AutomationML in a Nutshell

Arndt Lüder und Nicole Schmidt

Zusammenfassung

Die Welt der Produktionssysteme ist an einem Wendepunkt. Die wachsende Bedeutung der Kundenwünsche und die wachsende Geschwindigkeit des technischen Fortschritts haben Produktionssysteminhaber dazu gebracht, die Flexibilität von Produktionssystemen hinsichtlich Produktportfolio und Ressourcennutzung auszuweiten (Terkaj et al. 2009). Jedoch ist diese Flexibilitätserweiterung nicht kostenlos zu haben. Neue Vorgehensweisen und Methoden des Entwurfes und der Nutzung von Produktionssystemen haben sich als notwendig erwiesen, wie sie in der Industrie 4.0 Initiative anvisiert werden (Kagermann et al.; Jasperneite 2012).

Industrie 4.0 fordert eine verstärkte Integration in verschiedensten Richtungen bezogen auf die Struktur und Entwurf/Erstellung/Nutzung von Produktionssystemen. So empfiehlt es eine verstärkte Integration der verschiedenen Lebenszyklusphasen eines Produktionssystems, stärkere Integration der verschiedenen Ebenen der Automatisierungspyramide von der Feldebene bis zur Unternehmenssteuerung und die Integration entlang der Entwurfskette des Produktionssystems, d. h. die Abfolge von Aktivitäten, die von Ingenieuren mit entsprechenden Entwurfswerkzeugen auszuführen sind (Biffl et al. 2017).

Die zunehmende Flexibilität der Produktionssysteme erzwingt eine höhere Frequenz an Entwurfsaktivitäten (Neubau und Umbau). Deshalb nimmt die Bedeutung des Entwurfs im Lebenszyklus des Produktionssystems zu, dessen Anteile an Lebenszyklus und Kosten des Produktionssystems steigen. Die Integration von Ingenieuraktivitäten und ihre beteiligten Werkzeuge entlang der

A. Lüder (✉)
Lehrstuhl für Produktionssysteme und -automatisierung, Universität Magdeburg, Magdeburg, Deutschland
E-Mail: arndt.lueder@ovgu.de

N. Schmidt
PLM::Production, Daimler Protics GmbH, Leinfelden-Echterdingen, Deutschland
E-Mail: nicole.s.schmidt@daimler.com

B. Vogel-Heuser et al. (Hrsg.), *Handbuch Industrie 4.0*, Springer Reference Technik,
https://doi.org/10.1007/978-3-662-45537-1_61-2

Entwurfskette sollen ein Mittel sein, Zeit und Kosten des Entwurfs durch die Vermeidung von unnötigen Wiederholungen von Entwurfsaktivitäten zu sparen, eine Zunahme an Kontinuität der Entwurfswerkzeugketten sicherzustellen und eine Verbesserung der Zusammenarbeit unter den Ingenieuren (um nur einige erwartete Einflüsse zu nennen) zu erreichen (Biffl et al. 2017).

Eine Mittel, die Integration von Entwurfsaktivitäten und Werkzeugen entlang der Entwurfsketten des Produktionssystems zu ermöglichen und außerdem die Verwendung von Entwurfsdaten innerhalb der Nutzungsphase eines Produktionssystems möglich zu machen, ist ein geeignetes Datenaustauschformat. Folgend der Industrie 4.0 Roadmap (DIN/DKE 2018) muss ein solches Datenformat entwickelt werden. In diesem Paper wird das Datenaustauschformat AutomationML betrachtet. Um eine Bewertung der Anwendbarkeit von AutomationML im Industrie 4.0 Kontext zu ermöglichen, soll der Umfang der Darstellbarkeit von Entwurfsdaten mit AutomationML detailliert untersucht werden.

1 Einleitung

Der Entwurf von Produktionssystemen ist ein komplexer Prozess, der verschiedene Ingenieure aus verschiedenen Entwurfsbereichen einbezieht, die verschiedene Entwurfsaktivitäten durchführen und verschiedene Entwurfsartefakte nutzen/entwerfen, die nötig sind, um letztendlich in der Lage zu sein, ein Produktionssystem zu erstellen, zu nutzen und zu warten (Terkaj et al. 2009).

Wie unterschiedliche Untersuchungen gezeigt haben, umfasst der Entwurf von Produktionssystemen einen großen Anteil manueller Tätigkeiten (Alonso-Garcia et al. 2008). Jedoch müssen verschiedene Entwurfsaktivitäten innerhalb von unterschiedlichen Entwurfswerkzeugen wiederholt werden, da es keine geeigneten Instrumente für einen Datenaustausch zwischen diesen Werkzeugen gibt (Drath et al. 2011; Schmidt et al. 2014). Um dies zu vermeiden, werden Möglichkeiten für einen verlustfreien Datenaustausch entlang der kompletten Entwurfswerkzeugkette benötigt.

Um einen verlustfreien Datenaustausch zu gewährleisten, wurden verschiedene Herangehensweisen betrachtet. Viele Entwurfsorganisationen und – unternehmen haben ihre eigenen Softwarelösungen entwickelt. Bei der Betrachtung all dieser Ansätze können drei primäre Philosophien für die Sicherstellung eines verlustfreien Datenaustausches entlang der Entwurfsaktivitäten und -werkzeugketten identifiziert werden: „One Tool For All“, „Best of Breed“ und „Integration Framework“. Jedes von ihnen baut auf anderen Datenmodellen, Datenaustauschmethoden und -technologien und Softwaresystemen auf. Sie besitzen jeweils ihre besonderen Vor- und Nachteile (Hundt und Lüder 2012).

Innerhalb der „Best of Breed“ Philosophie (siehe Abb. 1), die für gewöhnlich in Entwurfsprojekten von KMUs und/oder Projekten, an denen mehr als ein Unternehmen beteiligt ist, angewendet wird, wie auch innerhalb der „Integration Framework“ Philosophie (siehe Abb. 2), werden existierende Entwurfswerkzeuge über einen

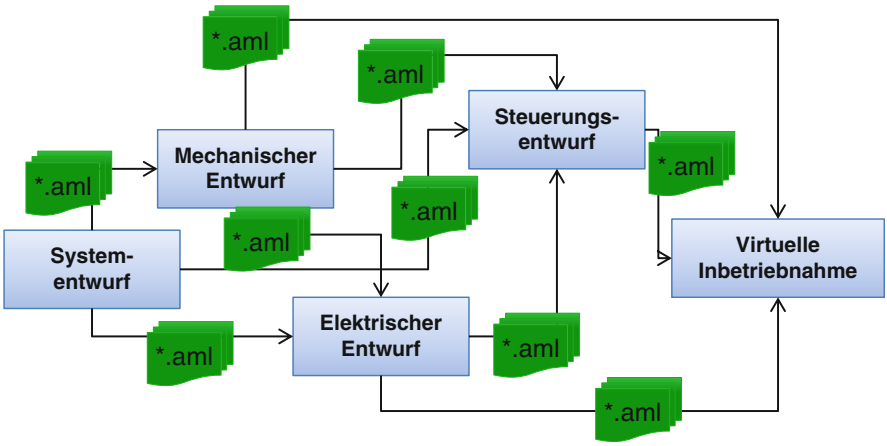

Abb. 1 Beispiel für ein „Best of Breed" basiertes Entwurfsnetzwerk

bilateralen Datenaustausch oder über eine zentralisierte Datenvermittlung kombiniert.

Um den benötigten Datenaustausch zwischen sich möglicherweise ändernden Entwurfswerkzeugen zu gewährleisten, könnte ein standardisiertes Datenaustauschformat wie AutomationML (Drath 2010) und STEP (Xu und Nee 2009) bevorzugt werden. Ein derartiges Datenformat muss in der Lage sein, alle oder zumindest einen Großteil der Informationen abzubilden, die innerhalb des Entwurfsprozesses des Produktionssystems benötigt und/oder erstellt werden.

Für ein solches Datenaustauschformat gibt es eine Reihe von (manchmal widersprüchlichen) Anforderungen, die erfüllt sein müssen:

- Das Datenformat sollte für verschiedene Anwendungsfälle angepasst werden können und flexible in Bezug auf Erweiterungen und Veränderungen sein.
- Die Datendarstellung sollte effizient sein.
- Die Datendarstellung sollte für Menschen lesbar sein.
- Die Datendarstellung sollte auf internationalen Standards basieren.

Zudem sollte ein Datenaustauschformat, das in beiden Philosophien anwendbar sein soll, Anforderungen hinsichtlich der Unterstützung von Methoden des Versionsmanagements, Vollständigkeitsmanagements und Konsistenzmanagements erfüllen sowie Anwendungsweisen zur Gewährleistung einer schrittweisen Einführung (Migration) unter Berücksichtigung der üblichen Arbeitsweisen der beteiligten Ingenieure (Engineeringhabit) unterstützen (Lüder et al. 2019).

Diese Voraussetzungen erfüllen XML basierte Datenformate (Drath et al. 2011) zumeist problemlos.

Folgend (Diedrich et al. 2011) erfordert der Datenaustausch zwischen Entwurfswerkzeugen zwei unterschiedliche Standardisierungsniveaus, die syntaktische und die semantische Ebene. Auf der syntaktischen Ebene wird die korrekte informati-

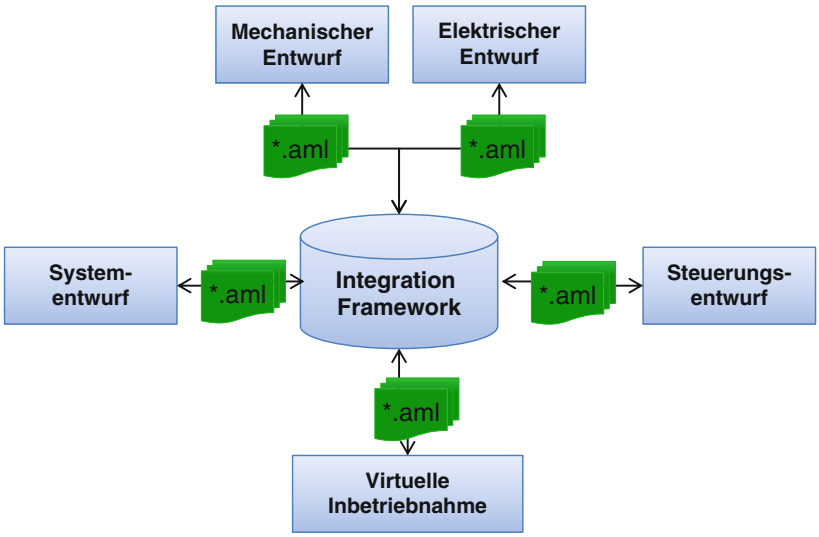

Abb. 2 Beispiel für ein „Integration Framework“ basiertes Entwurfsnetzwerk

onstechnische Darstellung der Datenobjekte innerhalb des Datenaustauschformats definiert. Dabei wird das Vokabular des Datenaustausches bereitgestellt. Im Gegensatz dazu wird auf der semantischen Ebene die Interpretation von Datenobjekten, d. h. ihre Bedeutung innerhalb der Konzeptualisierung von Objekten innerhalb der Entwurfswerkzeugketten, definiert.

Datenaustauschformate können die Umsetzung dieser zwei Ebenen auf zwei Arten angehen. Entweder werden Syntax und Semantik gemeinsam definiert, wie es bei der Entwicklung von STEP umgesetzt wurde, oder Syntax und Semantik werden getrennt definiert, wie bei AutomationML realisiert. Da die getrennte Definition der Semantik größere Flexibilität und Anpassungsvermögen des Datenaustauschformates für den Anwendungsfall ermöglicht, erscheint dieser Ansatz zu bevorzugen zu sein.

Nachfolgend wird die Automation Markup Language (AutomationML) detailliert beschrieben. Es wird dargestellt,

- welche Entwurfsprozesse und Entwurfsdaten bezogen auf die Anforderungen der Industrie 4.0 durch AutomationML in der aktuellen Version abgedeckt werden (Abschn. 2),
- was die generelle Architektur von AutomationML ist (Abschn. 4),
- wie die Topologie eines Produktionssystems, die seine Hierarchie an Systemkomponenten und Geräten umfasst, in AutomationML dargestellt wird (Abschn. 5),
- wie ein Modellelement in AutomationML um semantische Aspekte angereichert werden kann (Abschn. 6),
- wie Geometrie- und Kinematikinformationen in AutomationML modelliert werden (Abschn. 7),

- wie Verhaltensinformationen modelliert werden (Abschn. 8),
- wie Netzwerke in AutomationML modelliert werden (Abschn. 9),
- wie zusätzliche Informationen in Bezug auf Systemkomponenten und Geräte in einem AutomationML Modell ergänzt werden können (Abschn. 10),
- was berücksichtigt werden soll, wenn AutomationML für die Implementierung der Integration entlang von Entwurfsketten im Industrie 4.0 Kontext angewendet werden soll (Abschn. 11) und schlussendlich,
- welche Rolle AutomationML im Rahmen der Industrie 4.0 noch spielen kann (Abschn. 12).

2 Abgedeckte Entwurfsprozess und Entwurfsdaten

AutomationML wurde vorrangig für die Nutzung im Bereich des Entwurfs von Produktionssystemen und deren Inbetriebnahme entwickelt. In Anlehnung an die Betrachtung der Lebenszyklen verschiedener Systeme, die innerhalb eines Produktionssystems Relevanz besitzen (Produktionssystem, Produktionstechnologie, Produkt, Auftrag), wie sie in (VDI/VDE) gegeben sind, sind die für die Anwendung von AutomationML relevanten Phasen die Komponenten- und Technologie-entwicklungsphase, die für die Planung und Implementierung der Komponenten und Geräte des Produktionssystems verantwortlich sind, die Entwurfsphase des Produktionssystems, in der das detaillierte Design eines Produktionssystems ausführt wird, und die Inbetriebnahmephase einschließlich Systemtest, Installation und Ramp-up (siehe Abb. 3). Diese Phasen bilden gemeinsam den Fabrikplanungsprozess.

Hinsichtlich des Fabrikplanungsprozesses gibt es unterschiedliche, jedoch stark ähnliche Entwurfsprozesse, die in der Literatur beschrieben sind (Lüder et al. 2011). Ihre Unterschiede beziehen sich zumeist auf die Fokussierung auf spezifische Aspekte des Fabrikplanungsprozesses je nach Anwendungsfall (VDI/VDE). Abb. 4 zeigt einen Überblick über den aggregierten Prozess. Er besteht aus den fünf grundlegenden Phasen Analysephase, Basisplanungsphase, Detailplanungsphase, Systemintegrationsphase und Aufbau- und Nutzungsphase.

Ziel der Analysephase ist die Sammlung detaillierter Anforderungen an das zu entwerfende Produktionssystem, die sich zum einen aus dem zu produzierenden Produkt und zum anderen aus den für die Produktion notwendigen Prozessen ergeben. Zudem sollen weitere Anforderungen wie ökonomische Rahmenbedingungen, rechtliche Forderungen, Umweltschutzstandards, etc. gesammelt werden. Dazu beinhaltet die Analysephase die Aktivitäten Anforderungsentwurf und Systemprozessplanung, in denen die Anforderungen gesammelt, klassifiziert und konsistent beschrieben sowie alle Systemprozesse einschließlich notwendiger Herstellung- und Unterstützungsprozesse, die als konsistentes Funktionsmodell dargestellt werden. Das Ergebnis dieser Phase bildet die Prozessbeschreibung des Produktionssystems sowie die den einzelnen Prozessen bzw. Prozessteilen zugeordneten Anforderungen.

Die Basisplanungsphase deckt die Anlagengrobplanung des Produktionssystems ab. Sie berücksichtigt dabei noch keine Implementierungsdetails des später zu erstellenden Produktionssystems wie das konkrete Layout der Hallenfläche, auf

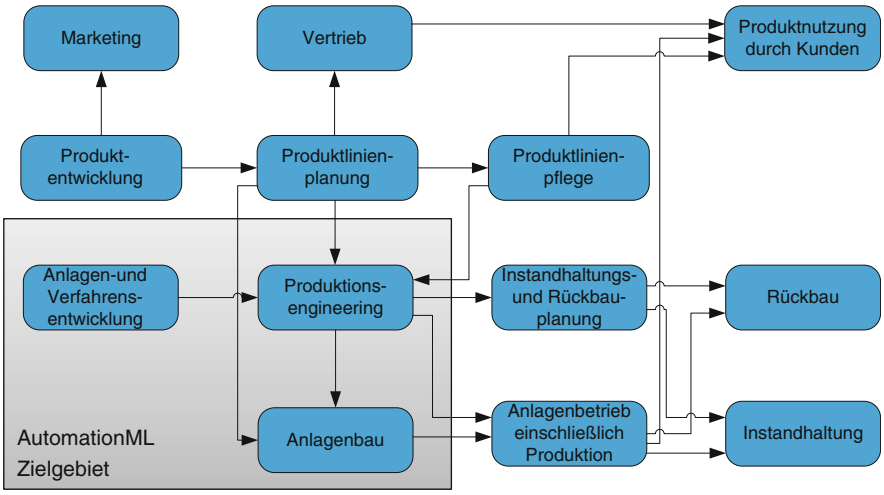

Abb. 3 Betrachteter Entwurfsprozess

der es aufzubauen ist. Dazu enthält diese Phase die Aktivitäten Komponentenauswahl, detaillierte Systemprozessplanung und Verhaltenssimulation. In der Komponentenplanung werden die technischen Mittel (Produktionsressourcen) zur Ausführung der einzelnen Prozesse bzw. Prozessteile aus der Prozessbeschreibung des Produktionssystems ausgewählt. Auf Basis der gewählten technischen Mittel kann dann der Produktionsprozess detailliert werden. Dabei werden die in den gewählten Produktionsressourcen ausführbaren Produktionsprozesse auf die entsprechenden Teile des notwendigen Produktionsprozesses abgebildet und notwendige weitere Prozessschritte (insbesondere sekundäre Prozesse) spezifiziert. Ist dies geschehen, können die gewählten Prozesse auf den gewählten technischen Mitteln simuliert werden. Damit können insbesondere die ökonomischen Anforderungen an das Produktionssystem validiert werden. Das Ergebnis der Basisplanungsphase bildet dann die validierte Menge von ausgewählten Produktionsressourcen und die detaillierten Spezifikation der Prozesse, die auf ihnen auszuführen sind.

Die Detailplanungsphase bezieht sich auf den detaillierten Entwurf des Produktionssystems, das sogenannte Functional Engineering. Hier werden alle notwendigen Planungsunterlagen und Realisierungsdetails entwickelt, die für einen erfolgreichen Aufbau und eine erfolgreiche Implementierung des Produktionssystems notwendig sind und die spezifischen Rahmenbedingungen der Umsetzung (zum Beispiel das nutzbare Hallenlayout) mit berücksichtigen. Damit umfasst diese Phase unter anderem den mechanischen, elektrischen, medienbezogen, steuerungstechnischen, netzwerktechnischen, und menschenbezogenen Entwurf des Produktionssystems. Teil dieser Phase sind die Aktivitäten mechanischer Entwurf, elektrischer Entwurf, Rohrleitungsplanung, Steuerungsentwurf, MMI Entwurf, Roboterprogrammierung und – simulation, Leitsystementwurf, Prozessanpassungsplanung und virtuelle Inbetriebnahme.

