Page Version ID: 155465263. 20. Juni 2016. URL: https://de.wikipedia.org/w/index.php?title=Strukturierter_Text&oldid=155465263 (besucht am 01. 02. 2017).

[71] Kishor S. Trivedi und Kalyanaraman Vaidyanathan. „Software reliability and rejuvenation: Modeling and analysis“. In: *IFIP International Symposium on Computer Performance Modeling, Measurement and Evaluation*. Springer, 2002, S. 318–345. URL: http://link.springer.com/chapter/10.1007/3-540-45798-4_14 (besucht am 31. 03. 2017).

[72] TUM. *Lehrstuhl für Automatisierung und Informationssysteme - Fakultät für Maschinenwesen - Technische Universität München*. The Pick and Place Unit – Demonstrator for Evolution in Industrial Plant Automation. 30. Okt. 2016. URL: http://www.ais.mw.tum.de/forschung/demonstratoren/ppu/ (besucht am 30. 10. 2016).

[73] *UML 2.4.1 Infrastructure*. URL: http://www.omg.org/spec/UML/2.4.1/Infrastructure/PDF/ (besucht am 28. 03. 2017).

[74] PROFIBUS Nutzerorganisation e. V. *PROFIBUS Systembeschreibung - Technologie und Anwendung*. URL: http://www.profibus.com/nc/download/technical-descriptions-books/downloads/profibus-technology-and-application-system-description/download/20714/ (besucht am 29. 03. 2017).

[75] Birgit Vogel-Heuser u. a. „Evolution of software in automated production systems: Challenges and research directions“. In: *Journal of Systems and Software* 110 (2015), S. 54–84. URL: http://www.sciencedirect.com/science/article/pii/S0164121215001818 (besucht am 08. 04. 2017).

[76] Birgit Vogel-Heuser u. a. „Interdisciplinary product line approach to increase reuse Analysis of the applicability of product lines for automation engineering in the machine manufacturing domain“. In: *AT-AUTOMATISIERUNGSTECHNIK* 63.2 (2015), S. 99–110.

[77] B. Vogel-Heuser u. a. „Maintenance effort estimation with KAMP4aPS for cross-disciplinary automated PLC-based Production Systems - a collaborative approach“. In: 2017.

[78] B. Vogel-Heuser u. a. „Researching Evolution in Industrial Plant Automation: Scenarios and Documentation of the Pick and Place Unit. Technical Report No. TUM-AIS-TR-01-14-02“. In: (2014). URL: https://mediatum.ub.tum.de/node?id=1208973 (besucht am 30. 10. 2016).

[79] B. Vogel-Heuser u. a. „Selected challenges of software evolution for automated production systems“. In: *2015 IEEE 13th International Conference on Industrial Informatics (INDIN)*. 2015 IEEE 13th International Conference on Industrial Informatics (INDIN). Juli 2015, S. 314–321. DOI: 10.1109/INDIN.2015.7281753.

[80] B. Vogel-Heuser u. a. „Towards a common classification of changes for information and automated production systems as precondition for maintenance effort estimation“. In: *2016 IEEE 14th International Conference on Industrial Informatics (INDIN)*. 2016 IEEE 14th International Conference on Industrial Informatics (INDIN). Juli 2016, S. 166–172. DOI: 10.1109/INDIN.2016.7819152.

A. Anhang

Abbildung A.1.: Vereinfachtes Metamodell der xPPU für Szenario 2 [77]

