AutomationML als Grundlage für einen durchgängigen Modellierung-, Simulations- und Integrationsprozess in der Anlagenplanung

M.Sc. **Sebastian Faltinski**, Prof. Dr. **Oliver Niggemann**,
Fraunhofer IOSB-INA;
Dipl.-Math. **Natalia Moriz**, B. Sc. **Nikolai Schetinin**,
Hochschule Ostwestfalen-Lippe

Kurzfassung

Planung, Test und Integration von automatisierten Anlagen wird in den letzten Jahren immer aufwändiger und wird zunehmend zu einem Engpass während der Entwicklung. Häufig werden Hardware und Steuerungssoftware von Produktionsanlagen parallel entwickelt, können jedoch erst zusammen getestet werden, wenn alle Komponenten fertiggestellt und integriert sind. Dies geschieht häufig erst im Feld und führt dadurch zu einer langen Inbetriebnahmezeit, da beispielsweise Fehler in der Steuerungssoftware behoben werden müssen.

Um die Steuerungssoftware solcher Anlagen bereits vor der Inbetriebnahme testen zu können, ist ein Modell der Anlage notwendig, gegen das die Steuerungssoftware getestet werden kann. Um solche Modelle während der Entwicklung zu erzeugen, ist ein moderner Entwicklungsprozess notwendig. Dieser muss in der Lage sein, die Daten verschiedener Planungswerkzeuge zu integrieren, die Wiederverwendung von bereits vorhandenen Modellen zu ermöglichen, ausführbare Modelle für die Simulation bereitzustellen und eine systemweite Planung zu ermöglichen.

In diesem Beitrag beschreiben wir exemplarisch einen solchen Entwicklungsprozess sowie die dafür notwendigen Werkzeuge auf prototypischer Basis. Der gezeigte Ansatz basiert auf dem Datenformat AutomationML (AML), welches die Integration unterschiedlicher Modelle zu einem Gesamtmodell ermöglicht.

1. Einleitung und Motivation

Eine lückenlose Entwicklungskette für die Planung und Wartung von industriellen Anlagen ist bisher eine ungelöste Aufgabe. Bisher werden für unterschiedliche Anwendungsfälle verschiedene Softwarewerkzeuge eingesetzt (CATIA, AutoCAD, Delmia, STEP 7, CoDeSys uvm.), was einen Datenaustausch zwischen diesen notwendig macht, um eine Anlage vollständig zu planen. Gerade hier kommt es jedoch häufig zu Problemen, da jedes dieser

Werkzeuge unterschiedliche Dateiformate für die Datenablage verwendet. Ein Datenaustausch zwischen einem CAD-Werkzeug (CATIA usw.), wie es im Maschinenbau verwendet wird, und einem E-CAD-Werkzeug (ePlan) aus der Elektroplanung ist daher nur schwer möglich. Ein solcher Austausch würde jedoch eine aufeinander aufbauende Planung ermöglichen. Die elektrische Verkabelung könnte so auf bereits vorhandenen mechanischen Daten aufbauen und die mechanischen Gegebenheiten (z.B. Positionen von Aktoren und Sensoren) der Anlage berücksichtigen, ohne diese erneut in das System eingeben zu müssen. Häufige Fehler in den Planungsdaten, die durch lückenhafte oder fehlerhafte manuelle Datenübertragung in ein Planungstool entstehen, werden so vermieden. Dieser Datenaustausch verkürzt einerseits die Planungszeit, andererseits reduziert sich die Fehleranfälligkeit des Planungsprozesses.

Um die einzelnen Werkzeuginseln der verschiedenen Gewerke zu verbinden, bietet das neue XML-basierte Dateiformat AutomationML einen vielversprechenden Ansatz [1]. Dieses Format wird von Unternehmen wie Daimler, ABB, KUKA, Siemens und verschiedenen Forschungsinstituten entwickelt. Ziel dabei ist ein Dateiformat für den herstellerunabhängigen Datenaustausch für verschiedene Bereiche der Anlagenplanung.