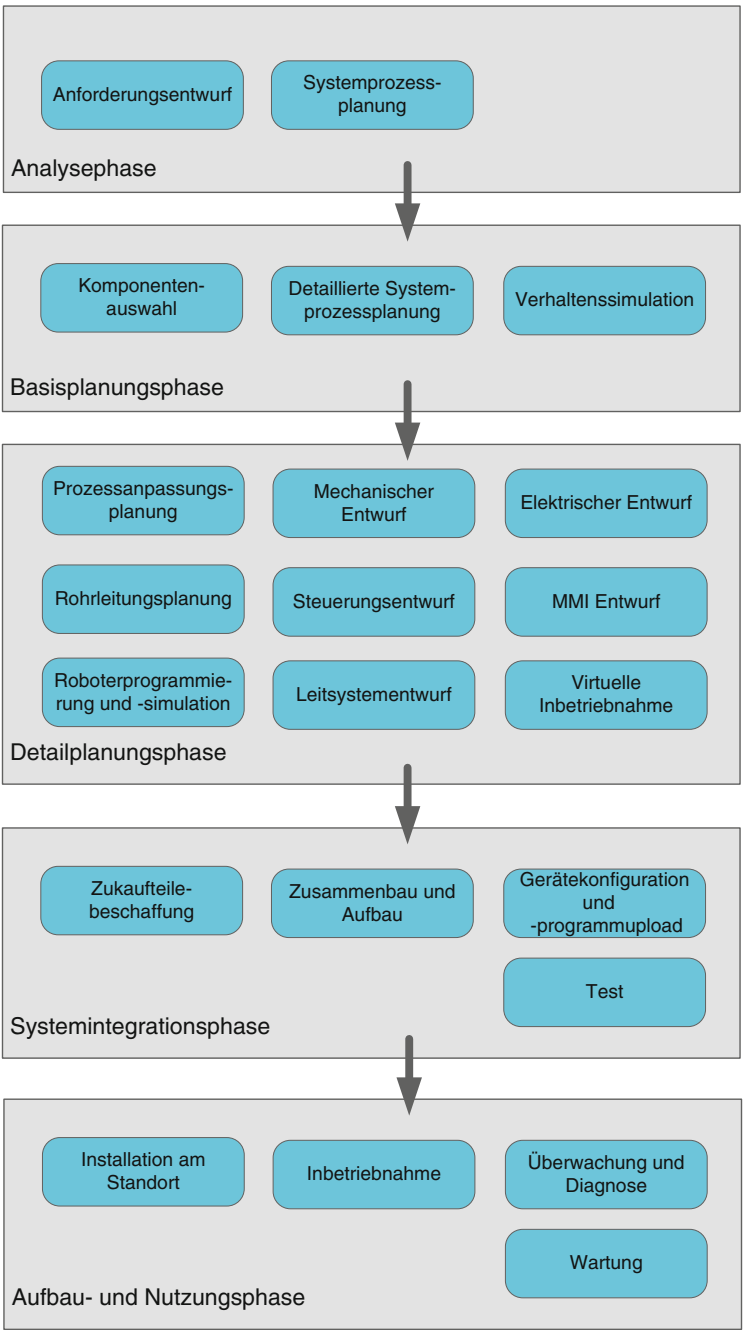

Abb. 4 Fabrikplanungsprozess

Der mechanische Entwurf spezifiziert die mechanische Struktur des Produktionssystems, die für die Ausführung der spezifizierten Produktionsprozesse notwendig ist. Dafür werden alle mechanischen Komponenten des Produktionssystems einschließlich der notwendigen Gerätetechnik (Sensoren und Aktoren) ausgewählt und im System positioniert. Der elektrische Entwurf ist verantwortlich für den Detailentwurf des gesamten elektrischen Systems des Produktionssystems. Dies schließt unter anderem die elektrische Verkabelung und das Kommunikationssystem und dessen Geräte mit ein und generiert damit entsprechende Signallisten für die Steuerungstechnik. Die Rohrleitungsplanung erstellt die detaillierte Beschreibung der hydraulischen, pneumatischen, etc. Komponenten des Produktionssystems, die sich auf die Nutzung von technischen Medien beziehen und beinhaltet damit neben den Rohrleitungssystemen auch die Rohrleitungsanschlüsse und die entsprechenden Geräte. Der Steuerungsentwurf hat zum Ziel, die Implementierung des gesamten Steuerungsprogrammcodes für die speicherprogrammierbaren Steuerungen des Produktionssystem sicherzustellen, der für die Gewährleistung eines sicheren, optimierten, und effizienten Systemverhaltens notwendig ist. In der Roboterprogrammierung und – simulation erfolgt analog die Erstellung des Steuerungscodes für Robotersteuerungen. Hier kommt zusätzlich die Validierung dieses, durch den Code erreichten, Bewegungsverhaltens mit simulativen Mitteln hinzu. Im MMI Entwurf und im Leitsystementwurf werden alle Mensch-Maschine-Schnittstellen einschließlich der notwendigen Gerätetechnik geplant und der dafür notwendige Programmcode erstellt und damit die Möglichkeiten des Menschen zum Eingriff in den Produktionsprozess festgelegt. Parallel zu allen genannten Aktivitäten (die partiell aufeinander aufbauen) erfolgt in der Prozessanpassungsplanung eine kontinuierliche Anpassung der Prozessplanung und der Layoutplanung an sich ändernde Anforderungen an das Produktionssystem als auch an die Ergebnisse der anderen Entwurfsarbeiten. So kann zum Beispiel eine spezifische mechanische Realisierung eines Prozesses weitere Unterstützungsprozesse erfordern. Am Ende können in der virtuelle Inbetriebnahme die entwickelten Beschreibungen des Produktionssystems, die Steuerungscodes, die Schnittstellen, etc. auf dem Wege der softwarebasierten Simulation überprüft werden. Ergebnis der Detailplanungsphase sind dann die validierten MCAD, ECAD und Fluidikpläne, Gerätelisten, Verkabelungslisten, Klemmlisten, Installationsanweisungen, Steuerungscodes, Konfigurationsanweisungen, etc. die für die physische Erstellung des Produktionssystems notwendig sind.

Ziel der Systemintegrationsphase ist die komponentenweise Installation des Produktionssystems auf der Basis der in der Detailplanungsphase erstellten Unterlagen. Dazu werden alle notwendigen Teile des Produktionssystems in der Zukaufteilebeschaffung beschafft, alle Komponenten zusammen- und aufgebaut, die verschiedenen Steuerungsgeräte konfiguriert, der Programmcode für speicherprogrammierbare Steuerungen, Robotersteuerungen, MMI, etc. geladen und letztendlich die Produktionssystemkomponenten getestet. Das Ergebnis der Systemintegrationsphase sind aufgebaute, in Betrieb genommene und vom Anlagennutzer abgenommene Produktionssystemkomponenten.

Diese werden dann in der finalen Aufbau- und Nutzungsphase wieder ab- und am intendierten Produktionsstandort wieder aufgebaut. In der Inbetriebnahme werden

sie dann als vollständiges Produktionssystem in Betrieb genommen und entsprechen einer Anfahrserie produziert. War dies erfolgreich, kann das Produktionssystem wie gewünscht genutzt werden. Um diese Nutzung möglichst lange sicherzustellen, erfolgen während der Nutzung die Überwachung, Diagnose und (ggf.) die Wartung des Produktionssystems.

Wie in Abb. 4 dargestellt ist, hängen die verschiedenen Entwurfsaktivitäten voneinander ab. Insbesondere benötigen Entwurfsaktivitäten Ergebnisse anderer Entwurfsaktivitäten. In jeder von ihnen werden aktivitätsspezifische Entwurfswerkzeuge verwendet, die üblicherweise an die Arbeiten in den Entwurfsaktivitäten optimal angepasst sind, d. h. optimal für eine effiziente und fehlerfreie Erstellung der Entwurfsartefakte und das dazu notwendige Treffen von entsprechende Entwurfsentscheidungen geeignet sind (Hundt und Lüder 2012). Diese Entwurfswerkzeuge beziehen sich auf spezielle Modellierungsmittel und besitzen dafür optimierte interne Datenmodelle, die dem Werkzeug und dessen Softwarestruktur entsprechen. Dies macht es sehr schwer, einen konsistenten und verlustfreien Informations- und Datenfluss (Austausch von digitalen Artefakten oder Teilen von ihnen) entlang der Werkzeugketten[1] des Fabrikplanungsprozesses sicherzustellen (Drath et al. 2011).

Um den konsistenten und verlustfreien Datenaustausch innerhalb der Werkzeugketten des Fabrikplanungsprozesses mit einem Datenaustauschformat wie AutomationML sicherstellen zu können, muss dieses Datenformat alle Informationen abbilden können, die in mindestens zwei Entwurfsaktivitäten und damit in mindestens zwei Werkzeugen der Werkzeugketten relevant sind. Fasst man die Entwurfsaktivitäten der oben genannten fünf Entwurfsphasen zusammen, so ergibt sich die nachfolgende Mindestmenge abzubildender Entwurfsinformationen:

- Topologieinformationen: Diese Inforationsmenge beschreibt die hierarchische Strukturierung des Produktionssystems von der Produktionssystemebene, über die Zell- und Ressourcenebenen bis hinunter zu den Ebenen der Geräte und mechanischen Bauteile (Kiefer et al. 2006). Sie deckt dabei neben den einzelnen Hierarchieelementen auch deren Relationen und beschreibenden Eigenschaften ab.
- Informationen bezogen auf die mechanischen Eigenschaften: Diese Informationsmenge beschreibt die mechanische Konstruktion des Produktionssystems einschließlich seiner geometrischen und kinematischen Eigenschaften. Üblicherweise wird sie als technische Zeichnung eines MCAD Werkzeuges entwickelt. Zudem enthält diese Informationsmenge physikalische Eigenschaften des Produktionssystems und seiner Teile wie Kräfte, Geschwindigkeiten und Drehwinkel sowie chemische Eigenschaften wie Materialinformationen.
- Informationen bezogen auf die elektrischen, pneumatischen und hydraulischen Eigenschaften: Diese Informationsmenge beschreibt die elektrische und fluidi-

[1]In diesem Papier wird der Begriff Entwurfskette verwendet unabhängig vom Fakt, dass die Entwurfsaktivitäten in einem realen Fabrikplanungsprozesses ein Netzwerk mit Parallelitäten, Nebenläufigkeiten, Abhängigkeiten und Zyklen bilden und diese in den Werkzeugketten abgebildet werden müssen.

sche Konstruktion des Produktionssystems einschließlich der Verkabelung und Verrohrung, wie sie mit ECAD und FCAD Werkzeugen erstellt werden können. Dazu umfasst sie zum einen die verbundenen Komponenten und zum anderen die Verbindungen zwischen ihnen mit ihren Schnittstellen und den jeweiligen Elektrik und Fluidik bezogenen Eigenschaften.
- Informationen bezogen auf die Funktionen des Produktionssystems: Diese Informationsmenge dient der Charakterisierung der Funktionen des Produktionssystems und seiner Komponenten. Dazu umfasst sie Verhaltensmodelle des ungesteuerten (physikalisch/chemisch/etc. möglichen) Verhaltens sowie des gesteuerten Verhaltens. Zu diesen kommen die funktionalen Parameter und die technischen Parameter. In von dieser Informationsmenge betrachteten Funktionen beziehen sich dabei nicht nur auf die in der Produktion notwendigen Funktionen, sondern auch auf alle notwendigen Hilf-, Diagnose-, Wartungs-, etc. Funktionen.
- Informationen bezogen auf die Steuerung des Produktionssystems: Diese Informationsmenge umfasst alle steuerungsgerätebezogenen Informationen. Dies sind insbesondere die Hardwarekonfiguration, Steuerungscode und Steuerungsparameter.
- Weitere Informationen: Diese Informationsmenge subsumiert weitere notwendige Informationen wie betriebswirtschaftlich relevante Informationen wie Herstellerartikelnummern oder Preise, organisatorische Informationen wie Montage- und Wartungsanleitungen, Handbücher usw.

Die genannten Informationsmengen sind noch einmal in Abb. 5 dargestellt. AutomationML ist in der Lage, all diese Informationsmengen abzubilden, wie in den nächsten Abschnitten verdeutlicht wird.

3 Anwendungsbeispiel

Nachfolgend wird die Abbildung der genannten Informationsmengen mit AutomationML beschrieben. Neben den abstrakten Abbildungsregeln wird dabei auch ein Beispiel vorgestellt, das sich auf ein auf Fischertechnik basierendes Labormodell eines Fertigungssystems bezieht. Dieses Labormodell steht am Institut für Arbeitswissenschaften, Fabrikautomatisierung und Fabrikbetrieb der Otto-von-Guericke Universität Magdeburg und besteht aus drei geschlossenen Kreisen von Transportmodulen (acht Drehtische, acht Transportbänder), wie in Abb. 6 dargestellt. In jedem dieser drei Kreise befindet sich eine Bearbeitungsmaschine mit je drei unterschiedlichen Effektoren, um Bearbeitungswerkzeuge darzustellen. Die Sensoren und Aktoren sind auf drei verschiedenen Modbus Buskopplern verschaltet und diskreten Signalen zugeordnet. Als Steuerung wird eine IEC 61131 basierte Soft-SPS auf Basis eines Raspberry-Pi verwendet.

Abb. 5 Abzubildende Informationsmengen

Für die Darstellungen in diesem Papier wird nur ein kleiner Ausschnitt dieses Labormodells verwendet, wie er in Abb. 7 schematisch gezeigt ist. Dieser Ausschnitt beinhaltet einen Drehtisch, der zwei Geräte enthält: einen Induktivsensor zur Werkstückerkennung und einen Antrieb zur Bandbewegung. Alle anderen Elemente des Drehtisches werden vernachlässigt. Die beiden betrachteten Geräte besitzen mindestens einen Anschlusspunkt (Pin), über den sie mit einem modularen Buskoppler verdrahtet werden können. Der Buskoppler wird durch einen modularen ModbusTCP Koppler repräsentiert, der die notwendige Hard- und Software besitzt, um die einzelnen Klemmpunkte des Buskopplers (Pins) auf entsprechende Registereinträge in einem Modbus TCP Server abzubilden. Der Buskoppler wiederum ist mittels eines Ethernet Kabels mit einem Raspberry-Pi verbunden. Der Raspberry-Pi führt in der auf ihm laufenden Soft-SPS das Steuerungsprogramm zur Ansteuerung der Sensoren und Aktoren des Drehtisches aus, mit dem die Sensoren und Aktoren angesteuert werden.

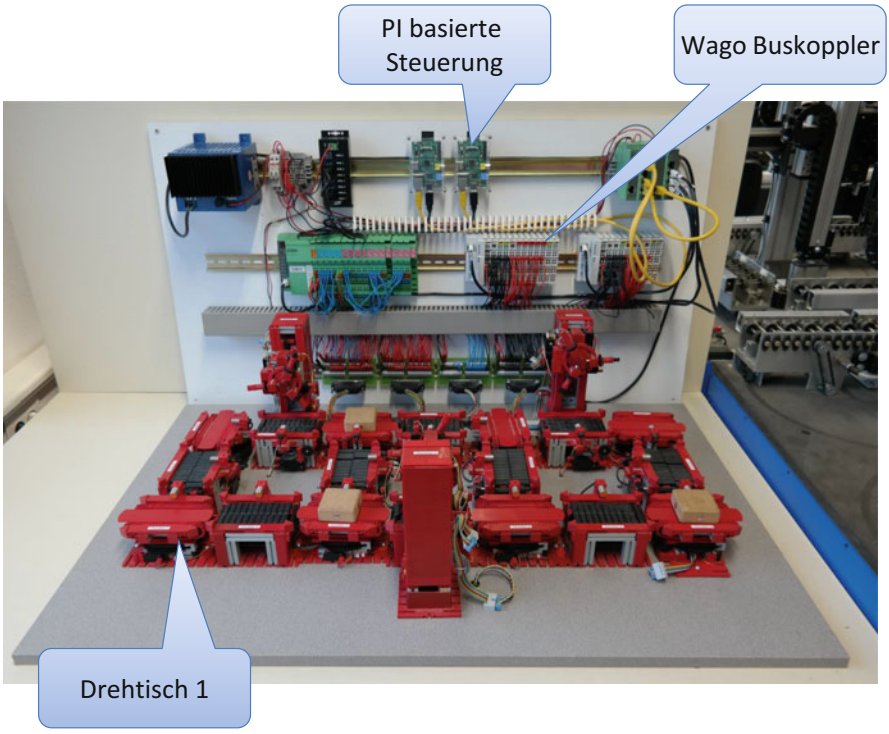

Abb. 6 Verwendetes Beispielsystem

4 Grundlegende Architektur von AutomationML

Das AutomationML Datenaustauschformat wurde und wird durch den AutomationML e.V. entwickelt (siehe (AutomationML e.V.)): Es stellt eine Lösung für den konsistenten und verlustfreien Datenaustausch im Fabrikplanungs- bzw. Produktionssystemplanungsprozess für den Bereich der Automatisierungstechnik und darüber hinaus dar. Es ist ein offenes, neutrales, XML-basiertes und freies Datenaustauschformat, welches einen domänen- und unternehmensübergreifenden Transfer von Entwurfsdaten im Rahmen des Entwurfsprozesses von Produktionssystemen in einer heterogenen Werkzeuglandschaften ermöglicht.

AutomationML folgt bei der Modellierung von Entwurfsinformationen einem objektorientierten Ansatz und ermöglicht die Beschreibung von physischen und logischen Anlagenkomponenten als Datenobjekte, die unterschiedliche Aspekte zusammenfassen. Objekte können dabei eine Hierarchie bilden, d. h. ein Objekt kann aus anderen Unterobjekten bestehen und kann selbst ein Teil einer größeren Objekts sein. Typische Objekte in der Fabrikautomatisierung enthalten Information über die Struktur (Topologie), Geometrie und Kinematik sowie das Verhalten.

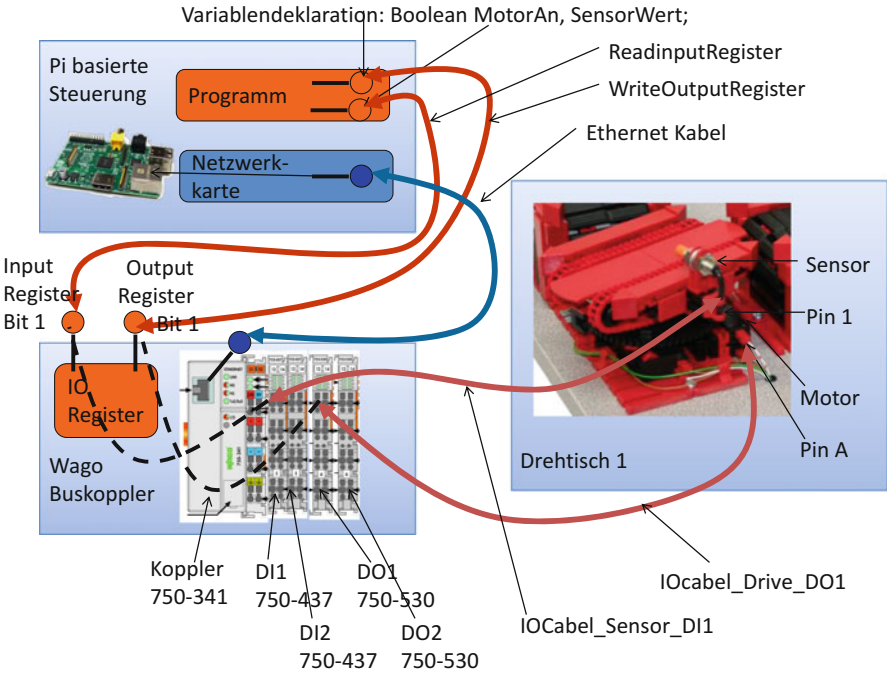

Abb. 7 Modellierter Ausschnitt des Beispielsystems

Dabei folgt AutomationML einem modularen Aufbau und integriert und adaptiert verschiedene bereits existierende XML-basierte Datenformate unter einem Dach, dem sogenannten Dachformat (siehe Abb. 8). Diese Datenformate werden „as-is“ genutzt und sind nicht für AutomationML Anforderungen verzweigt worden. Jedoch definiert AutomationML Anwendungsregeln.