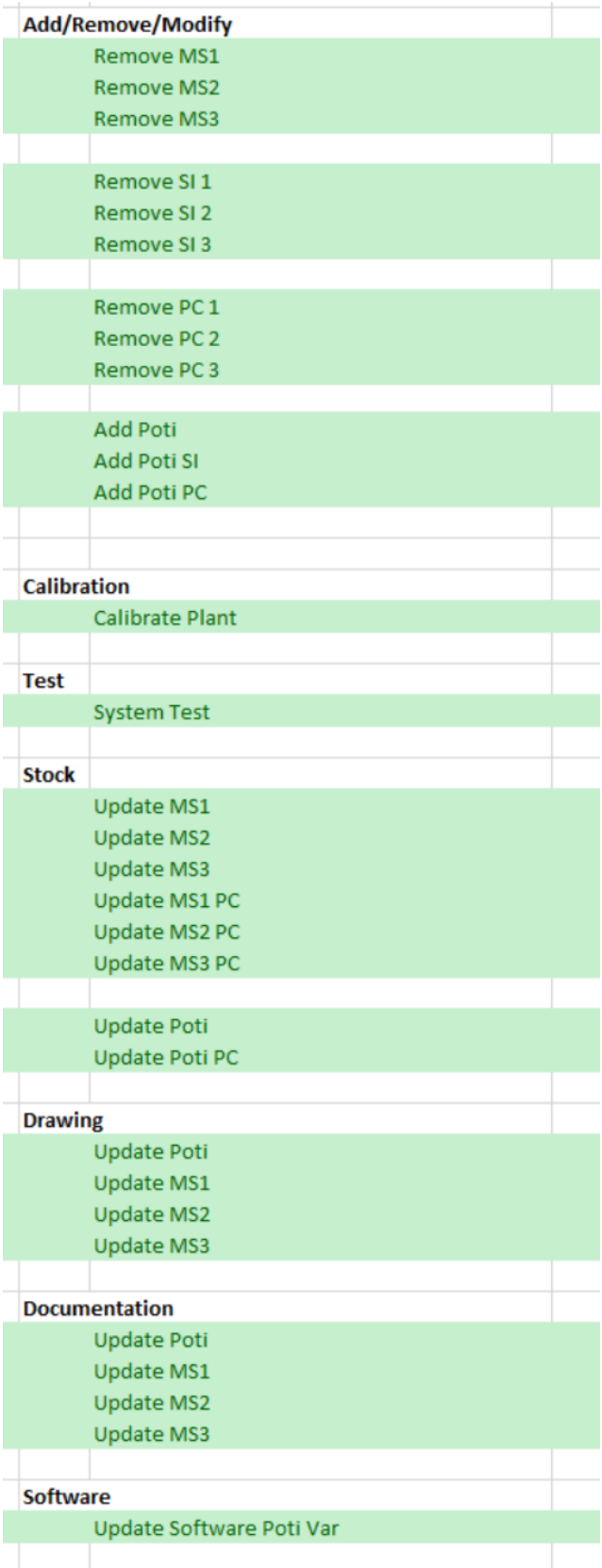

Abbildung A.2.: Manuell erstellte Aktivitätsliste für Szenario 1

Activity type	Activity description	Affected element
Architecture-related activi	Add PhysicalConnection ProductionCrane.TurningTable.Potentiometer.PC.	ProductionCrane.TurningTable.Potentiometer.PC
Update stocklist	StockList: Add stocklist (1 files) of interface ProductionCrane.TurningTable.Potentiometer.PC.	1 interface stocklist
Architecture-related activi	Add Potentiometer ProductionCrane.TurningTable.Potentiometer.	ProductionCrane.TurningTable.Potentiometer
Update documentation	Documentation: Add mainteneance (1 files) of component ProductionCrane.TurningTable.Potentiometer.	1 mainteneance documentation
Update CAD	ECAD: Add drawings (Component files) of component ProductionCrane.TurningTable.Potentiometer. PlantEngine	Component drawings of ProductionCrane.TurningTable.Potentiom
Update Software	Firmware of Element ProductionCrane.TurningTable.Potentiometer in ProgramType ReadMS	ProductionCrane.TurningTable.Potentiometer
Update stocklist	StockList: Add stocklist (1 files) of component ProductionCrane.TurningTable.Potentiometer.	1 component stocklist
Architecture-related activi	Add SignalInterface ProductionCrane.TurningTable.Potentiometer.SI.	ProductionCrane.TurningTable.Potentiometer.SI
Architecture-related activi	Add SignalInterface ProductionCrane.TurningTable.Potentiometer.SI.	ProductionCrane.TurningTable.Potentiometer.SI
Architecture-related activi	Remove MicroswitchModule ProductionCrane.TurningTable.MS.Conveyor.	ProductionCrane.TurningTable.MS.Conveyor
Update documentation	Documentation: Remove mainteneance (1 files) of component ProductionCrane.TurningTable.MS.Conveyor.	1 mainteneance documentation
Update CAD	ECAD: Remove drawings (Component files) of component ProductionCrane.TurningTable.MS.Conveyor. PlantEngi	Component drawings of ProductionCrane.TurningTable.MS.Convey
Update stocklist	StockList: Remove stocklist (1 files) of component ProductionCrane.TurningTable.MS.Conveyor.	1 component stocklist
Architecture-related activi	Remove MicroswitchModule ProductionCrane.TurningTable.MS.Stack.	ProductionCrane.TurningTable.MS.Stack
Update documentation	Documentation: Remove mainteneance (1 files) of component ProductionCrane.TurningTable.MS.Stack.	1 mainteneance documentation
Update CAD	ECAD: Remove drawings (Component files) of component ProductionCrane.TurningTable.MS.Stack. PlantEngineer	Component drawings of ProductionCrane.TurningTable.MS.Stack