AML beschreibt dabei jedoch keinen vorgegebenen Entwicklungsprozess, sondern ermöglicht eine Integration in den firmeninternen Entwicklungsprozess. Andererseits lässt sich auf Basis von AML ein neuer Entwicklungsprozess aufbauen, welcher moderne Entwicklungsverfahren bietet. Dieser unterstützt folgende Paradigmen, die auch in Bild 1 als Ablauf beschrieben sind:

Paradigma 1: Einen nahtlosen Entwicklungsprozess von automatisierten Anlagen durch erleichterten Datenaustausch zwischen allen Entwicklungswerkzeugen basierend auf AML

Paradigma 2: Die Möglichkeit der Wiederverwendung einzelner Teilmodelle wie z.B. 3D-Modelle, Strukturplanungen oder Steuerungssoftware einzelner Komponenten

Paradigma 3: Unterstützung von ausführbaren Modellen, die eine frühe Offline-Simulation und anschließend die Hardware-in-the-loop (HIL) Simulation ermöglichen, um Steuerungssoftware testen zu können, bevor eine Anlage verfügbar ist

Paradigma 4: Eine systemweite Modellierung, die ein verteiltes Automatisierungssystem abbilden kann und nicht nur auf einzelne Komponenten beschränkt ist, um Fehler beim Zusammenwirken aller Komponenten frühzeitig erkennen zu können

Diese vier Haupteigenschaften zeigen sich in den unterschiedlichen Phasen des Entwicklungsprozesses von automatisierten Anlagen.

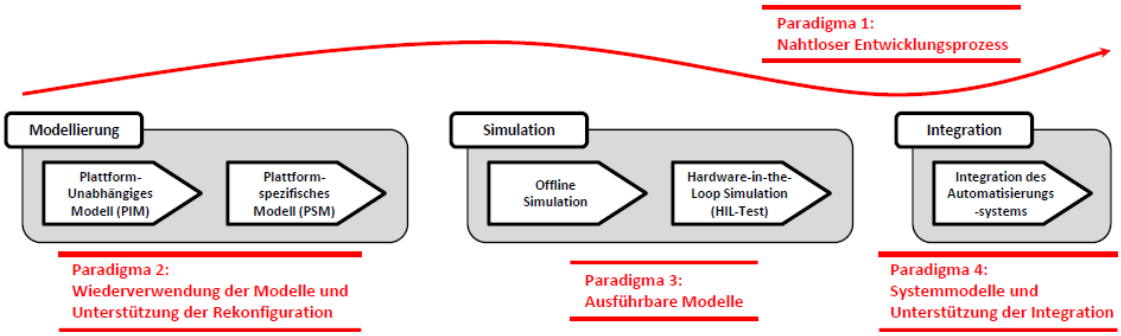

Bild 1: Moderner Entwicklungsprozess für Automatisierungssysteme

Im Weiteren bietet dieser Beitrag einen Überblick zum Stand der Technik aus dem Bereich der Entwicklung vom automatisierten Anlagen (Kapitel 2). Die Kapitel 3 bis 6 beschreiben die oben vorgestellten Paradigmen im Detail und orientieren sich dabei an der in Bild 1 gezeigten Reihenfolge. Kapitel 7 zeigt eine prototypische Werkzeugkette für den hier beschriebenen Entwicklungsprozess. Dieser Beitrag endet mit einer Zusammenfassung in Kapitel 7.

2. Stand der Technik

In einem modellbasierten Entwicklungsprozess werden Systemmodelle erstellt, die in den weiteren Entwicklungsphasen verwendet, weiterentwickelt, validiert und verifiziert werden können. Der modellbasierte Entwicklungsprozess wurde in [2] und [3] verwendet. In Letzterem wird das Konzept im Bereich der industriellen Automatisierung verwendet. Der modellbasierte Entwicklungsprozess von Automatisierungssystemen wurde in dem Projekt MODALE untersucht [4]. Das Projekt MEDEIA [5] beabsichtigt die Steigerung der Produktivität durch die modellbasierte Entwicklung eingebetteter Steuerungssysteme. Auch domainunabhängige Werkzeuge und Sprachen wie Matlab/Simulink, Modelica oder SysML werden für die Modellierung verschiedener Aspekte des Systems eingesetzt.

Da die Modelle oft von unterschiedlichen Werkzeugen verwendet werden, ist die Interoperabilität der Engineering-Werkzeuge von großer Bedeutung. So können für die Beschreibung der Anlagen unterschiedliche Datenformate benutzt werden, die die Datenkonsistenz und Interoperabilität zwischen den einzelnen Werkzeugen fördern. Häufig werden XML basierte Sprachen für die Modellierung verwendet [6].