Logisch unterteilt sich AutomationML in:

- Beschreibung der Anlagestruktur und der Kommunikationssysteme, die in einer Hierarchie aus AutomationML Objekten dargestellt und mithilfe von CAEX nach IEC 62424 beschrieben werden (International Electrotechnical Commission 2008),
- Beschreibung der Geometrie und der Kinematik von unterschiedlichen AutomationML Objekten, die mithilfe von COLLADA (ISO/PAS 17506:2012) dargestellt werden (International Organization for Standardization 2012),
- Beschreibung der Steuerung in Bezug auf logische Daten von verschiedenen AutomationML Objekten, die mithilfe von PLCopen XML dargestellt werden (PLCopen association 2012) und
- Beschreibung der Beziehungen zwischen den AutomationML Objekten und Verweisen zu Informationen, die in Dokumenten außerhalb des Dachformats gespeichert sind.

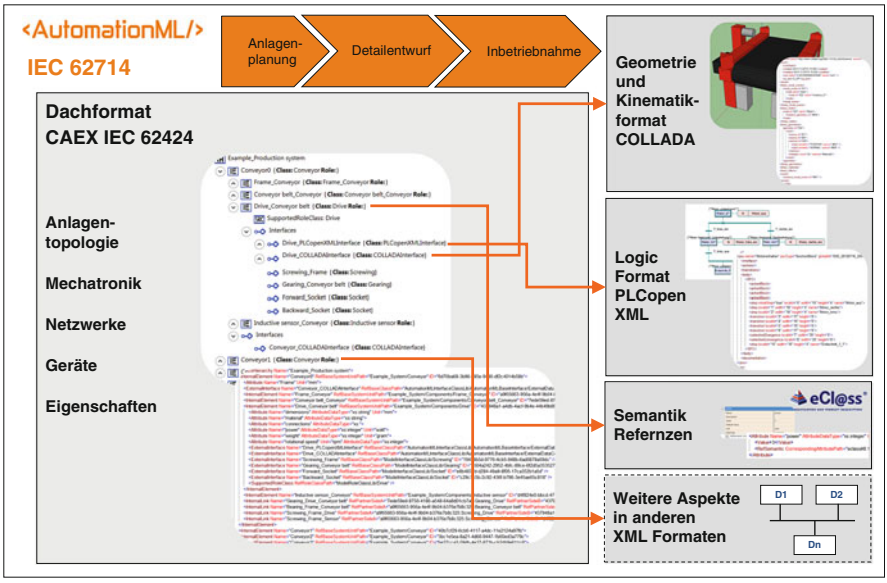

Abb. 8 Struktur eines AutomationML Projektes

AutomationML wird derzeit in der IEC 62714 Normenreihe standardisiert (International Electrotechnical Commission 2014). Für weitere Informationen zu AutomationML sei an dieser Stelle auf (Drath 2010) und (AutomationML e.V.) verwiesen.

Das Fundament von AutomationML ist der Einsatz von CAEX als Dachformat und die Definition eines geeigneten CAEX Profils, das alle relevanten Anforderungen von AutomationML zur Modellierung von Entwurfsinformationen eines Produktionssystems, zur Integration der drei genannten Datenformate CAEX, COLLADA und PLCopen XML und zur Erweiterung, falls zukünftig erforderlich, erfüllt.

CAEX ermöglicht die oben beschriebene, objektorientierte Vorgehensweise (siehe Abb. 9). Es erlaubt die Festlegung von Semantiken der Objekte unter Nutzung von Rollenklassen, die in RoleClassLibraries (Rollenklassenbibliotheken) definiert und erfasst sind. Schnittstellen (Interfaces) zwischen Objekten werden unter Nutzung von Interfaceklassen, die in InterfaceClassLibraries (Interfaceklassenbibliotheken) definiert und erfasst sind, spezifiziert. Auf Basis beider Typen von Klassen (und der entsprechenden Bibliotheken) werden Klassen von Objekten unter Nutzung von SystemUnitKlassen (SUC), die in SystemUnitClassLibraries (SystemUnitKlassenbibliotheken) definiert und erfasst sind, modelliert. Letztlich dienen alle modellierten Klassen der semantisch eindeutigen Modellierung der eigentlichen Projektinformationen, d. h. der einzelnen Objekte des modellierten Produktionssystems, in einer (oder mehreren) InstanceHierarchies (IH) als eine Hierarchie von InternalElements (IE) unter Bezugnahme auf sowohl SystemUnitKlassen, von denen sie abgeleitet

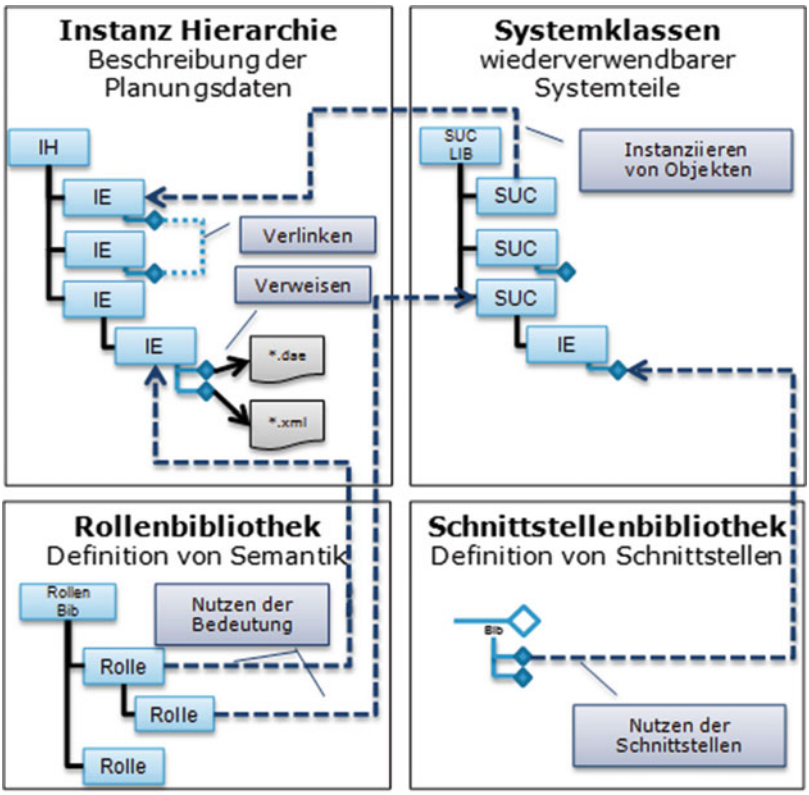

Abb. 9 AutomationML Topologiebeschreibung

werden, als auch Rollenklassen, die ihre Semantiken bestimmen. Diese können zur Verknüpfung untereinander oder mit extern modellierten Informationen (bspw. COLLADA oder PLCopen XML Dateien) Interfaces enthalten. Die spezifischen Eigenschaften von Objekten, SystemUnitKlassen, Rollen und Interfaces werden über Attribute abgebildet. Für weitere Details sei noch einmal auf die AutomationML Whitepaper in (Lindemann 2007) verwiesen.

Weitere wichtige Eigenschaften des AutomationML Datenaustauschformates sind: die Trennung von Syntax und Semantik für Datenobjekte auf Basis der Bibliotheken von Rollen und SystemUnitKlassen und ihres Referenzierens aus der InstanceHierarchy heraus, die Bereitstellung von eindeutigen Identifikationsmöglichkeiten für alle Datenobjekte über Universally Unique Identifiers (UUID) und die Bereitstellung von Versions- und Revisionsinformationen für jedes Datenobjekt durch entsprechende Attribute.

5 Modellierung der Systemtopologie und der Systemelemente

Wie oben beschrieben verwendet AutomationML CAEX als Dachformat für die Abbildung der Topologie und der Elemente eines Produktionssystems. Dafür stellt AutomationML die nachfolgenden Modellierungsmittel zur Verfügung.

Das erste Modellierungsmittel bilden die Rollenklassen (role classes) die in entsprechenden Rollenklassenbibliotheken (role class libraries) gesammelt werden. Eine Rollenklasse beschreibt eine abstrakte Funktion eines Systemelementes ohne dabei eine technische Realisierung dieser Funktion festzulegen. Entsprechend kann sie als Indikator für eine spezielle Semantik eines Systemelementes angesehen werden. Beispiel für die Rollenklassen sind die Klassen *MechanicalPart* und *Device*, die die Semantik von Strukturelementen tragen, oder die Klassen *LogicalDevice* und *PhysicalDevice*, die eine Semantik für Kommunikationssystemelemente beinhalten.

AutomationML definiert eine Menge von Basisrollen, die in Abb. 10 dargestellt sind. Zum einen werden im Teil 1 und 2 des AutomationML Standards (International Electrotechnical Commission 2014) mit der *AutomationMLBaseRoleClassLib* und weiteren Bibliotheken grundlegende Basisrollen definiert. Zum anderen kommen in den weiteren Teilen des AutomationML Standards weitere Rollen hinzu. Ein Beispiel ist hier die *CommunicationRoleClassLib*, die im Communication Whitepaper spezifiziert wurde (AutomationML e.V.).

Jeder AutomationML Anwender kann neue Rollenklassen passend für seine Anwendungsfälle des Datenaustausches selbst definieren. Dabei legt AutomationML einige Regeln für die Definition von neuen Rollenklassen fest.

Jede Rollenklasse muss einen, innerhalb des Rollenbaumes/Rollenpfades der Rollenbibliothek eindeutigen Namen besitzen. Damit kann sie eindeutig über den hierarchischen Namenspfad referenziert werden. Die Rollenklasse *Port* in Abb. 11 hat den Identifikationspfad *AutomationMLBaseRoleClassLib/AutomationMLBaseRole/Port.* Zusätzlich zur eindeutigen Benennung muss jede Rollenklasse direkt oder indirekt von der Basisrollenklasse *AutomationMLBaseRole* unter Nutzung des *RefBaseClassPath* Attributes abgeleitet werden.

Jede Rolle kann Attribute und Interfaces enthalten. Diese Attribute und Interfaces sollen es dem Importer eines Entwurfswerkzeuges ermöglichen, die eingelesenen Informationen korrekt zu interpretieren und zu verarbeiten.

Ein Beispiel einer nutzerdefinierten Rollenklasse ist die Klasse *ModbusTCPPhysicalDevice* mit den Attributen *MACaddress* und *IPAddress,* die ein Automatisierungsgerät beschreiben würde, das mittels ModbusTCP kommunizieren kann.

Das zweite Modellierungsmittel bilden die Interfaceklassen. Eine Interfaceklasse beschreibt den Endpunkt einer abstrakten Relation zwischen den Systemelementen oder die Referenz auf Informationen, die außerhalb der CAEX Modellierung festgehalten werden (siehe Geometrie und Kinematik bzw. Verhaltensmodellierung). Beispiele für Interfaceklassen sind die Interfaceklassen *SignalInterface* und *PhysicalEndPoint,* die abstrakte Schnittstellen für den Anschluss von Kabeln für die Signalverarbeitung beschreiben oder die Interfaceklasse *ExternalDataConnector* zur Beschreibung des Zugangspunktes zu einer extern gespeicherten Information.

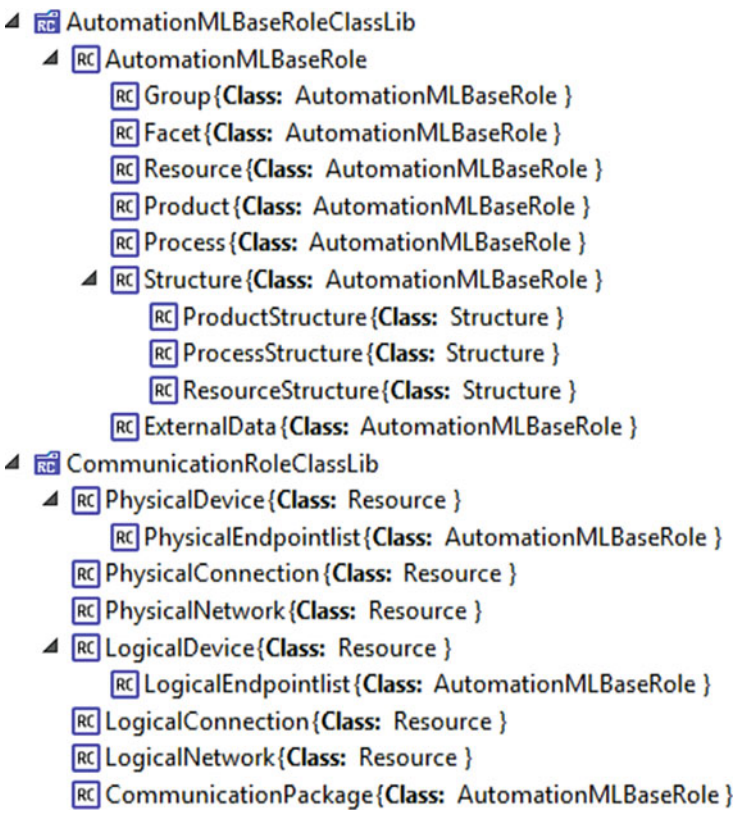

Abb. 10 AutomationMLBaseRoleClassLib und CommunicationRoleClassLib

AutomationML definiert eine Menge von Basisinterfaceklassen wie in Abb. 12 dargestellt. Zum einen werden im Teil 1 des AutomationML Standards (International Electrotechnical Commission 2014) mit der *AutomationMLInterfaceClassLib* grundlegende Basisinterfaces definiert. Zum anderen kommen in den weiteren Teilen des AutomationML Standards weitere Interfacebibliotheken hinzu. Ein Beispiel ist hier die *CommunicationInterfaceClassLib*, die im Communication Whitepaper spezifiziert wurde (AutomationML e.V.).

Jeder AutomationML Nutzer kann neue Interfaceklassen passend zu seinem Anwendungsfall des Datenaustauschs selbst definieren. AutomationML legt dazu einen Satz von Basisregeln fest.

Jede Interfaceklasse muss einen, innerhalb des Interfaceklassenbaumes der Interfaceklassenbibliothek eindeutigen Namen besitzen. Damit kann sie eindeutig über den hierarchischen Namenspfad referenziert werden. Die Interfaceklasse *COLLADA-Interface* in Abb. 13 hat den Identifikationspfad *AutomationMLInterfaceClassLib/AutomationMLBaseInterface/ExternalData-Connector/COLLADAInterface.* Zusätzlich zur eindeutigen Benennung muss jede Interfaceklasse direkt oder indirekt von

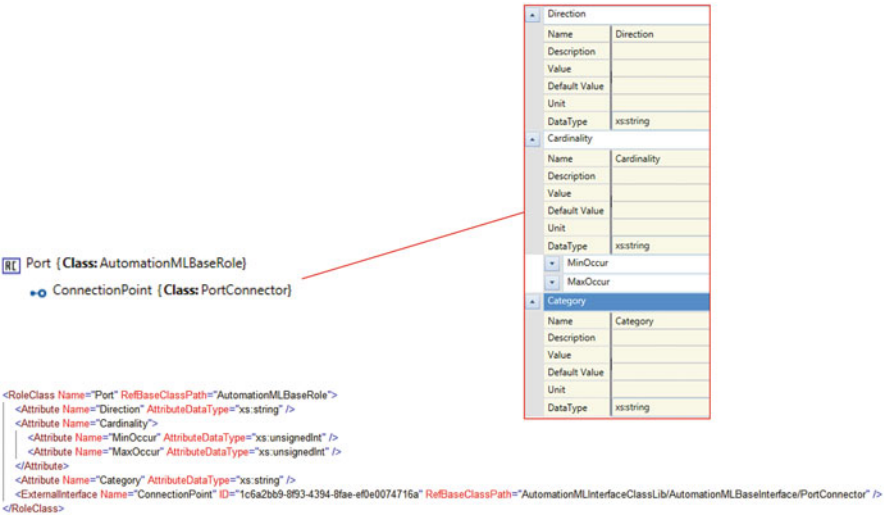

Abb. 11 *Port* Rollenklasse als Beispiel für eine Rollendefinition

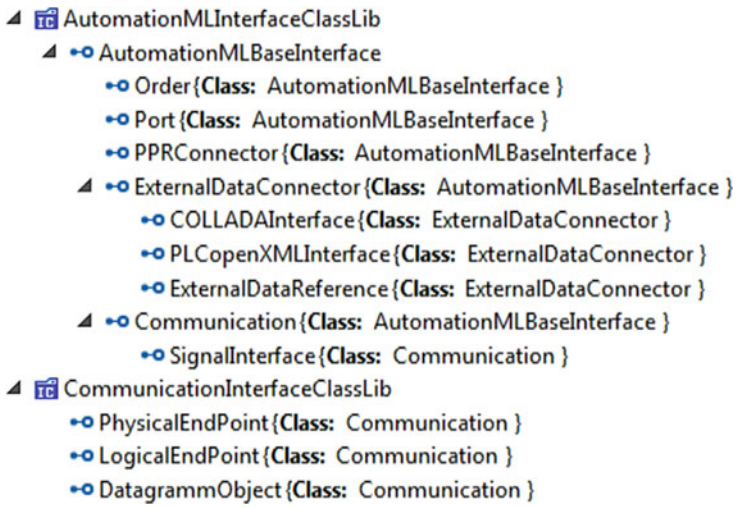

Abb. 12 AutomationMLBaseInterfaceClassLib und CommunicationInterfaceClassLib

der Basisinterfaceklasse *AutomationMLBaseInterface* unter Nutzung des *RefBaseClassPath* Attributes abgeleitet werden.

Jedes Interface kann Attribute enthalten. Diese Attribute sollen in jeder Instanz der Rollenklasse mit Werten gefüllt werden.

Ein Beispiel für eine nutzerdefinierte Interfaceklasse ist die Interfaceklasse *ModbusTCPSocket,* die im Anwendungsbeispiel definiert ist. Sie beschreibt die Schnitt-

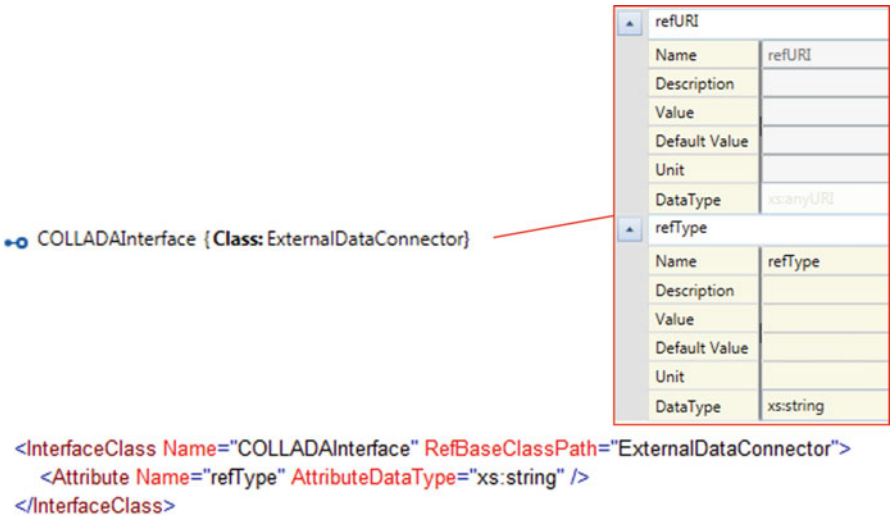

Abb. 13 COLLADAInterfaceClass als Beispiel für eine Interfacedefinition

stelle, an der ein Ethernetkabel mit einem Kommunikationsgerät verbunden werden kann, das über ModbusTCP kommuniziert.

Das dritte Modellierungsmittel sind die Systemunitklassen. Systemunitklassen können als wiederverwendbare Systemkomponenten bzw. Systemelemente verstanden werden, die wie ein Template in der Modellierung Verwendung finden. Üblicherweise entsprechen sie entweder einem Herstellerkatalog für Geräte oder Bauelemente oder einer Menge von Templates eines spezifischen Entwurfswerkzeuges zur Strukturierung der disziplinspezifischen Informationen.