Update stocklist	StockList: Remove stocklist (1 files) of component ProductionCrane.TurningTable.MS.Stack.	1 component stocklist
Architecture-related activi	Remove MicroswitchModule ProductionCrane.TurningTable.MS.Stamp.	ProductionCrane.TurningTable.MS.Stamp
Update documentation	Documentation: Remove mainteneance (1 files) of component ProductionCrane.TurningTable.MS.Stamp.	1 mainteneance documentation
Update CAD	ECAD: Remove drawings (Component files) of component ProductionCrane.TurningTable.MS.Stamp. PlantEngineer	Component drawings of ProductionCrane.TurningTable.MS.Stamp
Update stocklist	StockList: Remove stocklist (1 files) of component ProductionCrane.TurningTable.MS.Stamp.	1 component stocklist
Architecture-related activi	Remove PhysicalConnection ProductionCrane.TurningTable.MS.Conveyor.PC.	ProductionCrane.TurningTable.MS.Conveyor.PC
Update stocklist	StockList: Remove stocklist (1 files) of interface ProductionCrane.TurningTable.MS.Conveyor.PC.	1 interface stocklist
Architecture-related activi	Remove PhysicalConnection ProductionCrane.TurningTable.MS.Stack.PC.	ProductionCrane.TurningTable.MS.Stack.PC
Update stocklist	StockList: Remove stocklist (1 files) of interface ProductionCrane.TurningTable.MS.Stack.PC.	1 interface stocklist
Architecture-related activi	Remove PhysicalConnection ProductionCrane.TurningTable.MS.Stamp.PC.	ProductionCrane.TurningTable.MS.Stamp.PC
Update stocklist	StockList: Remove stocklist (1 files) of interface ProductionCrane.TurningTable.MS.Stamp.PC.	1 interface stocklist
Architecture-related activi	Remove SignalInterface ProductionCrane.TurningTable.MS.Conveyor.SI.	ProductionCrane.TurningTable.MS.Conveyor.SI
Architecture-related activi	Remove SignalInterface ProductionCrane.TurningTable.MS.Conveyor.SI.	ProductionCrane.TurningTable.MS.Conveyor.SI
Architecture-related activi	Remove SignalInterface ProductionCrane.TurningTable.MS.Stack.SI.	ProductionCrane.TurningTable.MS.Stack.SI
Architecture-related activi	Remove SignalInterface ProductionCrane.TurningTable.MS.Stack.SI.	ProductionCrane.TurningTable.MS.Stack.SI
Architecture-related activi	Remove SignalInterface ProductionCrane.TurningTable.MS.Stamp.SI.	ProductionCrane.TurningTable.MS.Stamp.SI
Architecture-related activi	Remove SignalInterface ProductionCrane.TurningTable.MS.Stamp.SI.	ProductionCrane.TurningTable.MS.Stamp.SI
Update Calibration	Calibrate Plant(s)	Calibration
Test execution	Test: Remove test (MainPlant testcases).	MainPlant test