Die Systemmodelle im Bereich der industriellen Automatisierungstechnik werden meist in einer Simulation ausgeführt und getestet. Es existieren verschiedene kommerzielle Werkzeuge für die Simulation. Zu erwähnen sind hier WinMod [7], SIMIT [8] und DELMIA [9]. Des

Weiteren existieren Werkzeuge für spezielle Aufgaben, wie z.B. die Programmierung von SPSen, die an den Standard 61131-3 angelehnt sind. Beispiele dafür sind CoDeSys, Step7 oder PCWORX. Neben der SPS-Programmierung existieren viele Werkzeuge für weitere Zwecke, wie z.B. HMI-Design oder Programmierung von Robotern. Eines der ersten Instrumente für einen integrierten Entwicklungsprozess ist der Siemens Automation Designer [10].
Es existieren viele Werkzeuge, die jeweils Teilbereiche der Entwicklung, wie Modellierung, Simulation oder Test, ermöglichen. Nur wenige Werkzeuge können den kompletten Entwicklungsprozess abdecken. Ein durchgängiger Entwicklungsprozess für Automatisierungssysteme ist nicht vorhanden. AML [11] scheint für den Entwurf der durchgängigen Werkzeugkette vielversprechend zu sein.

3. Paradigma 1: AutomationML für einen nahtlosen Entwicklungsprozess

Das Datenformat AML wird seit einigen Jahren mit dem Ziel entwickelt, den Datenaustausch zwischen Entwicklungswerkzeugen aus dem Bereich der Planung von Automatisierungsanlagen zu erleichtern. In der Vergangenheit war der Datenaustausch zwischen Planungssoftware der unterschiedlichen Gewerke nur schwer möglich. AML basiert auf CAEX und nutzt dieses XML-basierte Format um hierarchische Anlangenstrukturen zu speichern und weitere Datenformate bei Bedarf in das Modell zu integrieren. Dazu gehören Anlagentopologien (CAEX), 3D-Planungsinformationen (COLLADA) oder SPS-Programme (PLCopen XML), die zwischen unterschiedlichen Planungswerkzeugen ausgetauscht werden können. Innerhalb der hierarchischen CAEX-Struktur werden Komponenten wie Sensoren, Aktoren oder auch ganze Anlagenmodule abgelegt. Zusätzlich kann jede Komponente Attribute bekommen und mit anderen Komponenten über Interfaces verknüpft werden, um einen Daten- oder Materialaustausch abzubilden.
Um die in AML abgelegten Daten auch maschinenles- bzw. interpretierbar abzulegen, existieren eine *RoleClassLib* und eine *SystemUnitClassLib,* die den Komponenten eine Namensunabhängige Bedeutung zuordnen können. Die *RoleClassLib* bietet eine Bibliothek abstrakter Rollen für Komponenten (z.B. Antrieb, Förderband, Abstandssensor), um ihnen eine Bedeutung zuzuweisen. Eine detailliertere Beschreibung wird durch die *SystemUnitClassLib* bereitgestellt und erlaubt es, Komponenten eine eindeutige Produktbezeichnung zuzuweisen (z.B. Siemens S7-300).
Durch diesen hierarchischen Modellierungsansatz erlaubt AML einen durchgehenden Entwicklungsprozess: Es können verschiedene Aspekte aus der Anlagenplanung mit AML abgelegt werden und durch das Bibliothekskonzept kann die Anlagenmodellierung in mehreren Schritten von einem abstrakten zu einem sehr konkreten Modell erfolgen.

4. Paradigma 2: AutomationML zur Unterstützung von wiederverwendbaren Modellen

Modellierungkonzept

Der zweite Teil unseres Entwicklungsprozesses basiert auf wiederverwendbaren Modellen aus der Anlagenplanung. Dafür orientiert sich der Entwicklungsprozess an der Model-Driven-Architecture der Object Management Group (OMG) [12]. Ziel ist dabei eine Trennung von plattformunabhängigen (PIM) und plattformabhängigen (PSM) Modellen, wobei die Plattform die Zielplattform also Komponenten des Automatisierungssystems sind. Ein PIM umfasst z.B. die logische Architektur der Komponenten und die Softwaremodule, die für die Steuerung des Automatisierungssystems notwendig sind. Erst nachdem das PIM fertiggestellt ist, kann dieses Modell auf eine konkrete Plattform gebracht werden. Hierdurch entsteht ein plattformabhängiges Modell.

Durch die Trennung von PIM und PSM ergibt sich die Möglichkeit, PIM für mehrere Automatisierungssysteme verwenden zu können. Dies ist notwendig, da Anlagen(teile) im Bereich der Automatisierung häufig mehrfach in leicht abgewandelter Form (z.B. anderer Komponentenhersteller, anderes Bussystem usw.) verwendet werden.