Im AutomationML Standard werden keine grundlegenden Systemunitklassenbibliotheken definiert. Entsprechend ist die Definition von Systemunitklassen Aufgabe des AutomationML Nutzers. Auch hier definiert AutomationML Regeln für die Definition von Systemunitklassen.

In Analogie zu den Rollenklassen und Interfaceklassen sollen Systemuniklassen einen eindeutigen Namen besitzen. Ihnen muss mindestens eine Rolle über das Subelement SupportedRoleClass zugeordnet werden, um der Systemunitklasse eine eindeutige Semantik zuzuordnen. Jede Systemunitklasse kann eine Hierarchie von Subobjekten vom Typ InternalElement, Attribute sowie Instanzen von Interfaceklassen enthalten, die gemeinsam die Struktur und die Eigenschaften der modellierten Klasse von Systemelementen bilden.

Zusätzlich kann jede Systemunitklasse über das *RefBaseClassPath* Attribut von einer anderen Systemunitklasse abgeleitet werden. In diesem Fall erbt sie alle zugeordneten Rolleninstanzen, Interfaceinstanzen und Attribute vom Elternelement.

Ein Beispiel für eine nutzerdefinierte Systemunitklassenbibliothek ist in Abb. 14 gegeben. Abb. 15 enthält dann das Beispiel der nutzerdefinierten Systemunitklasse *Motor*, die einen elektrischen Antrieb repräsentiert.

Abb. 14 Beispiel einer Systemunitklassenbibliothek

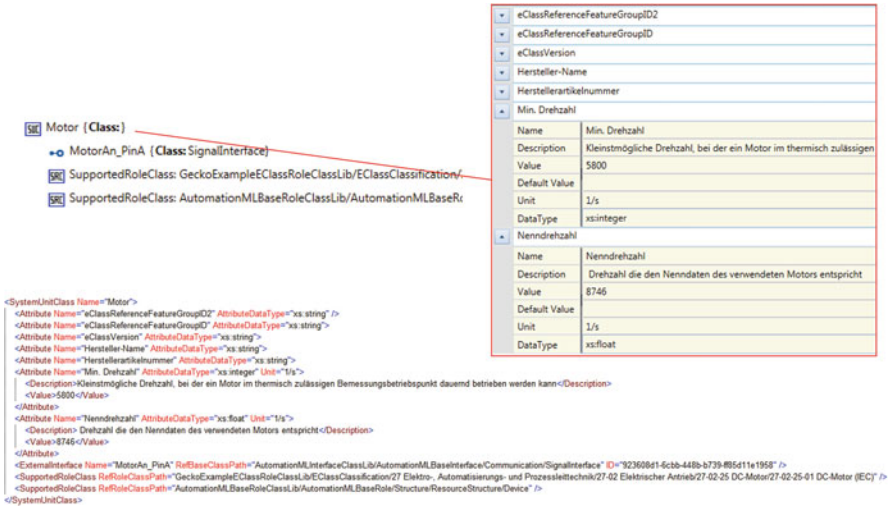

Abb. 15 *Motor* Systemunitklasse als Beispiel für eine nutzerdefinierte Systemunitklasse

Alle beschriebenen Modellierungskonzepte können Attribute besitzen. Attribute können als Eigenschaftsbeschreibungen aufgefasst werden, die den einzelnen Rollenklassen, Interfaceklassen und Systemunitklassen zugeordnet sind.

AutomationML legt Regeln für die Definition von Attributen fest. Jedes Attribut soll innerhalb seines Elternelementes einen eindeutigen Namen besitzen. Es darf ein *DataType* und ein *Unit* Attribut sowie Subelemente für Beschreibungen, Standardwerte, den aktuellen Wert und eine Semantikreferenz besitzen. Ein Beispiel für ein nutzerdefiniertes Attribut ist in Abb. 16 dargestellt.

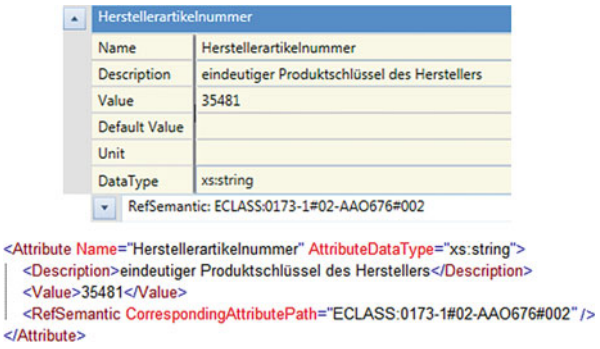

Abb. 16 *Herstellerartikelnummer* als Beispiel für ein nutzerdefiniertes Attribut

Das wichtigste Modellierungsmittel der AutomationML ist jedoch die InstanceHierarchy mit den in ihr hierarchisch strukturierenden *InternalElements*. Sie repräsentiert die aktuellen auszutauschenden Entwurfsdaten, die in CAEX modelliert werden sollen.

Das Arbeitspferd der entsprechenden Modellierung ist das *InternalElement*. Es repräsentiert eine Objektinstanz im Produktionssystem, die abgebildet werden soll. Je nach betrachtetem Abstraktionsniveau kann es physische Produktionssystemkomponenten wie die gesamte Anlage, die funktionalen Einheiten wie Maschinen und Drehtische, Geräte wie Motoren oder Steuerungen oder mechanische Komponenten wie Bänder und Kabel repräsentieren. Es kann auch logische Komponenten wie SPS Programme, Produktbeschreibungen oder Aufträge abbilden.

InternalElements in der InstanceHierarchy sind generell nutzerdefiniert. Sie können Attribute und Instanzen von Interfaceklassen enthalten, die aus beliebigen Interfaceklassenbibliotheken stammen können. Sie können eine Referenz auf eine Systemunitklasse, die einer beliebigen Systemunitklassenbibliothek entstammt, im *RefBaseSystemUnitPath* Attribut enthalten. Diese Referenz ermöglicht die Identifikation der Systemunitklasse, von der das InternalElement instanziiert wurde. Dies bedeutet auch, dass das InternalElement dieselbe Substruktur, dieselben Attribute und dieselben Interfaces wie die Systemunitklasse, von der es instanziiert wurde, enthalten sollte. Zudem sollte jedes InternalElement mindestens eine Rollenklasse aus einen beliebigen Rollenklassenbibliothek referenzieren. Dafür werden die *RoleRequirements* und *SupportedRoleClass* Subobjekte verwendet werden. Die referenzierten Rollen sollen die Semantiken der einzelnen InternalElements definierten. Die Struktur eines *InternalElements* ist in Abb. 17 enthalten.

Ein Beispiel einer *InstanceHierarchy*, dass die Beispielanlage modelliert, ist in Abb. 18 dargestellt. Hier ist eine Entitätenhierarchie aus logischen und physischen Objekten modelliert, die von der obersten Ebene von *InternalElements* dem *FlexibleManufacturingSystem* über InternalElements, die Drehtische (*Drehtisch1*), Buskoppler (*WagoIOA*) und Steuerungen (*PIBasedControllerA*) repräsentieren, bis hinunter zu *InternalElements* für Steuerungsprogramme (*MyPIProgram*), Automatisierungsgeräte (*myMotor*) und Kabel *IOKabel_Motor_DO1_DrathB*) reicht.

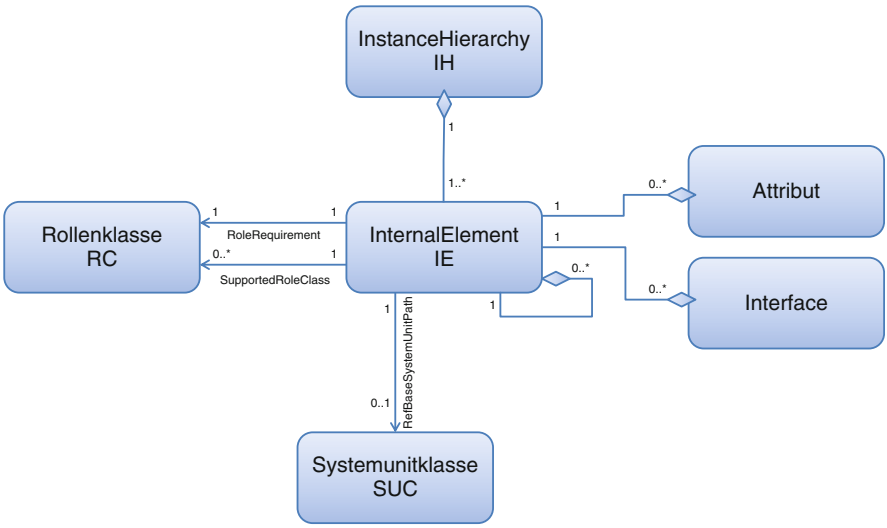

Abb. 17 Strukturmodell des InternalElements

Ein Beispiel eines spezifischen *InternalElements* ist in Abb. 19 gezeigt. Es beschreibt ein Kabel, dass ein Steuerungsgerät mit einem Buskoppler verbindet. Dieses *InternalElement* besitzt eine Vielzahl von Attributen wie die *Polzahl*, die die Anzahl der Adern im Kabel beschreibt, und die *min. zulässige Kabelaußentemperatur*, die die minimal zulässige Temperatur der Kabeloberfläche benennt. Zudem besitzt es zwei Interfaceinstanzen, die die beiden Kabelenden repräsentieren.

6 Integration von Objektsemantik

Ein kritischer Punkt beim Import von Entwurfsdaten in ein Entwurfswerkzeug ist die Abbildung der importieren Informationen auf des interne Datenmodell des importierenden Werkzeugs. In diesem Moment muss entschieden werden, welche Bedeutung ein gelesenes Datum innerhalb des internen Datenmodells besitzt und wie es entsprechend zu behandeln ist.

Beim Datenaustauschprozess mit AutomationML werden die eigentlichen auszutauschenden Daten in der *InstanceHierarchy* als *InternalElements* mit Attributen übertragen. Um die Semantik eines *InternalElements* in AutomationML zu modellieren, bestehen zwei Möglichkeiten: das Referenzieren von *Rollenklassen* und das Referenzieren von *Systemunitklassen*. Für das Referenzieren von Rollenklassen können die Subobjekte *RoleRequirements* und *SupportedRoleClass* verwendet werden. Für die Repräsentation der Semantik eines Attributes kann das Attribut *RefSemantic* verwendet werden, das es für jedes AutomationML Attribut gibt. Abb. 20 stellt diese Mittel zur Semantikabbildung noch einmal grafisch dar.

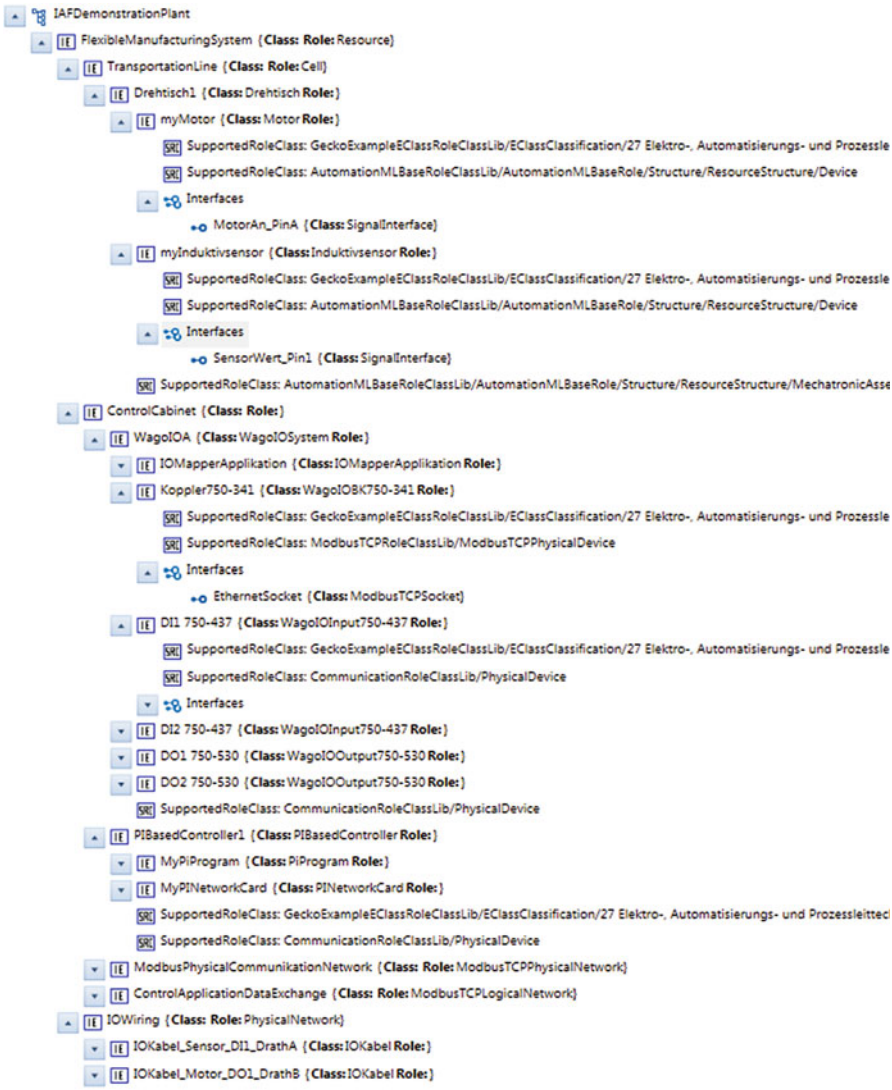

Abb. 18 Beispiel einer *InstanceHierarchy* für das Anlagenbeispiel (Ausschnitt)

AutomationML wird nicht in jedem Fall die Semantik von Objekten, die in Produktionssystemen relevant sind, selbst definieren. Dies ist auch gar nicht notwendig, da bereits eine Vielzahl von Klassifikationsstandards für Produktionssystemkomponenten und -geräte wie zum Beispiel eCl@ss (eCl@ss association) existieren.

eCl@ss ist eine hierarchisch strukturierte Semantikklassifikation für Materialien, Geräte, Produkte und Services mit einer logischen Strukturierung, die einen Detail-

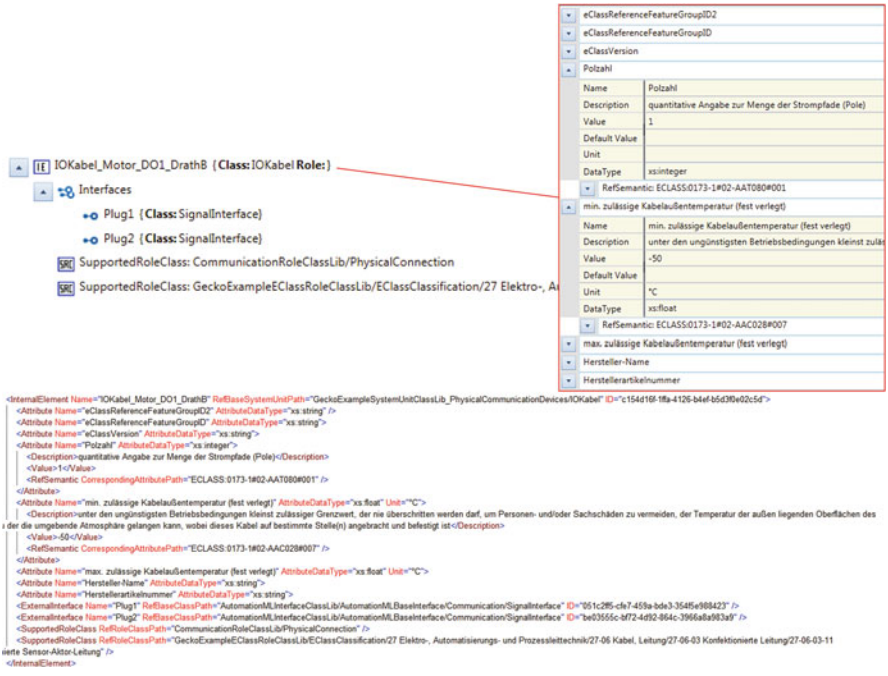

Abb. 19 *IOKabel_Motor_DO1_DrathB* als Beispiel für ein *InternalElement*

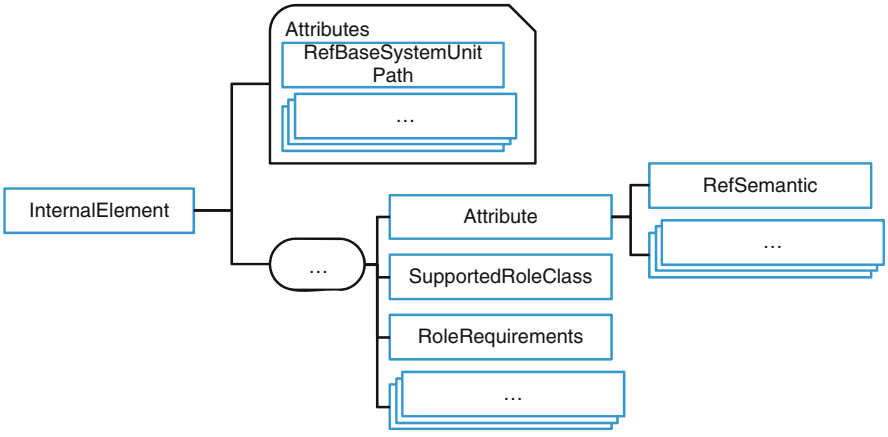

Abb. 20 Möglichkeiten der Semantikintegration in *InternalElements* und Attribute

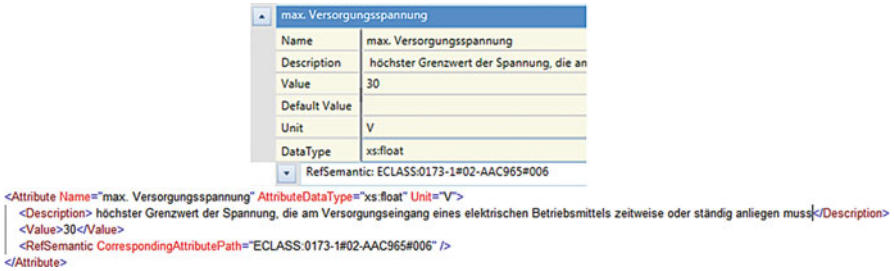

Abb. 21 Beispiel der Semantikbeschreibung für ein Attribut

lierungsgrad erreicht, der den produktspezifischen Eigenschaften von Produktionssystemkomponenten genügt. Für jede Klasse von Geräten und Materialien werden standardisierte beschreibende Eigenschaften festgelegt, die für die detaillierte Charakterisierung und Auswahl von Geräten und Materialien genutzt werden kann.

Schlüsselelement der eCl@ss Spezifikation ist die IRDI (International Registration Data Identifier), die auf den internationalen Standards ISO/IEC 11179-6, ISO 29002, und ISO 6532 basiert. Sie stellt eine Möglichkeit zur eineindeutigen Identifikation jedes einzelnen Attributes und jeder einzelnen Klassifikationsklasse bereit.

Um die Semantik von Attributen aus AutomationML heraus referenzieren zu können, wird die IRDI von eCl@ss Attributen verwendet. Für jedes Attribut kann das Attribut *CorrespondingAttributePath* des CAEX Schemaelementes *RefSemantic* in jedem Attribut mit einem String der Form *ECLASS: + IRDI* belegt werden, wobei die IRDI der IRDI des zu referenzierenden eCl@ss Attributs entsprechen soll.

Abb. 21 stellt ein Beispiel für diese Nutzung der *RefSemantic* dar. Hier ist das Attribut *max. Versorgungsspannung* enthalten, das die maximal zulässige Versorgungsspannung für einen induktiven Sensors angibt. Die Semantik dieses Attributs wird durch die IRDI *0173-1#02-AAC962#006* eindeutig festgelegt.