Abbildung A.3.: Automatisch erstellte Aktivitätsliste für Szenario 1

Add/Remove/Modify

- Modify Conveyor0
- Modify Conveyor1
- Modify Conveyor2
- Modify Conveyor3
- Modify Conveyor4
- Modify Crane1
- Modify Crane2
- Modify Box1
- Modify Controller1
- Modify Cable CC1toConv0
- Modifiy Cable CC1toCrane1
- Modify Cable CC1toCrane2
- Modify Cable BoxtoCC1
- Modify Cable Crane1toConv1
- Modify Cable Crane1toConv2
- Modify Cable Crane2toConv3
- Modify Cable Crane2toConv4

Stock

- Update Conv0
- Update Conv1
- Update Conv2
- Update Conv3
- Update Conv4
- Update Crane1
- Update Crane2
- Update Box1
- Update CC1
- Update Cable Crane1toConv1
- Update Cable Crane1toConv2
- Update Cable Crane2toConv3
- Update Cable Crane2toConv4
- Update Cable CC1toConv0
- Update Cable CC1toCrane1
- Update Cable CC1toCrane2
- Update Cable BoxtoCC1

Drawing

- Update Con0
- Update Con1
- Update Con2
- Update Con3
- Update Con4
- Update Crane1
- Update Crane2
- Update Box1
- Update CC1

Documentation

- Update Con0
- Update Con1
- Update Con2
- Update Con3
- Update Con4
- Update Crane1
- Update Crane2
- Update Box1