Plattformunabhängiges Modell

Bei dem in Kapitel 1 beschriebenen Entwicklungsprozess ist das PIM-Modell der erste Schritt in der Anlagenplanung. Dieser Schritt gliedert sich in zwei Teilmodelle, Strukturmodell und Verhaltensmodell, die im Folgenden beschrieben werden:

Zu Beginn einer Anlagenplanung ist zunächst eine abstrakte Struktur des Automatisierungssystems festzulegen. Dies wird in einem Strukturmodell festgehalten, welches den Aufbau einer Anlage mit ihren unterschiedlichen Modulen beschreibt. Diese abstrakte Struktur kann später durch eine detailliertere Planung der einzelnen Module erweitert werden. Hierfür werden beispielsweise Sensoren und Aktoren in die einzelnen Module eingefügt. Um den einzelnen Komponenten ihre Funktion auch maschinenlesbar zuzuweisen, wird die *RoleClassLib* von AML verwendet.

Neben dem mechanischen Aufbau einer Anlage beinhaltet ein plattformunabhängiges Modell auch das Verhalten der vorhandenen Komponenten. Zu unterscheiden sind dabei zwei Arten von Verhaltensmodellen:

- **Sequencing**: Verhaltensmodelle für die Beschreibung des Verhaltens von Steuerungsapplikationen (SPS-Software)
- **Behavior**: Verhaltensmodelle für die Beschreibung des Verhaltens der Komponenten in Bezug auf ihre Physik

Um das Sequencing zu beschreiben bietet AML direkt die Möglichkeit an, Steuerungsalgorithmen in Form des PLCopen XML Standards zu integrieren. Dabei werden PLCopen XML Dateien aus dem AML-Modell referenziert und so in Verbindung mit einer Komponente gesetzt, um ihren Zweck beschreiben zu können. Gleiches gilt für die Modellierung des physikalischen Verhaltens einzelner Anlagenkomponenten (Behavior). Hierfür können Modelle für ganze Anlagenmodule oder auch nur für einzelne (Teil-)Komponenten erstellt und von der betreffenden Komponente aus dem AML-Strukturmodell referenziert werden.

Beide Arten der Verhaltensmodelle entstehen dabei mit externen Werkzeugen und aus diesen im entsprechenden Format exportiert. Das *Sequencing* wird in Werkzeugen zur SPS Programmierung (z.B. PCWorx) erzeugt, daraus exportiert und in das Anlagenmodell eingebunden.

Das Behavior-Modell entsteht hingegen mit Werkzeugen, die nach dem Modelica Standard [14] arbeiten. Dieser Standard stellt eine objektorientierte und standardisierte Sprache zur Modellierung komplexer Systeme bereit. Bekannt sind hier Werkzeuge wie SimulationX, AMESim oder Dymola. Bei der Modellierung kann hier auf eine umfangreiche Bibliothek von Komponenten (elektrische Komponenten, Zustandsautomaten, thermische Modelle usw.) und deren Verhalten zurückgegriffen werden. Dies bietet den Vorteil, dass weniger Detailwissen für die Modellierung von häufig sehr komplexen Komponenten wie Robotern oder elektrischen Antrieben notwendig ist.

Um diese Verhaltensmodelle in das Anlagenmodell zu integrieren, werden die ausführbaren Verhaltensmodelle aus den Werkzeugen als Functional-Mockup-Unit (FMU) exportiert. Die so exportieren Modelle sind ausführbar in einer entsprechenden Simulationsumgebung. Ihr Inhalt ist jedoch vor Zugriff geschützt, was einen Know-How-Schutz für die teilweise komplexen Modelle der Anlagenkomponenten bietet.

Bild 2 zeigt die Struktur eines AML Modells in Verbindung mit mehreren FMUs. Durch das Referenzieren einzelner FMUs von einer Komponente im Topologiemodell wird der FMU eine Bedeutung gegeben, da sie nur das Verhalten dieser einer Komponente modelliert [13]. So ist es möglich, das Verhalten einer ganzen Anlage aus den Verhaltensmodellen einzelner Teilkomponenten zu erstellen.