Die Semantikbeschreibung von *InternalElements* ist etwas komplexer. Hier kommt das Rollenkonzept zur Anwendung. Die Klassifikation des Klassifikationskataloges, der verwendet werden soll (in diesem Falle eCl@ss), muss in eine nutzerdefinierte AutomationML Rollenbibliothek übersetzt werden. Dabei sollte zum einen die hierarchische Struktur des Klassifikationsstandards beibehalten und zum anderen für jede Klassifikationsklasse eine entsprechende Rolle mit allen klassenspezifischen Attributen erstellt werden. Bei der Rollendefinition sollten drei weitere klassifikationsspezifische Attribute angelegt werden, die den Klassifikationsstandard in seiner Version, die Identifikation der Klasse und die Klassen-IRDI beinhalten sollen. Ein Beispiel für eine derartige nutzerdefinierte Rollenklassenbibliothek für das Anwendungsbeispiel ist in Abb. 22 gegeben.

Die erstellten Rollenklassen werden dann aus den *InternalElements* heraus unter Nutzung der *RoleRequirements* und *SupportedRoleClass* Subobjekte referenziert. Ein Beispiel dieser Referenzierung zeigt Abb. 23. Hier wird das *InternalElement myMotor* als IEC DC Drive mit der Klassifikationsklasse *27-02-25-01* nach eCl@ss identifiziert.

Abb. 22 Beispiel einer Rollenklassenbibliothek zur Semantikbeschreibung für *InternalElements*

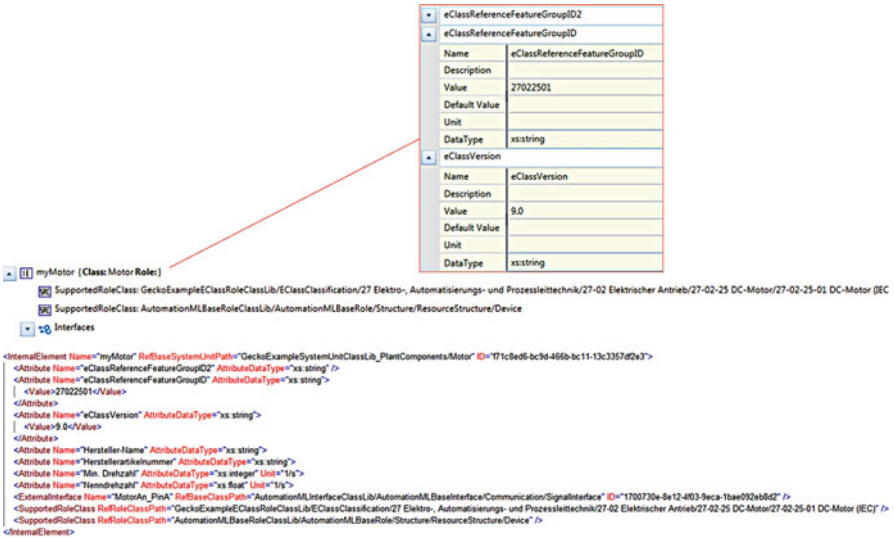

Abb. 23 Beispiel der Semantikbeschreibung für ein *InternalElement*

7 Geometrie und Kinematik

Wie bereits oben beschrieben, nutzt AutomationML für die Beschreibung von Geometrie- und Kinematikinformationen den internationalen Standard COLLADA 1.4.1 und 1.5.0, der innerhalb der ISO/PAS 17506:2012 standardisiert ist (International Organization for Standardization 2012).

Zum ersten werden relevante Geometrien und Kinematiken mittels COLLADA modelliert und als zweites aus einem CAEX Datenmodell der Topologie heraus referenziert.

COLLADA steht für COLLAborative Design Activity. Es wurde innerhalb der KHRONOS Association unter der Führung von Sony als ein Austauschformat für das Feld der Digital Content Creation der Spieleindustrie entwickelt. Es ermöglicht die Darstellung von 3D Objekten innerhalb von 3D Szenen einschließlich der relevanten visuellen, kinematischen und dynamischen Eigenschaften für eine Objektanimation.

COLLADA (Arnaud und Barnes 2006) ist ein XML basiertes Datenformat mit einer modularen Struktur, die die Definition von Bibliotheken für Geometrien, Materialien, Beleuchtungen, Kameras, visuellen Szenen, kinematischen Modellen, kinematischen Szenen und anderem ermöglicht. Ein Beispiel einer COLLADA Datei ist in Abb. 24 dargestellt. Das linke obere Bild zeigt das originale Band in der Beispielanlage. Zu diesem Beispiel ist das entsprechende Modell links unten und die COLLADA Datei in Ausschnitten rechts dargestellt.

Das wichtigste Modellierungsmittel für die Integration von COLLADA Dateien in AutomationML Projekten ist die eindeutige Identifizierbarkeit von Modellierungsobjekten in COLLADA. Alle für AutomationML wichtigen Modellierungselemente in COLLADA (wie Geometrien, visuelle Szenen oder kinematische Szenen) besitzen eine eindeutige Identifikationsnummer (ID).

Für die Referenzierung dieser identifizierbaren Objekte spezifiziert AutomationML die spezielle Interfaceklasse *COLLADAInterface* in der *AutomationMLInterfaceClassLib*. Diese Interfaceklasse ist (wie in Abb. 25 gezeigt) von der Interfaceklasse *ExternalDataConnector* abgeleitet und beinhaltet daher das Attribut *refURI.* Dieses Attribut wird dazu verwendet, auf IDs innerhalb einer COLLADA Datei zu referenzieren. Dazu soll der Wert dieses Attributes einen String der Form *file:/// filename.dae#ID* enthalten. Das Attribut *refType* dient der Spezifikation der Art, wie ein Objekt in einer Szene eingebettet ist. Damit kann ein physikalischer Zusammenhang von Objekten modelliert werden. So sollte sich zum Beispiel ein Werkstück mit dem Transportband bewegen, wenn es auf diesem liegt und das Band sich bewegt.

Ein Beispiel für die Integration einer Geometrie in ein AutomationML Projekt ist in Abb. 26 zu sehen. Es zeigt, wie die Systemunitklasse zu einem Motor um eine entsprechende Geometrie erweitert werden kann.

Natürlich wird eine *InstanceHierarchy* mehr als ein *InternalElement* enthalten, das eine Geometriebeschreibung besitzt. Um die korrekte Positionierung des entsprechenden Geometrieobjektes in einer Szene zu ermöglichen, definiert AutomationML ein spezielles Attribut für *InternalElements*, das zur Positionsbestimmung dieses Elements bezogen auf die Koordinaten des Eltern-*InternalElements* genutzt

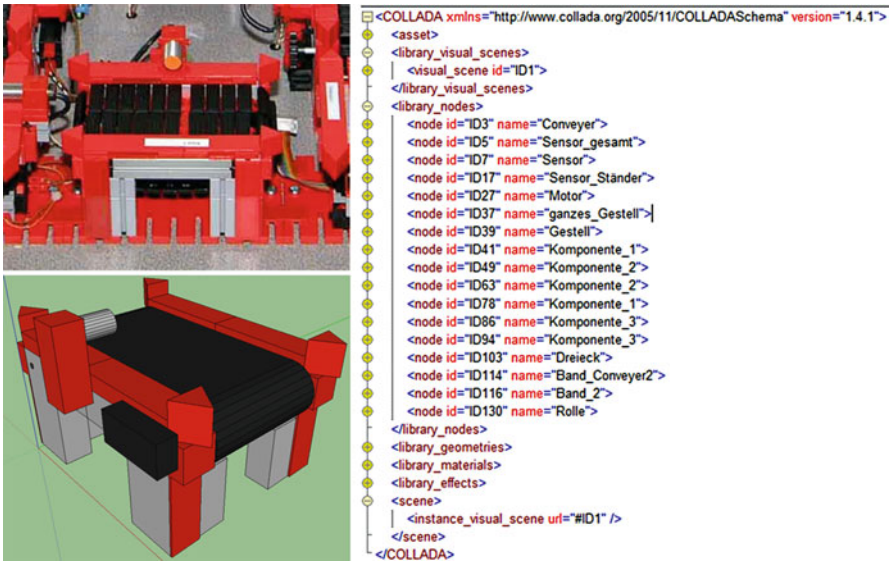

Abb. 24 Beispiel einer COLLADA Datei

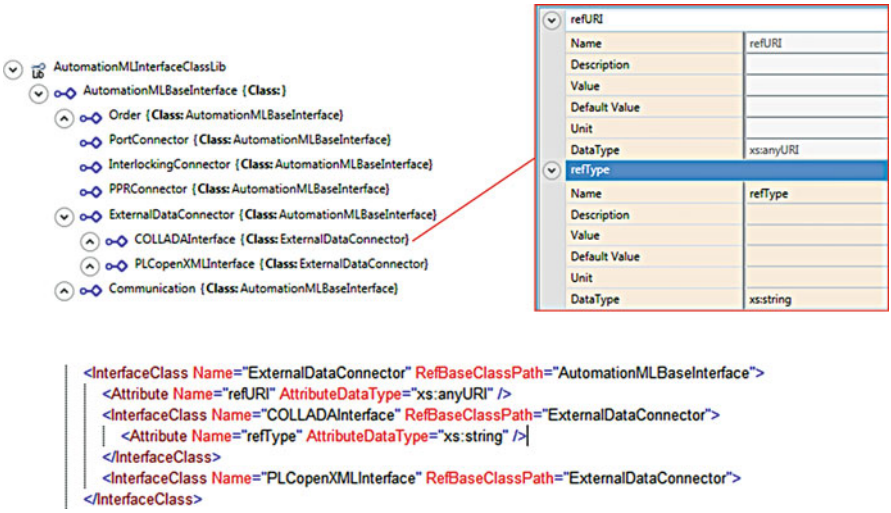

Abb. 25 Definition der *COLLADAInterface* Interfaceklasse

werden kann, das *Frame* Attribut. Dieses Attribut ist in Abb. 27 dargestellt und enthält den Verschiebungs- und Verdrehungsvektor für das betrachtete Objekt entlang der drei kartesischen Achsen x, y und z sowie den Rotationen um diese Achsen.

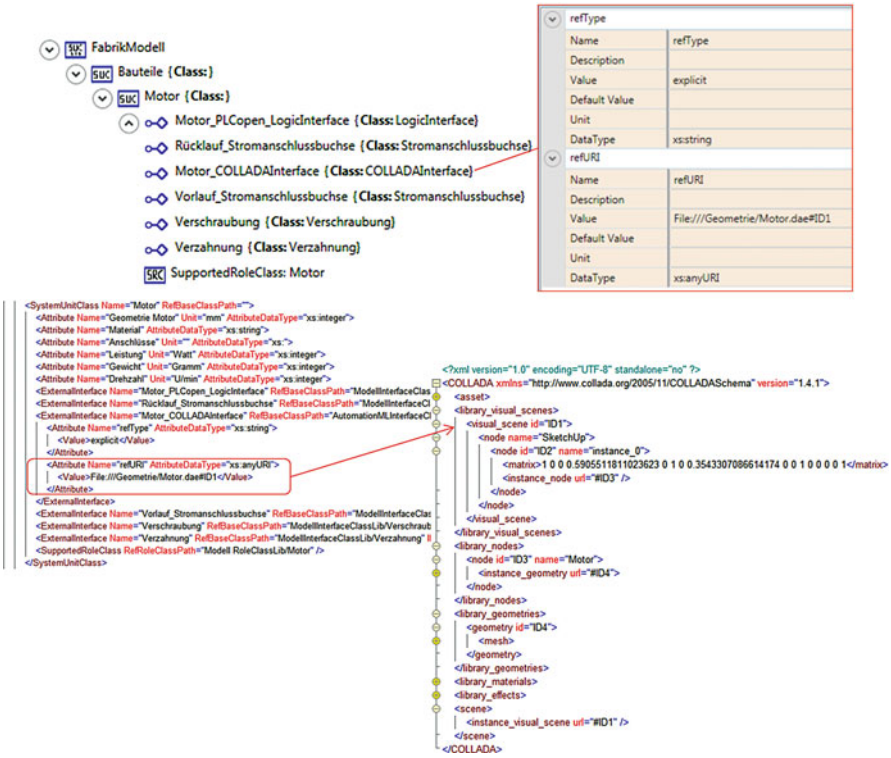

Abb. 26 Beispiel der Geometrieintegration

8 Verhaltensmodellierung

Ähnlich der Modellierung von Geometrie- und Kinematikinformationen nutzt AutomationML zur Modellierung von Verhalten ein weiteres XML basiertes Datenaustauschformat, PLCopen XML (PLCopen association 2012), was durch die PLCopen Assoziation entwickelt und letztendlich in der IEC 61131 als Teil 10 standardisiert wurde (International Electrotechnical Commission 2019). Der AutomationML e.V. hat zur Verhaltensmodellierung und zur Integration in CAEX einen zweistufigen Prozess erarbeitet. Als Erstes wird das relevante Verhalten modelliert und in einer PLCopen XML Datei abgelegt. Als Zweites wird diese Datei bzw. deren Inhalt aus der CAEX Datei heraus referenziert.

PLCopen ist eine hersteller- und produktunabhängige, weltweite Assoziation, die sich zum Ziel gesetzt hat, das Thema der Steuerungsprogrammierung durch internationale Standards zu unterstützen. Speziell fördern sie die Anwendung der IEC 61131-3. Mit PLCopen XML hat die PLCopen Assoziation ein neutrales, offenes, XML basiertes Datenformat geschaffen, was den Austausch von Steuerungsprogrammen zwischen verschiedenen Softwareumgebungen ermöglicht. Automati-

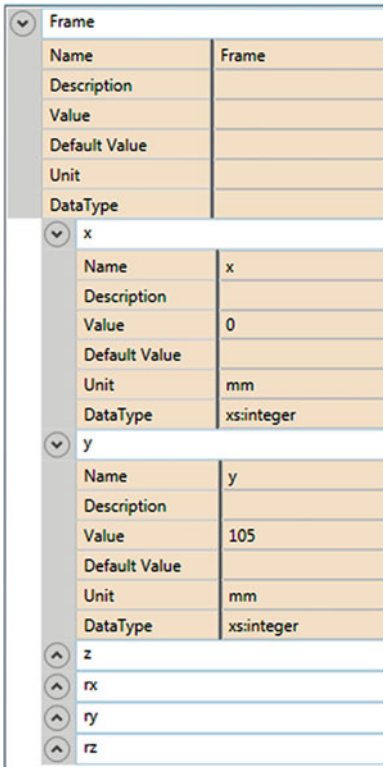

Abb. 27 Beispiel eines Frame Attributs

onML nutzt PLCopen XML Version 2.0, 2.0.1 und IEC 61131-10. Die Vorletztere ist aktuell die meist genutzte der IEC 61131-3 Edition 2.

Eine PLCopen XML Datei ist so strukturiert, dass sie alle essenziellen Teile eines IEC 61131-3 Steuerungsprogrammierungsprojektes fasst. Sie beinhaltet Softwarewerkzeuginformationen, den eigentlichen Steuerungscode (die Programmierungsstruktur beinbehaltend) sowie SPS-Hardwareinformationen. Wichtigster Bestandteil für AutomationML sind die Programmorganisationseinheiten (POEs), siehe Abb. 28. Jede POE beschreibt eine strukturelle Einheit eines SPS-Programms, die den Code in einer der fünf IEC 61131-3 Programmiersprachen enthält sowie die Variablendeklaration. Jede POE und jedes Objekt in der POE kann einen globalen Identifier haben, wodurch eine eindeutige Referenzierung der Objekte möglich ist.

Da AutomationML anstrebt, den gesamten Entwurfsprozess von Produktionssystemen abzudecken bzw. dessen Daten austauschbar zu machen, müssen auch die Verhaltensbeschreibungen auf unterschiedlichen Abstraktionsebenen bzw. Detaillierungsleveln betrachtet werden. Wie in Abb. 29 zu sehen, werden Gantt und PERT Charts zur groben Planung von Abläufen am Anfang des Entwurfsprozesses eingesetzt. Mit Fortschreiten im Prozess werden die Modelle immer konkreter. Es kommen Impulsdiagramme oder Logiknetzwerke zum Einsatz. Am Ende stehen dann der

```
<?xml version="1.0" encoding="utf-8"?>
<project xmlns="http://www.plcopen.org/xml/tc6_0200">
  <fileHeader companyName="" productName="CODESYS" productVersion="CODESYS V3.5 SP3 Patch 7" creationDateTime="2013-12-18T19:32:28.4147223" />
  <contentHeader name="TEST.project" modificationDateTime="2013-12-18T18:56:49.0513577">
  <types>
    <dataTypes />
    <pous>
      <pou name="Motor" pouType="functionBlock" globalId="ISID_20131218-500">
        <interface>
          <localVars>
            <variable name="Signal" globalId="ISID_20131218-501">
            <variable name="AUS" globalId="ISID_20131218-502">
          </localVars>
        </interface>
        <body>
          <SFC>
            <step localId="0" initialStep="true" name="Motor AUS" globalId="ISID_20131218-503">
            <selectionDivergence localId="1">
            <inVariable localId="2">
            <transition localId="3">
            <step localId="4" name="Motor dreht links">
            <inVariable localId="5">
            <transition localId="6">
            <inVariable localId="7">
            <transition localId="8">
            <step localId="9" name="Motor dreht rechts">
            <inVariable localId="10">
            <transition localId="11">
            <selectionConvergence localId="12">
            <jumpStep localId="13" targetName="Init">
          </SFC>
        </body>
        <addData />
      </pou>
    </pous>
  </types>
  <instances>
  <addData>
</project>
```

Abb. 28 PLCopen XML Beispieldatei

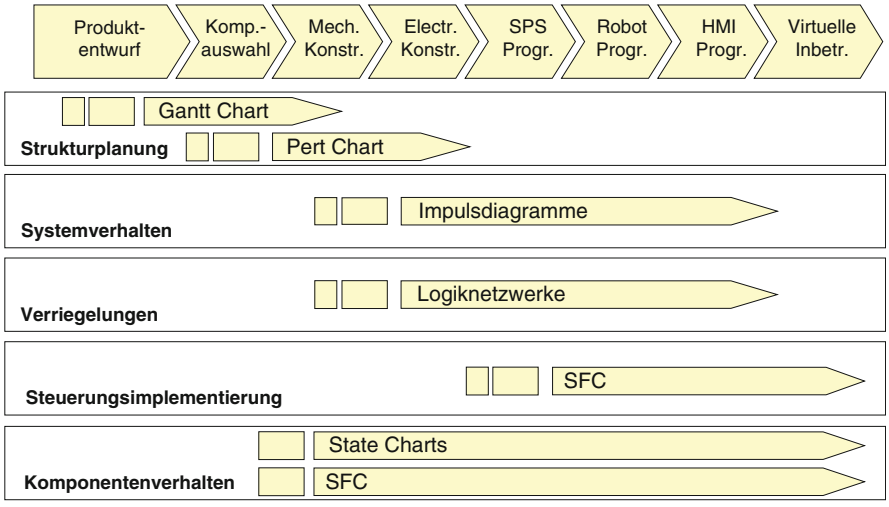

Abb. 29 Von AutomationML betrachtete Verhaltensbeschreibungen

finale Steuerungscode und das konkrete Komponentenverhalten, was als Automat abgebildet werden kann. (Hundt 2012)

Der AutomationML e.V. hatte sich zu Beginn der Entwicklungsarbeiten entschieden, nicht alle IEC 61131-3 Programmierungssprachen zu unterstützen. Da die meisten relevanten Verhaltensmodelle ereignisdiskrete Systeme abbilden, wird lediglich Ablaufsprache (AS) bzw. Sequential Function Chart (SFC) zur Modellierung des gesteuerten Verhaltens verwendet. Daher wurden Transformationsregeln definiert, die ein Mapping der eingangs genannten Verhaltensbeschreibungen (Gantt, PERT Charts, Impulsdiagramme und State Charts) zu SFC und umgekehrt ermöglichen. Weiterführende Informationen sind in (Drath 2010; AutomationML e.V.) und (Hundt 2012) zu finden. Für die Abbildung von Logiknetzwerken und die Modellierung von Komponentenverhalten (ungesteuertem Verhalten) wird die Funktionsbausteinsprache (FBS) verwendet. Sowohl AS als auch FBS Modelle können in PLCopen XML ausgedrückt werden.