Calibration

- Calibrate Plant

Test

- Test Plant

Abbildung A.4.: Manuell erstellte Aktivitätsliste für Szenario 2

Activity description	Affected element	Causing elements
Modify ProfibusDPBox.	BoxNetwork1	ProfibusDPBox "BoxNetwork1"
Documentation: Modify maintenance (1 files) of component BoxNetwork1.	1 maintenance documentation	
ECAD: Modify drawings (Component files) of component BoxNetwork1. PlantEngineer	Component drawings of BoxNetwork1	
StockList: Modify stocklist (1 files) of component BoxNetwork1.	1 component stocklist	
Modify ProfibusDPCable.	CC1_to_Conveyor0	SignalInterface "CC1.to.Conveyor0"
StockList: Modify stocklist (1 files) of component CC1_to_Conveyor0.	1 component stocklist	
Modify ProfibusDPCable.	CC1_to_Crane2	SignalInterface "CC1.to.Crane2", SignalInterface "Crane2.to.CC1"
StockList: Modify stocklist (1 files) of component CC1_to_Crane2.	1 component stocklist	
Modify ProfibusDPCable.	Crane2_to_Conveyor3	SignalInterface "Conveyor3.to.Crane2"
StockList: Modify stocklist (1 files) of component Crane2_to_Conveyor3.	1 component stocklist	
Modify ProfibusDPCable.	Crane2_to_Conveyor4	SignalInterface "Conveyor4.to.Crane2"
StockList: Modify stocklist (1 files) of component Crane2_to_Conveyor4.	1 component stocklist	
Modify ProfibusDPCable.	CC1_to_Crane1	SignalInterface "CC1.to.Crane", SignalInterface "Crane1.to.CC1"
StockList: Modify stocklist (1 files) of component CC1_to_Crane1.	1 component stocklist	
Modify ProfibusDPCable.	Crane1_to_Conveyor2	SignalInterface "Conveyor2.to.Crane1"
StockList: Modify stocklist (1 files) of component Crane1_to_Conveyor2.	1 component stocklist	
Modify ProfibusDPCable.	Crane1_to_Conveyor1	SignalInterface "Conveyor1.to.Crane1"
StockList: Modify stocklist (1 files) of component Crane1_to_Conveyor1.	1 component stocklist	
Modify ProfibusDPCable.	Box_to_CC1	SignalInterface "CC1.to.Box"
Calibrate Plant(s)	Calibration	
StockList: Modify stocklist (1 files) of component Box_to_CC1.	1 component stocklist	
Test: Modify test (MainPlant testcases).	MainPlant test	
Modify ProfibusDPMaster.	CC1_Master	SignalInterface "CC1.to.Box", SignalInterface "CC1.to.Conveyor0", SignalInterface "CC1.to.Crane", SignalInterface "CC1.to.Crane2"
StockList: Modify stocklist (1 files) of component CC1_Master.	1 component stocklist	
ECAD: Modify drawings (Component files) of component CC1_Master. PlantEngineer	Component drawings of CC1_Master	
Modify ProfibusDPSlave.	Crane1_Slave	SignalInterface "Crane1.to.CC1"
Documentation: Modify maintenance (1 files) of component Crane1_Slave.	1 maintenance documentation	
ECAD: Modify drawings (Component files) of component Crane1_Slave. PlantEngineer	Component drawings of Crane1_Slave	
StockList: Modify stocklist (1 files) of component Crane1_Slave.	1 component stocklist	
Modify ProfibusDPSlave.	Conveyor1_Slave	SignalInterface "Conveyor1.to.Crane1"
Documentation: Modify maintenance (1 files) of component Conveyor1_Slave.	1 maintenance documentation	
ECAD: Modify drawings (Component files) of component Conveyor1_Slave. PlantEngineer	Component drawings of Conveyor1_Slave	
StockList: Modify stocklist (1 files) of component Conveyor1_Slave.	1 component stocklist	
Modify ProfibusDPSlave.	Conveyor2_Slave	SignalInterface "Conveyor2.to.Crane1"
Documentation: Modify maintenance (1 files) of component Conveyor2_Slave.	1 maintenance documentation	
ECAD: Modify drawings (Component files) of component Conveyor2_Slave. PlantEngineer	Component drawings of Conveyor2_Slave	
StockList: Modify stocklist (1 files) of component Conveyor2_Slave.	1 component stocklist	
Modify ProfibusDPSlave.	Conveyor4_Slave	SignalInterface "Conveyor4.to.Crane2"
Documentation: Modify maintenance (1 files) of component Conveyor4_Slave.	1 maintenance documentation	
ECAD: Modify drawings (Component files) of component Conveyor4_Slave. PlantEngineer	Component drawings of Conveyor4_Slave	
StockList: Modify stocklist (1 files) of component Conveyor4_Slave.	1 component stocklist	
Modify ProfibusDPSlave.	Conveyor0_Slave	SignalInterface "CC1.to.Conveyor0"
Documentation: Modify maintenance (1 files) of component Conveyor0_Slave.	1 maintenance documentation	
ECAD: Modify drawings (Component files) of component Conveyor0_Slave. PlantEngineer	Component drawings of Conveyor0_Slave	
StockList: Modify stocklist (1 files) of component Conveyor0_Slave.	1 component stocklist	
Modify ProfibusDPSlave.	Crane2_Slave	SignalInterface "Crane2.to.CC1"
Documentation: Modify maintenance (1 files) of component Crane2_Slave.	1 maintenance documentation	
ECAD: Modify drawings (Component files) of component Crane2_Slave. PlantEngineer	Component drawings of Crane2_Slave	
StockList: Modify stocklist (1 files) of component Crane2_Slave.	1 component stocklist	
Modify ProfibusDPSlave.	Conveyor3_Slave	SignalInterface "Conveyor3.to.Crane2"
Documentation: Modify maintenance (1 files) of component Conveyor3_Slave.	1 maintenance documentation	
ECAD: Modify drawings (Component files) of component Conveyor3_Slave. PlantEngineer	Component drawings of Conveyor3_Slave	
StockList: Modify stocklist (1 files) of component Conveyor3_Slave.	1 component stocklist	

Abbildung A.5.: Automatisch erstellte Aktivitätsliste für Szenario 2