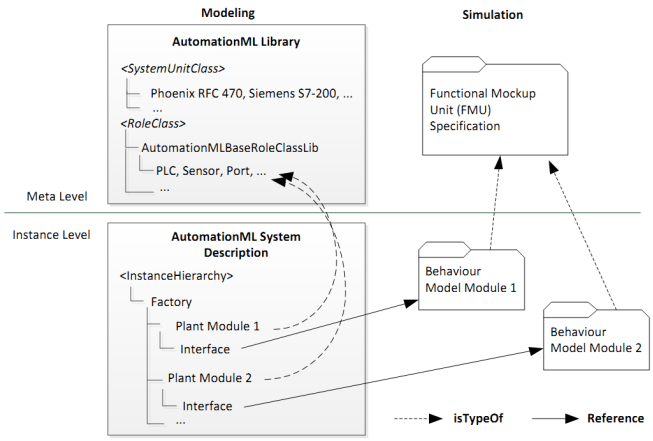

Bild 2: Aufbau eines AutomationML Modells mit extern referenzierten FMUs

Plattformabhängiges Modell

Mit einem plattformunabhängigen Modell ist zunächst die Struktur des Automatisierungssystems beschrieben. Um das System aufbauen zu können, ist eine Abbildung dieser Struktur auf eine konkrete Hardware notwendig. Dabei entsteht ein plattformabhängiges Modell. Ohne diese Abbildung ist nicht klar, welche Komponente eine im plattformunabhängigen Modell beschriebene Aufgabe erfüllen soll. Um diese zusätzliche Information hinzuzufügen, bietet AML die *SystemUnitClassLib* (vgl. Kapitel 3). Jedes Element des AML-Modells kann durch Referenzieren eines der Elemente der *SystemUnitClassLib* auf eine konkrete Hardware abgebildet werden. Dadurch entsteht ein plattformabhängiges Modell der Anlage.

5. Paradigma 3: AutomationML für die Simulation

Durch die Integration von ausführbaren Verhaltensmodellen im Gesamtmodell ist ein früher Test des geplanten Automatisierungssystems möglich. Hierfür haben wir ein komponentenorientierts Simulationsframework erstellt, das in Bild 3 dargestellt ist. Aus dem AML basierten Systemmodell werden die verfügbaren Simulationskomponenten (SPS-Programme und Verhaltensmodelle) in das Simulationsframework übertragen. Die Simulation des Gesamtverhaltens erfolgt dann wie folgt:

- Jede Komponente hat einen Satz von Eingangs- und Ausgangsdaten, die durch Datentypen wie Integer oder Boolean beschrieben sind.
- Jede Komponente teilt dem Simulationsframework mit, wenn neue Ausgangsdaten bereitstehen.

- Die Ausgangsdaten werden von den Komponenten abgeholt und an die Komponenten verteilt, die diese Daten als Eingangsdaten verwenden. Diese Verbindungen zwischen den Komponenten sind im AML-Modell definiert.
- Zum Test der Komponenten wird bei jeder Simulation ein Testfall in das Simulationsframework eingegeben. Jeder Testfall beschreibt zusätzlich Signale, die zu definierten Zeitpunkten gesetzt werden sowie ein Testergebnis, das zu einem definierten Zeitpunkt in Abhängigkeit der aktuellen Systemvariablen erzeugt wird.

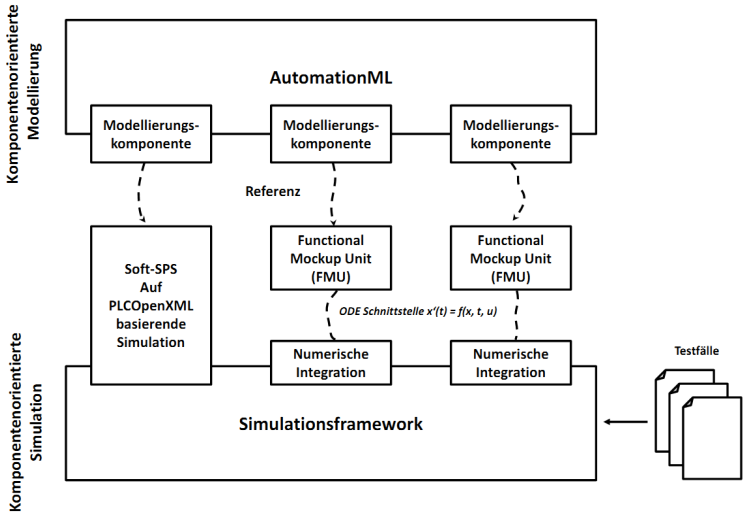

Bild 3: Komponentenorientierte Modelle in einer komponentenorientierten Simulationsumgebung