Zur Referenzierung des Inhaltes einer PLCopen XML Datei hat der AutomationML e.V. eine spezielle Interfaceklasse *PLCopenXMLInterface* aus der *AutomationMLInterfaceClassLib* definiert. Verhaltensbeschreibungsrelevante Interfaceklassen werden von dieser Klasse abgeleitet. *PLCopenXMLInterface* selbst (in Abb. 30 dargestellt) ist wiederum von *ExternalDataConnector* abgeleitet und besitzt somit das Attribut *refURI*. In diesem Attribut befindet sich der Pfad zu einer PLCopen XML Datei. Die Ableitungen von *PLCopenXMLInterface* besitzen allerdings nicht mehr nur den Pfad zur Datei; Sie verweisen über den globalen Identifier auf ein konkretes Objekt (z. B. POE oder Variable) innerhalb der Datei (gemäß *file:///filename.xml#globalID*.).

Ein Beispiel zur Integration einer Verhaltensbeschreibung in ein AutomationML Projekt bzw. zur Referenzierung einer Verhaltensbeschreibung aus einem AutomationML Projekt ist in Abb. 31 zu sehen. Es zeigt, wie eine Systemunitklasse *Motor* um seine Verhaltensbeschreibung erweitert wird.

9 Modellierung von Netzwerken

In Produktionssystemen lassen sich Netzwerke unterschiedlicher Art finden, z. B. bei der Verkabelung, den Rohrleitungen, den Kommunikationssystemen oder in der Logistik. All diesen Netzwerken ist eins gemein: Sie können als graphenbasierte Strukturen dargestellt werden. Aus diesem Grund hat der AutomationML e.V. eine Methodologie zur Modellierung dieser entwickelt und auf die unterschiedlichen Netzwerkausprägungen angewandt (Lüder et al. 2013).

Ein Graph G = (V (G), E (G)) wird durch zwei nicht-leere Mengen definiert: einer Menge von Knoten (vertex) V (G) und Kanten (edge) E (G). Für beide Mengen gilt E (G) $\subseteq$ V (G) x V (G), d. h. die Knoten sind durch Kanten verbunden (Balakrishnan und Ranganathan 2012). Sollen Informationen zu diesen Objekten des Graphen hinzugefügt werden, wird das über Labels realisiert. Labels können unterschiedliche Ausprägungen annehmen, z. B. Zahlenwerte oder auch Textfelder. Bei der Entwicklung von Graphenmodellen stellen die Labels eins der wichtigsten Merkmale dar.

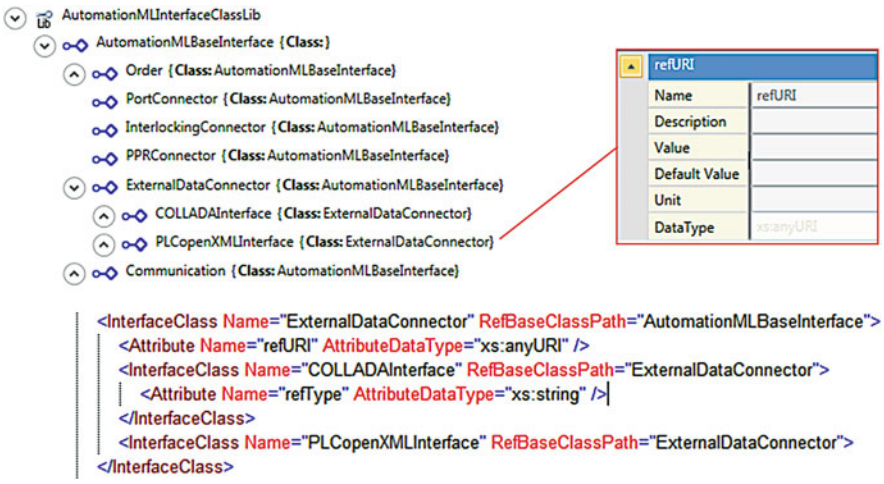

Abb. 30 Definition der Interfaceklasse *PLCopenXMLInterface*

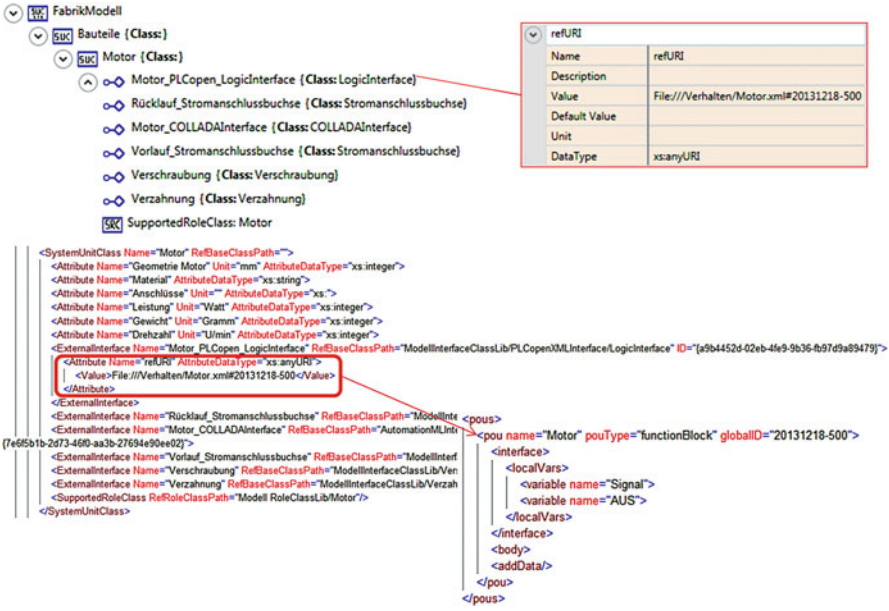

Abb. 31 Beispiel zur Referenzierung einer Verhaltensbeschreibung

Die oben gegebene Definition eines Graphen muss bei Verwendung von Labels erweitert werden: LG = (V (G), E (G), L1, L2). Solch ein gelabelter Graph weist zwei weitere Mappings aus L1, L2. Für die Mappings gilt, das die Menge von Annotationen A1 und A2 mit L1: V(G) ➔ A1 ein Mapping der Menge der Knoten

zur Menge der Annotationen 1 ist und L2: E(G) ➔ A2 ein Mapping der Menge der Kanten zur Menge der Annotationen 2 ist.

Den Startpunkt zur Graphenmodellierung bildet die Definition von Transformationsregeln, die die Abbildung der Graphenobjekte (Knoten und Kanten) auf AutomationML Objekte beschreibt. Diese Abbildung findet in CAEX statt. Es wird ein *InternalElement*, was den gesamten Graphen repräsentiert, angelegt. Merkmale und zusätzliche Informationen, die den Graphen näher beschreiben (sprich Labels), werden diesem Element als Attribute angehängt. Im nächsten Schritt werden die Knoten- und Kantenobjekte als Kindelemente unterhalb des Graphenelements angelegt. Dazu werden die Knoten sowie die Kanten mit den jeweils zugehörigen Labels als *InternalElement*s mit Attributen erstellt. Zur besseren Übersichtlichkeit empfiehlt es sich, alle Kantenelemente als Kindelemente eines zusätzlichen *InternalElements* anzulegen, was alle Kanten unter sich gruppiert. Um die Beziehungen zwischen Knoten und Kanten darzustellen, werden Interfaces genutzt. Das bedeutet, dass alle *InternalElement*s, die Knoten repräsentieren, mit so vielen Interfaces ausgestattet werden, wie sie inzidente Kanten aufweisen. Die Kanten-*InternalElement*s haben in der Regel nur zwei Interfaces. Interfaces von inzidenten Kanten und Knoten können dann mittels eines *InternalLinks* verbunden werden. Ein Beispiel findet sich in Abb. 32.

Der AutomationML e.V. hat diese Methodologie als erstes zur Modellierung von Kommunikationssystemen angewendet. Dazu wurden alle relevanten Objekte identifiziert, die als Graph modelliert werden sollten, es wurden Rollen- und Interfaceklassen definiert und ein Modellierungsvorgehen vorgeschlagen (siehe Communication Whitepaper (AutomationML e.V.)).

Jedes Kommunikationssystem besteht aus zwei Ebenen: Der logischen und der physikalischen Ebene. Die logische Ebene besteht aus „Bausteinen“ der Steuerungsapplikation, die unterschiedliche Funktionen des Steuerungsprozesses bereitstellen und selbst logische Geräte formen. Im Allgemeinen tauschen die logischen Geräte (Teile der Steuerungsapplikation) Informationen über logische Endpunkte aus. Dieser Informationsaustausch wird über logische Verbindungen zwischen den logischen Geräten realisiert. Das logische Netzwerk kann aus verschiedenen Objekten mit unterschiedlichen Eigenschaften, die diese beschrieben, bestehen. Während die logischen Geräte eindeutige Identifier, Zykluszeiten etc. haben können, können logische Endpunkte den Datentyp und logische Verbindungen die Übertragungsrate als Eigenschaft haben. In jedem Fall stellen sich diese beschreibenden Eigenschaften als Attribute der entsprechenden Objekte dar. Auf der physikalischen Ebene finden sich die physikalischen Geräte wieder. Sie besitzen physikalische Endpunkte, die die Schnittstellen wie Buchsen oder Ports repräsentieren. Die Endpunkte werden über physikalische Verbindungen verbunden. Im Unterschied zum logischen Ebene kann es auf der physikalischen Ebene zusätzliche physikalische Entitäten in Form von Infrastrukturkomponenten (wie Switches etc.) geben. Allerdings können die Objekte beider Ebenen beschreibende Eigenschaften aufweisen. Während die physikalischen Geräte Eigenschaften wie Prozessorleistung oder Identifiers haben können, können physikalische Endpunkte eine Adresse oder eine maximale Datenrate und die physikalischen Verbindungen eine mögliche Übertragungsrate oder Kabeltyp haben. In

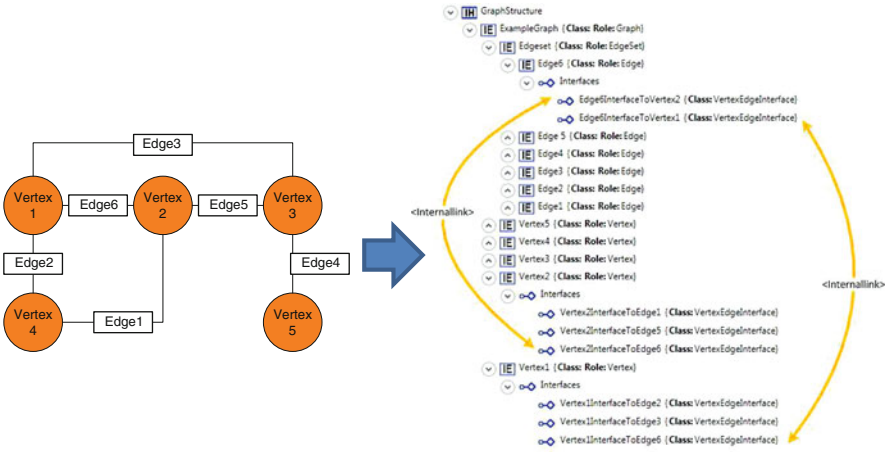

Abb. 32 Beispiel eines Graphenmodells in AutomationML

jedem Fall stellen sich diese beschreibenden Eigenschaften als Attribute der entsprechenden Objekte dar.

Beide Ebenen müssen jedoch für eine vollständige Netzwerkbeschreibung miteinander in Beziehung gesetzt werden. Dafür werden die logischen Geräte innerhalb der physikalischen Geräte untergebracht; Die physikalischen Geräte implementieren die Logischen. Zusätzlich werden die zusammengehörenden logischen und physikalischen Endpunkte aufeinander gemappt. Ein striktes Mapping der logischen Verbindungen auf die Physikalischen ist nicht vorgeschrieben, da bei manchen Kommunikationstechnologien die zu übertragende Information auf ihrem Weg zum Zielgerät einen Umweg über weitere physikalische Geräte nehmen kann. Ein solches Beispiel ist in Abb. 33 zu sehen.

Innerhalb der Kommunikationssysteme werden auch Kommunikationsdatagramme (bekannt als Protocol Data Units/PDU) zwischen den einzelnen Teilen der Steuerungsapplikation ausgetauscht. Diese PDUs sind der logischen Verbindung zusortiert und beinhalten Steuerungsinformationen (wie Sensor- und Aktorsignale, Stati, Alarme etc.), die in AutomationML als Interfaces von Typ *PLCopenXMLInterface* modelliert werden (siehe oben). Das bedeutet, dass jede logische Verbindung ein PDU-Objekt enthält, was über diese Verbindung ausgetauscht wird. Jedes der PDU-Objekte ist mit einem *PLCopenXMLInterface* oder *SignalInterface* verbunden, die die ausgetauschte Information repräsentiert.

Als Grundlage der Methode zur Kommunikationssystemmodellierung in AutomationML dient die Definition von spezifischen Rollen- und Interfaceklassen. Die AutomationML Kommunikationsrollenklassenbibliothek umfasst bestimmte Rollenklassen, die *InternalElement*s als physikalische Geräte, physikalische Verbindungen etc. sowie als logische Geräte, logische Verbindungen etc. oder Kommunikationspakete kennzeichnen. Die AutomationML Kommunikationsinterfaceklassenbibliothek umfasst bestimmte Interfaceklassen, die *ExternalInterface*s als physikalische Endpunkte, logische

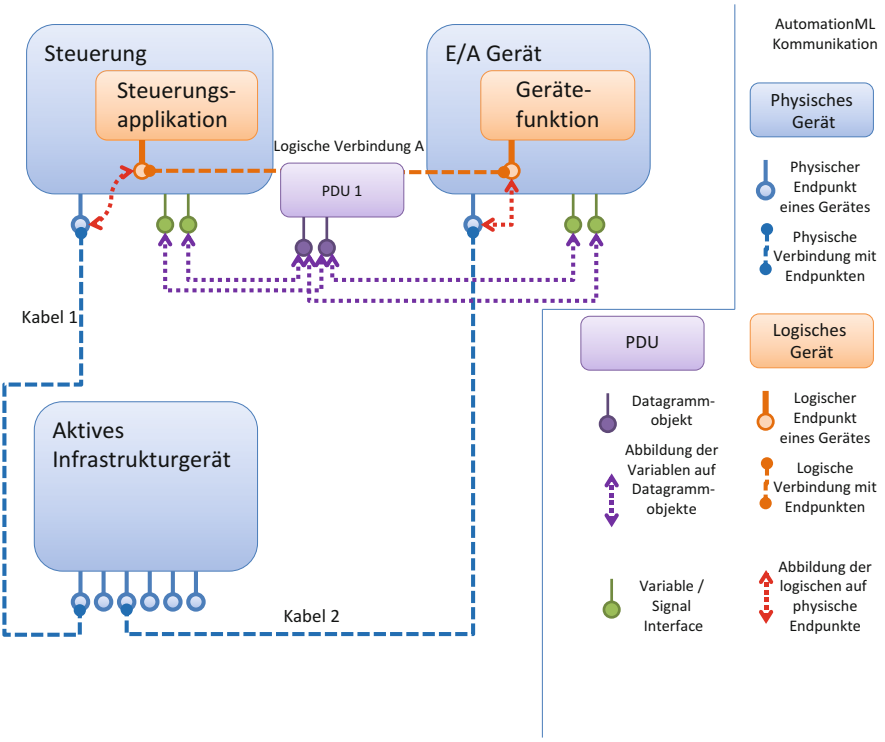

Abb. 33 Schematische Darstellung eines Kommunikationssystems in AutomationML

Endpunkte etc. kennzeichnen. Beide Bibliotheken, *CommunicationRoleClassLib* und *CommunicationInterfaceClassLib,* sind im oberen Teil von Abb. 34 zu sehen.

Auf Grundlage dieser allgemeinen Rollen- und Interfaceklassen können je nach Anwendungsfall kommunikationstechnologie- und kommunikationsprotokollabhängig Rollen- und Interfaceklassen abgeleitet werden. Ein Beispiel ist im unteren Bereich von Abb. 34 zu sehen: *ModbusTCPRoleClassLib* und *ModbusTCPInterfaceClassLib*.

Die anwendungsfallabhängigen Rollen- und Interfaceklassenbibliotheken können bei der Erstellung von gängigen physikalischen/logischen Geräten/Verbindungen als Systemunitklassen genutzt werden.

Zur eindeutigen Identifizierung der Semantik der unterschiedlichen Systemunitklassen werden die definierten Rollenklassen herangezogen und verwendet.

Jedes physikalische Gerät wird mit so vielen physikalischen Endpunkten ausgestattet, wie physikalische Ports vorhanden sind. Diese werden in die sog. *Endpointlist* aufgenommen. Jedes logische Gerät wird mit so vielen logischen Endpunkten ausgestattet, wie es logische Applikationszugangspunkte gibt. Auch diese werden in einer *Endpointlist* aufgenommen.

Jede physikalische Verbindung wird mit so vielen physikalischen Endpunkten ausgestattet, zu wie vielen physikalischen Geräten sie verbunden werden kann. Im Fall der üblichen Verkabelung gibt es genau zwei Endpunkte. Jede logische Verbin-

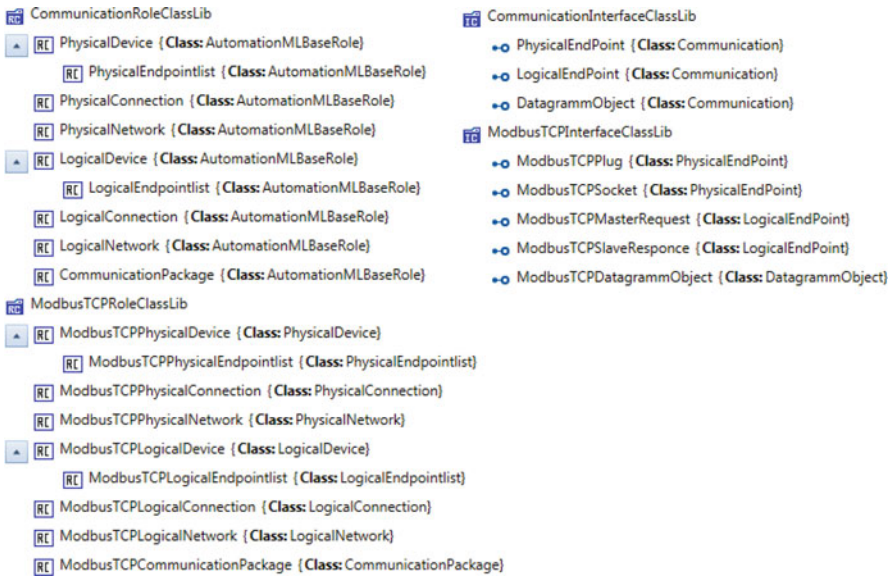

Abb. 34 Allgemeine Rollen- und Interfaceklassen sowie konkrete Ableitungen davon

dung wird mit so vielen logischen Endpunkten ausgestattet, zu wie vielen logischen Geräten sie verbunden werden kann. Im Fall einer Master-Slave-Kommunikation gibt es genau zwei Endpunkte. Im Fall einer Multicast-Kommunikation gibt es mehr als zwei Endpunkte.

Zur Abbildung von spezifischen Eigenschaften der verschiedenen Systemunitklassen müssen entsprechende Attribute modelliert werden.