Bisher unterstützt das Simulationsframework SPS-Software in Form von PLCopen XML und FMUs als Verhaltensmodelle der Anlage. Die FMUs verfügen über eine Schnittstelle, die das modellierte System als eine Menge von Differentialgleichungen beschreiben und über die über C-Funktionen zugegriffen werden kann. Um das Modell ausführen zu können, wird ein numerischer Integrator verwendet, der aus den Eingangsdaten neue Ausgangsdaten mit den Regeln des Modells erzeugt. Neben der Simulation von Steuerungssoftware und Verhaltensmodellen (Offline-Simulation) unterstützt das Simulationsframework auch die Realtime-Simulation, bei der eine reale SPS über ein Bussystem an das Simulationsframework angebunden wird. Dieses bildet bei der Realtime-Simulation lediglich das Verhalten der Umgebung nach. Beide Szenarien sind im Folgenden beschrieben:

Offline-Simulation

Bei der Offline-Simulation werden sowohl SPS-Software als auch Verhaltensmodelle simuliert. Ein Beispiel ist in Bild 4 dargestellt. Komponente 1 beinhaltet das SPS-Programm, die beiden anderen Komponenten simulieren die Umgebung.

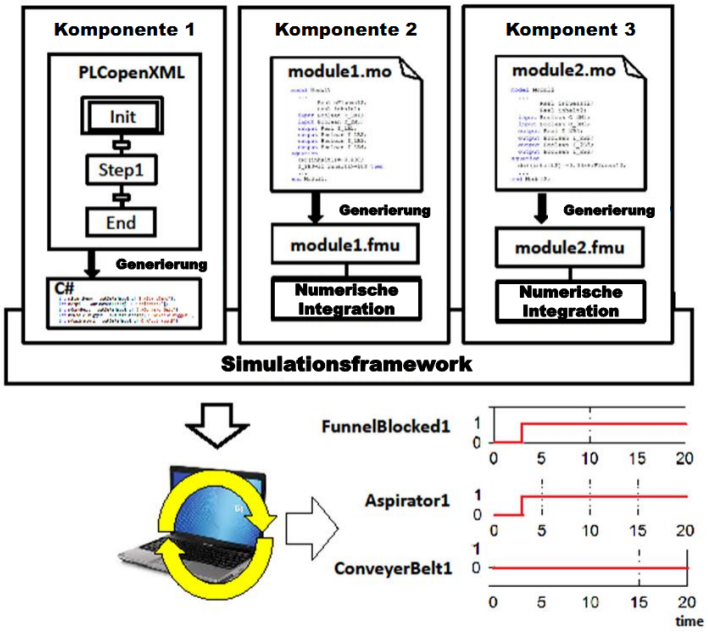

Bild 4: Offline-Simulation

Bei dem hier gezeigten Beispiel handelt es sich um die Simulation des in Bild 6 skizzierten Systems. Material wird aus einem Behälter mit einem Förderband in einen Trichter transportiert. Dabei darf das Förderband nur dann anlaufen, wenn der Trichter nicht blockiert ist. Das Verhalten des SPS-Programms wird für diese Situation getestet. Da der Trichter verstopft ist, läuft das Förderband nicht an. Die SPS-Software verhält sich somit in Bezug auf diesen Testfall korrekt.

Realtime-Simulation

Bei der Realtime-Simulation wird die simulierte Steuerung durch einen Hardware-Attachment-Point ersetzt (vgl. Bild 5). An diesen wird eine echte Steuerung über ein Bussystem, hier PROFINET, angeschlossen. Bei dem gezeigten Beispiel handelt es sich um das gleich Szenario wie bei der Offline-Simulation. Hier ergibt die Simulation jedoch ein anderes Ergebnis, das Förderband läuft trotz blockiertem Trichter an.

Da sich die Steuerungssoftware in der Offline-Simulation in Bezug auf das gezeigte Szenario als fehlerfrei erwiesen hat, muss der Fehler in diesem Fall im Kommunikationssystem liegen. Es könnte beispielsweise eine Fehlkonfiguration vorliegen.

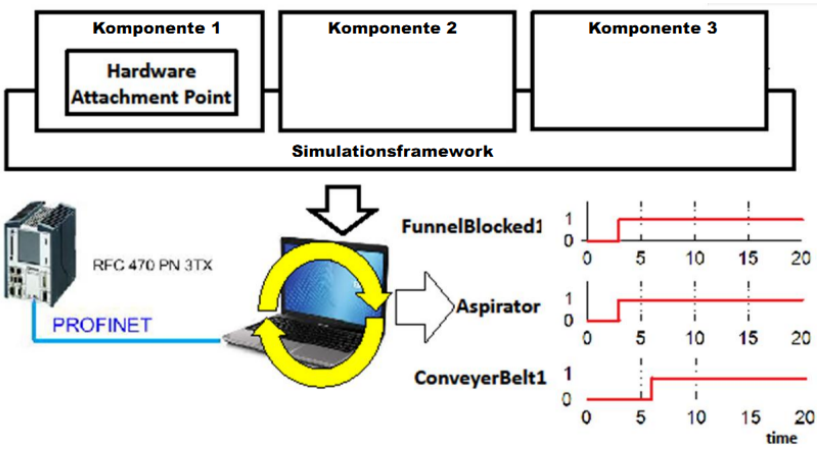

Bild 5: Realtime-Simulation

6. Paradigma 4: AutomationML für die Systemintegration

Der letzte Schritt des bereits beschriebenen Entwicklungsprozesses besteht in der Integration der modellierten Anlage in ein reales System. Das fehlerfreie Modell beinhaltet nun Planungsdaten, die für eine Integration der realen Automatisierungskomponenten verwendet werden können. Dafür müssen die Informationen aus dem Modell extrahiert und in die domainspezifischen Werkzeuge importiert werden. Ein Beispiel hierfür ist die Verwendung der Steuerungssoftware aus dem Modell in einer realen Anlage. Zusätzlich können beispielsweise Informationen zur Visualisierung der Anlage aus dem Topologiemodell extrahiert werden.

7. Prototypische Werkzeugkette

Erstellung von plattformunabhängigen Strukturmodellen

Um die in Kapitel 5 beschriebenen Strukturmodelle zu erstellen, ist ein Editor notwendig, der diese als AML-Datei exportieren kann. Einen solchen Editor haben wir auf Basis des Visual Studio 2010 Visualization and Modeling SDK erstellt. Mit diesem Editor ist es möglich, die Struktur einer Anlage durch die Anordnung grafischer Komponenten zu modellieren (vgl. Bild 6). Durch diesen Editor kann zunächst ein PIM erstellt werden. Hierfür werden Komponenten, die Elemente aus der *RoleClassLib* (vgl. Kapitel 4) darstellen, im Editor so angeordnet, dass sie der beabsichtigten Anlagenstruktur entsprechen. Beim Export werden diese Rolleninformationen in AML abgelegt.

Erstellung von plattformunabhängigen Verhaltensmodellen

Wie bereits beschrieben werden Verhaltensmodelle als FMU in das Anlagenmodell integriert. Daher können die Verhaltensmodelle mit allen Werkzeugen erstellt werden, die einen Export

der Modelle als FMU unterstützen. Bei dem in Bild 6 gezeigten Beispiel wurden die Verhaltensmodelle für ganze Module der Anlage mit Dymola erstellt. Es ist jedoch auch möglich, nur Teile eines Moduls (z.B. Förderband und Lagerbehälter) zu modellieren und diese Modelle in das Gesamtmodell zu integrieren.