Zur Abbildung von PDUs werden entsprechende Systemunitklassen definiert und mit einer Ableitung der Rolle *CommunicationPackage* versehen. Innerhalb der Klasse wird außerdem für jede zu übertagende Information ein Interface angelegt, was von der Interfaceklasse *DatagrammObject* abgeleitet ist. Zur Abbildung von spezifischen Eigenschaften der verschiedenen PDUs müssen entsprechende Attribute modelliert werden. Eine beispielhafte Systemunitklassenbibliothek ist in Abb. 35 zu sehen.

Aus der Systemunitklassenbibliothek heraus kann das Kommunikationssystem modelliert werden. Dafür werden alle notwendigen physikalischen und logischen Geräte als *InternalElement*s in der *InstanceHierarchy* instanziiert. Besonders die hierarchische Struktur des zu modellierenden Systems muss bewahrt werden. Das gilt vor allem für die Integration von logischen Geräten in den physikalischen Geräten (siehe Abb. 36; logisches Gerät *MyPIProgram*, physikalisches Gerät *PIBasedController1*).

Nachdem alle Geräte modelliert wurden, müssen die Attribute der Geräte u. U. vervollständigt, aber auf jeden Fall mit Werten gefüllt werden.

Im Anschluss können die Verbindungen verbunden werden. Dafür werden in der *InstanceHierarchy* des Netzwerkes zwei *InternalElement*s angelegt und denen abgeleitete Rollen der Rollenklassen *PhysicalNetwork* und *LogicalNetwork* zugewiesen.

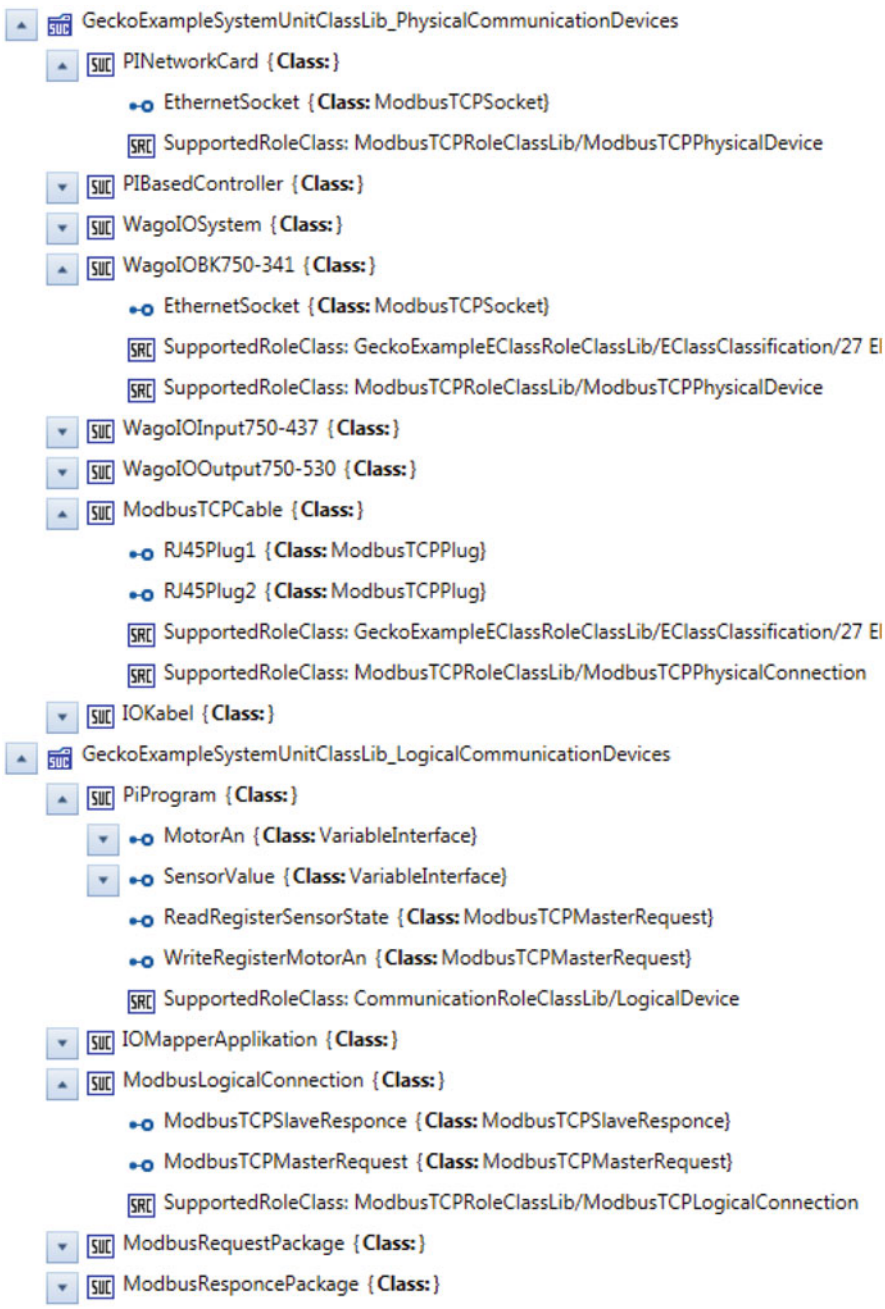

Abb. 35 Beispielhafte Systemunitklassenbibliothek

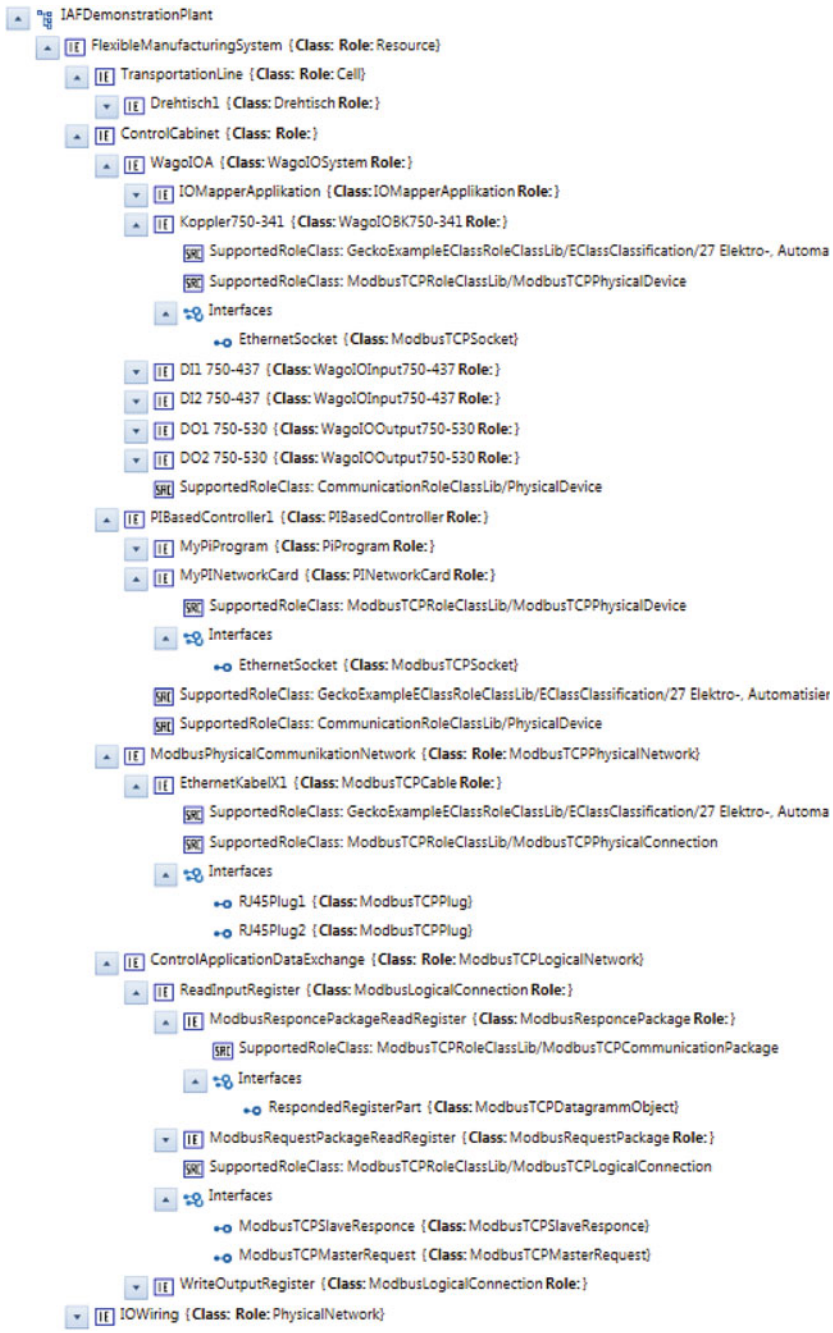

Abb. 36 Beispielhafte Kommunikationssystemhierarchie

Sie fungieren als Container für jeweils die physikalischen Verbindungen und für die Logischen. Für jede physikalische Verbindung wird ein *InternalElement* angelegt, was eine Rolle abgeleitet von der Rollenklasse *PhysicalConnection* bekommt. Die Modellierung der Verbindungen wird durch die Vervollständigung der notwendigen Attribute und des Befüllens der Attribute mit Werten abgeschlossen. Gleiches wird für die Modellierung von logischen Verbindungen getan.

Wenn alle relevanten Geräte und Verbindungen in der *InstanceHierarchy* instanziiert wurden, müssen die in einem nächsten Schritt via *InternalLinks* miteinander verbunden werden. Die Geräte und Verbindungen, die in Beziehung zueinander stehen, werden über deren Interfaces mittels *InternalLink*s verbunden. Um die logischen Endpunkte mit den physikalischen Endpunkten in Verbindung zu setzen, werden auch sie mittels eines *InternalLink*s verbunden. Für das Beispiel zeigt Abb. 37 die sich ergebende Struktur.

Zur Abbildung von PDUs werden entsprechende *InternalElement*s angelegt und mit einer Ableitung der Rollenklasse *CommunicationPackage* versehen. Innerhalb des *InternalElement*s wird außerdem für jede zu übertagende Information ein Interface angelegt, was von der Interfaceklasse *DatagrammObject* abgeleitet ist. Zur Abbildung von spezifischen Eigenschaften der verschiedenen PDUs müssen entsprechende Attribute an *InternalElement* und Interface modelliert werden. Die Interfaces vom Typ *DatagrammObject* sind via *InternalLinks* mit den Interfaces vom Typ *PLCopenXMLInterface* oder *SignalInterface* verbunden.

Eine sehr spezifische Anwendung der Graphenmodellierung, die gleichzeitig eine Anwendung und eine Erweiterung der Kommunikationssystemmodellierung darstellt, wird durch die Modellierung von Steuerungskonfigurationen gebildet. Diese wird in AutomationML über die Application Recommendation (AR) Automation Project Configuration (APC) abgebildet. Sie basiert auf der Definition spezifischer Knotenklassen, die die hardwareorientierte Gerätehierarchie in einem Automatisierungsnetzwerk und die Beziehungen in dieser Gerätehierarchie abbildet. Diese wurde gemäß der oben beschriebenen Regeln in AutomationML Rollen- und Interfaceklassen übertragen, was in Abb. 38 dargestellt ist.

Diese spezifische Modellierung von Netzwerken hat bereits weite praktische Verbreitung gefunden und wird von allen namhaften SPS Programmiersystem- und ECAD-System-Herstellern unterstützt.

10 Integration von weiteren, externen Informationen

AutomationML besitzt mit seinen Interfaces ein Modellierungsmittel, was dazu genutzt werden kann, Informationen in externen Dateien zu referenzieren. Dieses Modellierungsmittel wird u. a. zur Referenzierung von Geometrie- und Kinematikinformationen, gespeichert in COLLADA Dateien, sowie von Verhaltensinformationen, gespeichert in PLCopen XML Dateien, genutzt. Zu diesem Zweck werden geeignete Interfaceklassen von der generischen Interfaceklasse *ExternalDataConnector* abgeleitet.

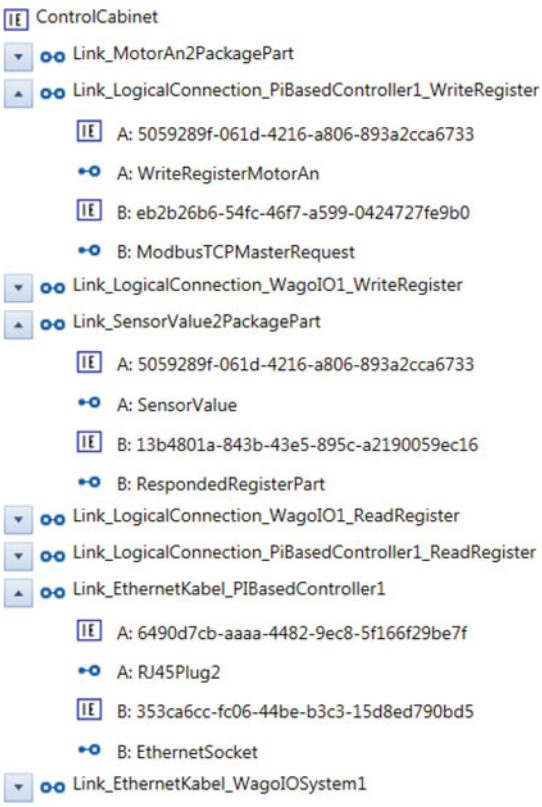

Abb. 37 InternalLinks innerhalb des Beispiels

Diese Interfaceklasse wird in diesem Fall dazu genutzt, neue Interfaceklassen von ihr abzuleiten, mit denen externe Dateien, z. B. technische Datenblätter, Abbildungen, Handbücher etc., referenziert werden können. Als Beispiel kann hier das *ExternalDataReference* genannt werden, was vom *ExternalDataConnector* abgeleitet ist. Die *ExternalDataReference* besitzt als Attribut die *refURI*, in die der Pfad zur externen Datei hinterlegt ist, und ein Attribut *MIMEType*, welches den Dokumententyp gemäß MIME Standard (Multipurpose Internet Mail Extensions) spezifiziert. Abhängig von der Anzahl der Dokumente, welche einem Objekt zugewiesen werden sollen, werden *InternalElement*s als Kindelemente angelegt. Zusätzlich wird eine Rollenklassenbibliothek modelliert, die eine bestimmte Semantik definiert, z. B. Rollenklasse für technische Datenblätter, Abbildungen, Handbücher etc. Diese Rollenklassen können dann dem entsprechenden *InternalElement* zugewiesen werden.

Die beschriebene Vorgehensweise kann für mehrere Datenformate angewendet werden (Lüder et al. 2014). Ein Beispiel für die Integration von Handbüchern im PDF Format als zusätzliche Information ist in Abb. 38 dargestellt.

Aktuell arbeitet der AutomationML e.V. an der Spezifikation der Mechanismen, um extern abgespeicherte Informationen Objekten zuwiesen zu können (Abb. 39).

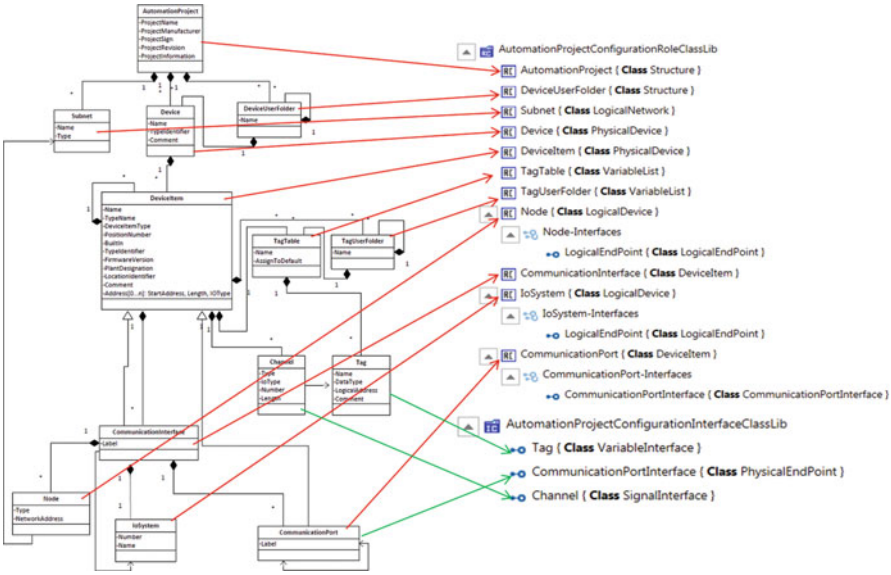

Abb. 38 Konzeptmenge der AR APC

An dieser Stelle soll explizit darauf hingewiesen werden, dass die hier am Beispiel eines PDF vorgestellte Methode für jeden beliebigen Dateityp anwendbar ist. Das gilt auch und insbesondere für Geometrieformate wie JT. Hier besteht jedoch ein entscheidender Unterschied zu COLLADA Integration. Andere Geometrieformate können nur als Block integriert werden und beschreiben damit die Geometrie zu einem AutomationML Objekt, das als *InternalElement* abgebildet ist. Eine Identifikation von Objekten in der Geometriedatei ist nur bei COLLADA möglich.

11 Anwendungsprozess

Die AutomationML Spezifikationen definieren im Allgemeinen die Abbildung von Informationen, die auf der Anwendung von CAEX, COLLADA und PLCopen XML basieren. Allerdings erzwingt die Nutzung von AutomationML einen Anwendungsprozess, welcher mehr oder weniger definiert ist. Dieser Prozess basiert auf einer generellen Sicht auf den Datenaustausch und besteht aus zwei Phasen, die als erstes die Identifikation der Daten, die ausgetauscht werden müssen, umfasst und als zweites die Modellierung der identifizierten Daten.

Grundlage der Anwendung von AutomationML bildet der allgemeine Blick auf den Datenaustausch zwischen zwei oder mehr Entwurfswerkzeugen (siehe Abb. 40). Jedes Entwurfswerkzeug, was am Entwurfsprozess beteiligt ist, hat üblicherweise sein eigenes Datenmodell, welches optimal an die eigenen Toolzwecke angepasst ist. Aus diesem Grund unterscheiden sich die Datenmodelle der einzelnen Softwarewerkzeuge in der Regel. Um einen Datenaustausch zwischen den Tools jedoch zu ermöglichen,

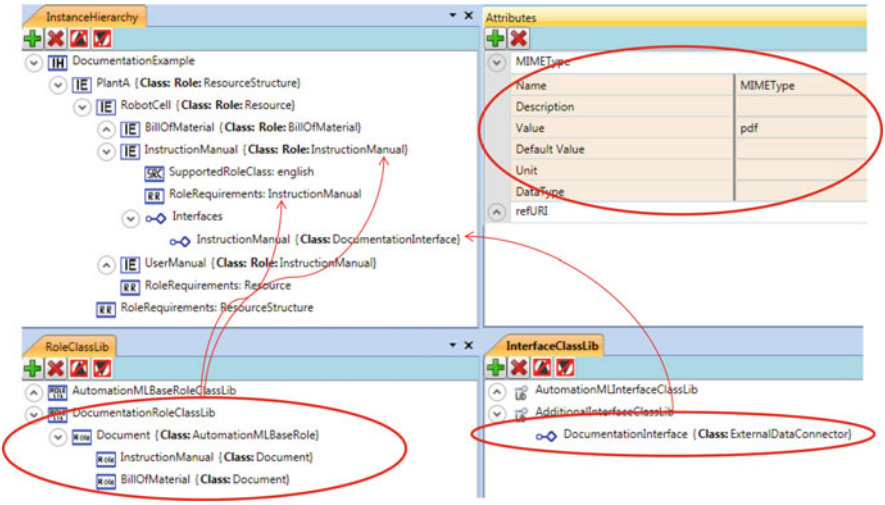

Abb. 39 Beispiel zur Integration eines PDF Dokumentes in AutomationML

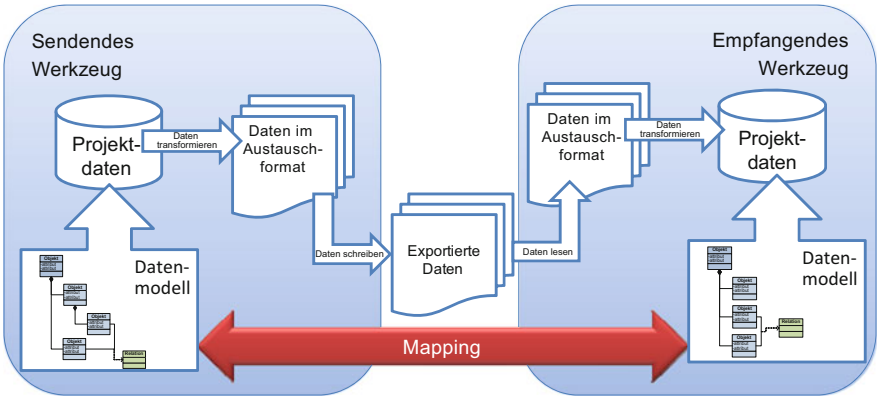

Abb. 40 Notwendiges Mapping der Datenmodelle für die Anwendung von AutomationML

muss das sendende Tool seine Daten geeignet umwandeln und in das Datenaustauschformat schreiben. Das empfangende Tool muss dagegen die Daten interpretieren und diese geeignet in sein eigenes Datenmodell schreiben können. Somit entspricht die Kombination von Import und Export einem Mapping der internen Datenmodelle der beteiligten Tools. Dieses Datenmapping und auch die Modellierung der auszutauschenden Daten müssen vor dem Einsatz von AutomationML geklärt sein.