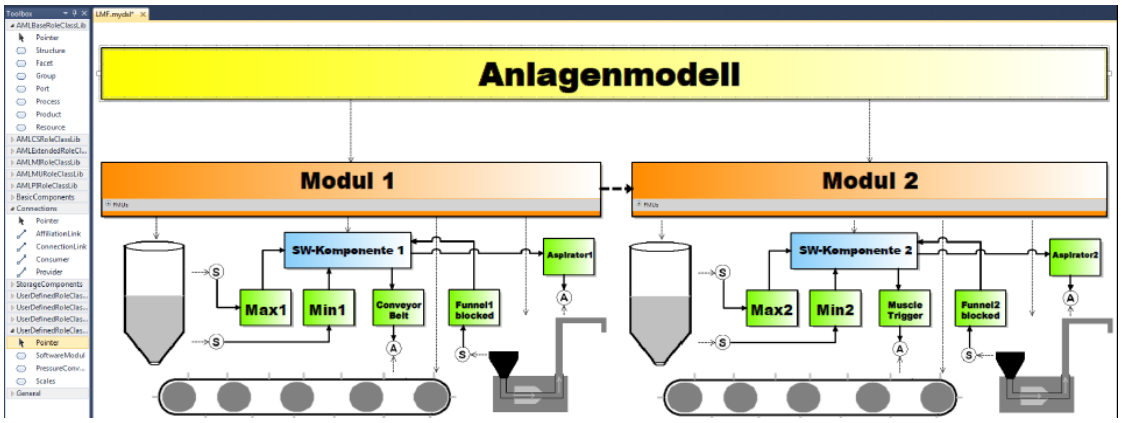

Bild 6: Editor für PIM und PSM

Erstellung von plattformabhängigen Modellen

Wie in Kapitel 4 beschrieben entsteht ein plattformabhängiges Modell durch Referenzieren eines Elements der *SystemUnitClassLib* aus dem Strukturmodell in AML. Dieser Vorgang ist auch mit dem in Bild 6 gezeigten Editor möglich. Dazu werden die zur Verfügung stehenden SystemUnitClass-Bibliotheken geladen und können dann mit den bereits geplanten Anlagenkomponenten verknüpft werden.

8. Zusammenfassung und Ausblick

In diesem Beitrag wurde ein auf AML basierter Entwicklungsprozess dargestellt, der dem MDA-Prinzip der OMG folgt. Dabei entstehen wiederverwendbare und simulationsfähige Anlagenmodelle, die für einen frühen Test von Steuerungssoftware automatisierter Produktionsanlagen geeignet sind. Aufbauend auf einem komponentenbasierten Modellierungsansatz entsteht das Systemmodell. Dieses beinhaltet u.a. die Anlagentopologie und ausführbare Verhaltensmodelle. Diese Verhaltensmodelle der einzelnen Komponenten werden für den Online- und Offline-Test verwendet, um das Gesamtsystem zu testen. Für einen solchen Test wird die noch nicht vorhandene Anlage durch ein ausführbares Verhaltensmodell ersetzt und dieses als virtuelle Testumgebung für die Steuerungssoftware verwendet.

Aus dem getesteten Systemmodell können die Planungsdaten extrahiert und mit domainspezifischen Werkzeugen in eine reale Anlage integriert werden.
Zukünftige Arbeiten beziehen sich auf die detailliertere Modellierung des Automatisierungssystems (Bussysteme, Merkmale von IO-Systemen usw.) im Modellierungseditor.

Danksagung

Die Forschung zu dieser Veröffentlichung wird im Rahmen des Projekts inITial durch das Ministerium für Innovation, Wissenschaft und Forschung des Landes Nordrhein-Westfalen gefördert.

Quellen

[1] R. Drath, D. Weidemann, S. Lips, L. Hundt, A. Lüder and M. Schleipen, *Datenaustausch in der Anlagenplanung mit AutomationML*, Springer 2010

[2] S. Stein, S. Kühne, Jens Drawehn, Sven Feja and Werner Rotzoll, *Evaluation of OrViA Framework for Model-Driven SOA Implementations: An Industrial Case Study*, 6th International Conference on Business Process Management, 2008, Milan, Italy

[3] T. Strasser, C. Sünder and A. Valentini, *Model-Driven Embedded Systems Design Environment for the Industrial Automation Sector*, Proceedings of the 6th IEEE International Conference on Industrial Informatics, 2008, Daejeon, South Korea

[4] MODALE Project, *Modellbasiertes Anlagen-Engineering, kundenorientierte Dienstleistungen für Anlagensteuerung- und Kontrolle*, 2005, www.modale.org/

[5] MEDEIA Project, *Overview of the MEDEIA Project*, 2011, www.medeia.eu

[6] M. Wollschlaeger, L. Lindemann, S. Runde und M. Mühlhause: *XML in der Automation – Systematisches Sprachdesign*, EKA 2010, Magdeburg

[7] Mewes & Partner, WinMod, 2012, www.winmod.de

[8] Siemens, SMIT, 2012, www.industry.siemens.com

[9] Dassault Systems, *DELMIA Digital Manufacturing & Production*, 2012, www.3ds.com

[10] Siemens, Automation Designer, www.automation.siemens.com, 2012

[11] AutomationML, Spezifikation V1.1, www.automationml.org, 2009

[12] S. Mellor, K. Scott, A. Uhl und D. Weise, *MDA Distilled: Principles of Model-Driven Architecture*. Addison Wesley, 2004

[13] N. Moriz, S. Faltinski, O. Gräser, O. Niggemann, M. Barth, A. Fay; *Integration und Anwendung von objektorientierten Simulationsmodellen in AutomationML*, Automation 2011 Baden Baden

[14] Modelica Association, www.modelica.org, 2011