Demnach ist der erste Schritt der Anwendung von AutomationML die detaillierte Analyse des Entwurfsprozesses, in dem das Datenaustauschformat zum Einsatz kommen soll. Die Analyse umfasst dabei die detaillierte Betrachtung der beteiligten Entwurfsaktivitäten, -artefakte und -werkzeuge. An dieser Stelle muss herausgear-

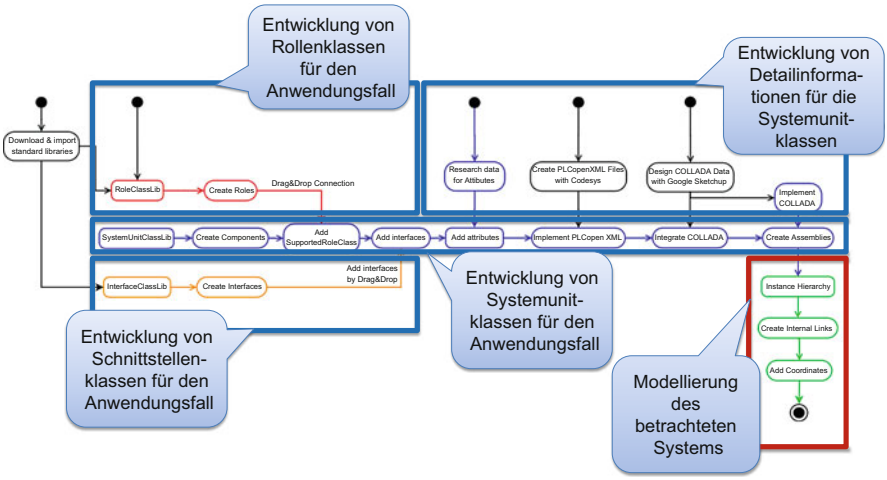

Abb. 41 Zweite Phase des impliziten AutomationML Anwendungsprozesses

beitet werden, welche Daten ausgetauscht werden sollen und wie diese untereinander zusammenhängen. Auf diese Weise wird ein Gesamtmodell der auszutauschenden Daten erstellt. Für jedes Tool muss dann erarbeitet werden, welche dieser Daten welchen Daten im eigenen, internen Datenmodell des Tools entsprechen.

Im nächsten Schritt muss die Abbildung des Gesamtmodells in AutomationML entwickelt werden. Dieser Prozess beginnt, wie in Abb. 41 dargestellt, (die Semantik betrachtend) mit der Identifikation der wichtigsten Objekttypen und ihrer möglicher Relationen. Damit können die notwendigen Rollen- und Interfaceklassen erstellt und mit beschreibenden Eigenschaften bzw. Attributen ausgestattet werden. Beide sind in den entsprechenden Bibliotheken abgelegt. Als Nächstes werden die für den Anwendungsfall als sinnvoll erachteten, wiederverwendbaren Objekte bzw. Komponenten identifiziert. Diese Komponenten werden mit Substrukturen, Interfaces, Attributen etc. als Systemunitklassen (in einer Systemunitklassenbibliothek) modelliert. Beim Importieren und Exportieren können diese vom Softwarewerkzeug zum schnelleren Datenmapping herangezogen werden. Mit der Fertigstellung der Bibliotheken ist die Modellierung der Daten in AutomationML abgeschlossen. Die Erstellung der Bibliotheken findet in der Regel nur einmal statt. Währenddessen die Erstellung der eigentlichen Daten – der Instanzdaten – im Laufe des Entwurfsprozesses mehrmals stattfinden kann. Auf der anderen Seite ist es auch vorstellbar, dass die Bibliotheken erst sukzessive, parallel zum Entwurfsprozess entwickelt werden.

12 Wachsende Rolle in der Industrie 4.0

Wie bereits oben beschrieben, wurde und wird AutomationML entwickelt, um den verlustfreien Datenaustausch entlang der Entwurfsketten von Produktionssystemen sicherzustellen. Die dabei entstehenden und ausgetauschten Entwurfsdaten können

jedoch auch in späteren Phasen des Lebenszyklus von Produktionssystemen interessant sein.

Hier propagiert die Industrie 4.0 Initiative die Nutzung der AutomationML basierten Entwurfsdaten zur (zumindest teilweisen) Erstellung der Verwaltungsschale für eine Industrie 4.0 Komponente (Plattform Industrie 4.0 2019). Ausgangspunkt dieser Überlegungen ist die Feststellung, das im Rahmen der Architektur für eine Industrie 4.0 Komponente AutomationML in Kombination zum Beispiel mit OPC UA in der Lage ist, die Kommunikations- und Informationsebenen des 7-Ebenen-Modelles einer Industrie 4.0 Komponente abzubilden und damit die Technologien für die Realisierung einer Verwaltungsschale bereitzustellen (siehe Abb. 42).

Entsprechend wurde in Kooperation von AutomationML e.V. und OPC Foundation eine Companion-Spezifikation erstellt, die die Abbildung von AutomationML Projekten auf OPC UA Nodesets beschreibt. Diese wurde als DIN Spec 16592 standardisiert (DIN Spec 16592 2016). Damit und in Kombination mit anderen Companion-Spezifikationen wird es möglich, eine, der in Abb. 43 dargestellten, analoge Architektur zu realisieren. In dieser bildet ein AutomationML basierter OPC UA Server die relevanten Entwurfsinformationen ab und verknüpft diese mit anderen (zum Beispiel Laufzeit-) Informationen, die auf anderen OPC UA Servern vorgehalten werden (Lüder et al. 2017).

Zum Zeitpunkt der Erstellung dieses Beitrages (Juni 2019) arbeitet eine gemeinsame Arbeitsgruppe der Industrie 4.0 Plattform und des AutomationML e.V. an der detaillierten Spezifikation, wie welche Informationen, die in einer Verwaltungsschale für eine Industrie 4.0 Komponente relevant sind, in AutomationML abzubilden sind. Ergebnisse werden hier Ende 2019 erwartet.

13 Zusammenfassung

In diesem Beitrag wurde der aktuelle Entwicklungsstand des Datenaustauschformats AutomationML präsentiert. In den letzten dreizehn Jahren – vom Beginn der Entwicklung von AutomationML mit neun Unternehmen und Forschungseinrichtungen bis zum heutigen AutomationML e.V. mit mehr als 50 Mitgliedern – entstand ein verbreitetes Datenaustauschformat, welches die an den Entwurfsprozess von Produktionssystemen gestellten Anforderungen erfüllt. Es ist in der Lage, die Ergebnisse von der Planung der Anlagenstruktur über den mechanischen, elektrischen und Steuerungsentwurf sowie der Rohrleitungsplanung, der virtuelle Inbetriebnahme bis hin zur Installation und Inbetriebnahme abzuspeichern und damit zwischen Softwareentwurfswerkzeugen austauschbar zu machen. Dies ermöglicht den Entwurfswerkzeugen, in eine heterogene Werkzeuglandschaft integriert werden zu können, wie es im Rahmen der Industrie 4.0 Initiative gefordert ist.

Heute beschleunigt der AutomationML e.V. seinen Schritt in Sachen Standardisierung und bei der Entwicklung von Anwendungsempfehlungen (ARs) für konkrete Anwendungsfälle. Auf der einen Seite geht es um den Austausch von bestimmten Entwurfsinformationen in bestimmten Anwendungsfällen. Hier liegt der aktuelle Fokus u. a. auf der Beschreibung von wiederverwendbaren Komponenten der verschiedenen Strukturebenen von Produktionssystemen, der Integration von in

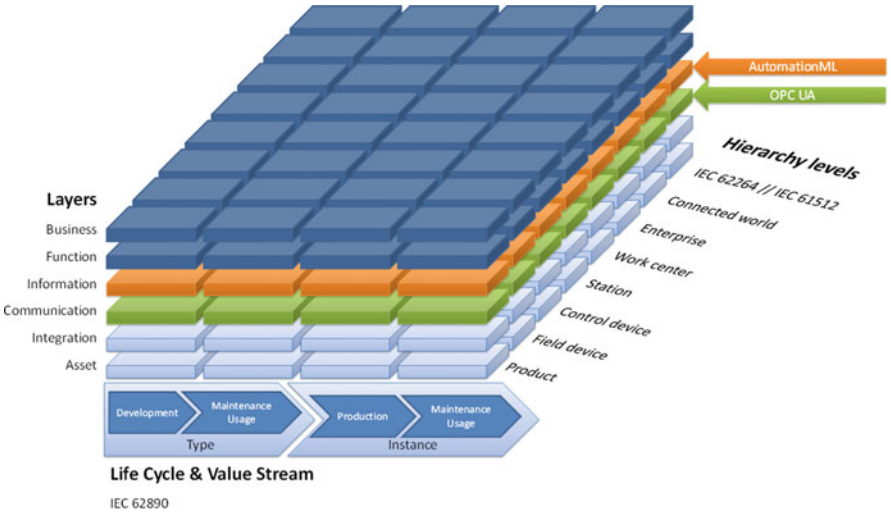

Abb. 42 Einordnung von AutomationML und OPC UA in die Industrie 4.0 Architektur

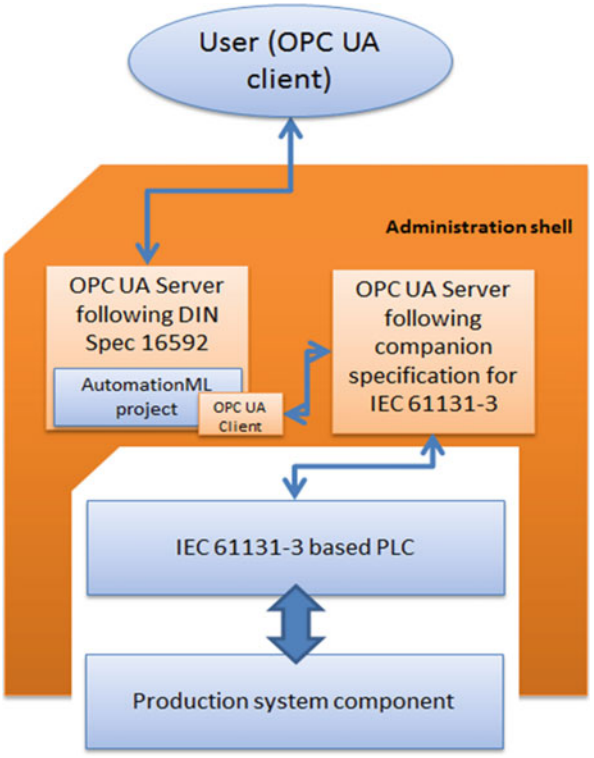

Abb. 43 Technologie-orientierte Realisierungsidee für eine Verwaltungsschale zu einer Industrie 4.0 Komponente

AutomationML modellierten Daten in Datenmanagementsystemen, der Modellierung spezifischer häufig relevanten Anlagenstrukturen wie Antriebssträngen und der Identifikation von Anforderungen innerhalb des Entwurfsprozesses. Auf der anderen Seite werden Vorgehensweisen beim Datenaustausch untersucht sowie Empfehlungen zur Datenstrukturierung erarbeitet. Es werden beispielsweise die Modellierung von Produktionsprozessen unter Nutzung von Ressourcen zur Herstellung von Produkten und die Definition von spezialisierten Rollenklassenbibliotheken für den Materialfluss betrachtet.

Das Hauptaugenmerk der weiteren Entwicklungen von AutomationML liegt auf der optimalen Ausrichtung bzw. Anpassung und Anwendbarkeit des Datenformats bzgl. der vollständigen Erfüllung der Anforderungen einer Integration in die Entwurfskette, wie es die Industrie 4.0 Initiative vorschlägt. Interessierte Unternehmen und Forschungseinrichtungen sind herzlich eingeladen, sich an diesen Arbeiten zu beteiligen.

Literatur

Alonso-Garcia A, Hirzle A, Burkhardt A (2008) Steuerungstechnische Standards als Fundament für die Leitechnik. ATP (9):42–47

Arnaud R, Barnes M (2006) COLLADA – sailing the gulf of 3D digital content creation. A K Peters, LTD, Wellesley

AutomationML e.V: AutomationML web page. www.automationml.org. Zugegriffen im Februar 2015

Balakrishnan R, Ranganathan K (2012) A textbook of graph theory. Springer, New York

Biffl S, Lüder A, Gerhard D (Hrsg) (2017) Multi-disciplinary engineering for cyber-physical production systems – data models and software solutions for handling complex engineering projects. Springer, Cham, ISBN 978-3-319-56344-2

Diedrich C, Lüder A, Hundt L (2011) Bedeutung der Interoperabilität bei Entwurf und Nutzung von automatisierten Produktionssystemen. at – Automatisierungstechnik 59(7):426–438

DIN Spec 16592 (2016) Combining OPC unified architecture and automation markup language. Beuth Publisher, Berlin. https://www.beuth.de/de/technische-regel/din-spec-16592/265597431. Zugegriffen im August 2019

DIN/DKE: Deutsche Normungsroadmap Industrie 4.0, Version 3, März 2018. https://www.din.de/de/forschung-und-innovation/themen/industrie4-0/roadmap-industrie40-62178

Drath R (2010) Datenaustausch in der Anlagenplanung mit AutomationML. Springer, Heidelberg

Drath R, Fay A, Barth M (2011) Interoperabilität von Engineering-Werkzeugen. at – Automatisierungstechnik 59(7):451–460

eCl@ss association: eCl@ss classification system. http://wiki.eclass.de/wiki/Main_Page

Hundt L (2012) Durchgängiger Austausch von Daten zur Verhaltensbeschreibung von Automatisierungssystemen, PhD Thesis, Faculty of Mechanical Engineering, Otto-von-Guericke University Magdeburg

Hundt L, Lüder A (2012) Development of a method for the implementation of interoperable tool chains applying mechatronical thinking – use case engineering of logic control. In: Proceedings of 17th IEEE international conference on Emerging Technologies and Factory Automation (ETFA 2012), Krakow, Sept 2012

International Electrotechnical Commission (2008) IEC 62424 – Representation of process control engineering – requests in P&I diagrams and data exchange between P&ID tools and PCE-CAE tools. www.iec.ch

International Electrotechnical Commission (2014) IEC 62714 – engineering data exchange format for use in industrial automation systems engineering- AutomationML. www.iec.ch
International Electrotechnical Commission (2019) IEC 61131-10 – programmable controllers – Part 10: PLC open XML exchange format. www.iec.ch
International Organization for Standardization (2012) ISO/PAS 17506:2012 – industrial automation systems and integration – COLLADA digital asset schema specification for 3D visualization of industrial data. www.iso.org
Jasperneite J (2012) Industrie 4.0 – Alter Wein in neuen Schläuchen? Computer&Automation 12(12):24–28
Kagermann H, Wahlster W, Helbig J (Hrsg) (2013) Umsetzungsempfehlungen für das Zukunftsprojekt Industrie 4.0 – Deutschlands Zukunft als Industriestandort sichern, Forschungsunion Wirtschaft und Wissenschaft, Arbeitskreis Industrie 4.0. http://www.plattform-i40.de/sites/default/files/Umsetzungsempfehlungen%20Industrie4.0_0.pdf. Zugegriffen im November 2013
Kiefer J, Baer T, Bley H (2006) Mechatronic-oriented engineering of manufacturing systems taking the example of the body shop. In: Proceedings of 13th CIRP international conference on life cycle engineering, Leuven, June 2006. http://www.mech.kuleuven.be/lce2006/064.pdf
Lindemann U (2007) Methodische Entwicklung technischer Produkte. Springer, Heidelberg
Lüder A, Foehr M, Hundt L, Hoffmann M, Langer Y, St. Frank (2011) Aggregation of engineering processes regarding the mechatronic approach. In: Proceedings of 16th IEEE international conference on Emerging Technologies and Factory Automation (ETFA 2011), Toulouse, Sept 2011
Lüder A, Schmidt N, Helgermann S (2013) Lossless exchange of graph based structure information of production systems by AutomationML. In: Proceedings of 18th IEEE international conference on Emerging Technologies and Factory Automation (ETFA 2013), Cagliari, Sept 2013
Lüder A, Schmidt N, Rosendahl R, John M (2014) Integrating different information types within AutomationML. In: Proceedings of 19th IEEE internernational conference on Emerging Technologies and Factory Automation (ETFA), Sept 2014, Barcelona
Lüder A, Schleipen M, Schmidt N, Pfrommer J, Henßen R (2017) One step towards an Industry 4.0 component. In: Proceedings of 13th IEEE Conference on Automation Science and Engineering (CASE), Aug 2017, Xi'an
Lüder A, Pauly J-L, Kirchheim K (2019) Multi-disciplinary engineering of production systems – challenges for quality of control software, software quality: the complexity and challenges of software engineering and software quality in the cloud. In: Proceedings of international conference on Software Quality (SWQD 2019), Springer International Publishing, Cham, S 3–13
Plattform Industrie 4.0 (2019) Details of the asset administration shell – part 1 – the exchange of information between partners in the value chain of Industrie 4.0. https://www.plattform-i40.de/I40/Redaktion/DE/Downloads/Publikation/2018-verwaltungsschale-im-detail.html. Zugegriffen im December 2019
PLCopen association (2012) PLCopen XML. www.plcopen.org
Schmidt N, Lüder A, Steininger H, Biffl S (2014) Analyzing requirements on software tools according to the functional engineering phase in the technical systems engineering process. In: Proceedings of 19th IEEE international conference on Emerging Technologies and Factory Automation (ETFA), Sept 2014, Barcelona
Terkaj W, Tolio T, Valente A (2009) Focused flexibility in production systems, in changeable and reconfigurable manufacturing systems. Springer Series in Advanced Manufacturing, vol I, London, S 47–66
VDI/VDE – GMA Fachausschuss 7.21 „Industrie 4.0": VDI-Statusreport Industrie 4.0 – Wertschöpfungsketten. VDI, Frankfurt am Main. http://www.vdi.de/fileadmin/vdi_de/redak-teur_dateien/gma_dateien/VDI_Industrie_4.0_Wertschoepfungsketten_2014.pdf. Zugegriffen im August 2019
Xu X, Nee A (2009) Advanced design and manufacturing based on STEP. Springer, London