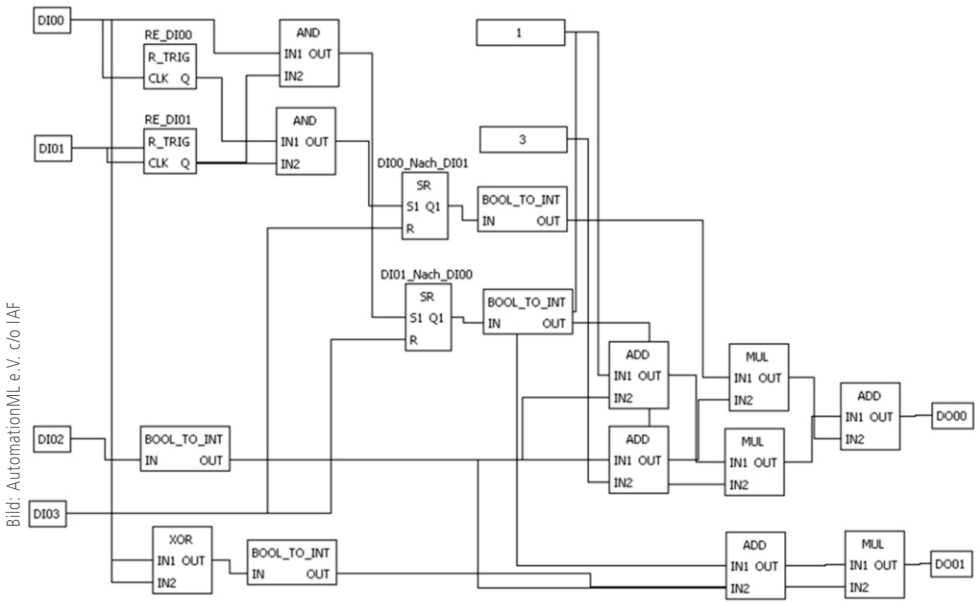

Bild: AutomationML e.V. c/o IAF

Bild 1: Steuerung des Antriebs im Funktionsblockdiagramm nach IEC61131-3

AutomationML: Verhältnismäßig gut!

Serie AutomationML Teil 9: Eignet sich AutomationML als neutrale Sprache zur Beschreibung von Verhalten?

Die virtuelle Planung von Fertigungsanlagen gewinnt immer mehr an Bedeutung. Die Logik einzelner Komponenten sowie vollständige Anlagenabläufe inklusive aller möglichen Sonderfunktionen sollen im Vorfeld ausgiebig getestet werden können, noch vor der Installation auf der Baustelle. Dazu gehört beispielsweise auch das Verhalten von Antrieben für Förderer, welches dann mithilfe von gängigen Werkzeugen, für die WinMOD von Mewes & Partner oder Tecnomatix Process Simulate von Siemens PLM als Beispiele genannt werden könnten, simuliert werden kann.

Voraussetzung für die Simulation ist das Vorhandensein entsprechender simulierbarer Modelle, die neben der Geometrie und Kinematik auch das Verhalten der Systemkomponenten beschreiben. Die Anbieter von Systemkomponenten geben derzeit das Verhalten ihrer Komponenten in einem Handbuch mit oder sie bieten spezifische Module für die einzelnen Simulationswerkzeuge an. Die Möglichkeit, das Verhalten in einer neutralen Sprache zu beschreiben, würde sie unabhängig von dem vom Auftraggeber genutzten Simulationswerkzeug machen, vorausgesetzt eine Überführung in dieses Werkzeug ist möglich. Zudem können Bibliotheken von Systemkomponenten entstehen, die für verschie-

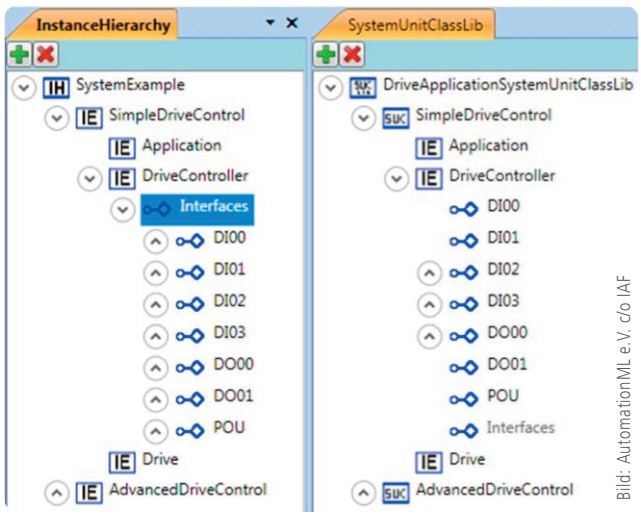

Bild: AutomationML e.V. c/o IAF

Bild 2: Komponentenbeispiel

```
<InternalElement Name="DriveController" ID="{397d601e-3300-473a-a6b0-3d8b6a28d5c7}">
  <ExternalInterface Name="DI00" RefBaseClassPath="AutomationMLInterfaceClassLib/AutomationMLBaseInterface/ExternalDataConnector/PLCopenXMLInterface/VariableInterface" ID="{291
    <Attribute Name="refURI" AttributeDataType="xs:anyURI">
      <Value>//simpledrivecontrol.xml#DI00</Value>
    </Attribute>
  </ExternalInterface>
  <ExternalInterface Name="DI01" RefBaseClassPath="AutomationMLInterfaceClassLib/AutomationMLBaseInterface/ExternalDataConnector/PLCopenXMLInterface/VariableInterface" ID="{915
    <Attribute Name="refURI" AttributeDataType="xs:anyURI">
      <Value>//simpledrivecontrol.xml#DI01</Value>
    </Attribute>
  </ExternalInterface>
  <ExternalInterface Name="DI02" RefBaseClassPath="AutomationMLInterfaceClassLib/AutomationMLBaseInterface/ExternalDataConnector/PLCopenXMLInterface/VariableInterface" ID="{162
  <ExternalInterface Name="DI03" RefBaseClassPath="AutomationMLInterfaceClassLib/AutomationMLBaseInterface/ExternalDataConnector/PLCopenXMLInterface/VariableInterface" ID="{3d4
  <ExternalInterface Name="DO00" RefBaseClassPath="AutomationMLInterfaceClassLib/AutomationMLBaseInterface/ExternalDataConnector/PLCopenXMLInterface/VariableInterface" ID="{0b
  <ExternalInterface Name="DO01" RefBaseClassPath="AutomationMLInterfaceClassLib/AutomationMLBaseInterface/ExternalDataConnector/PLCopenXMLInterface/VariableInterface" ID="{b8
  <ExternalInterface Name="POU" RefBaseClassPath="AutomationMLInterfaceClassLib/AutomationMLBaseInterface/ExternalDataConnector/PLCopenXMLInterface/LogicInterface" ID="{2bf24
    <Attribute Name="refURI" AttributeDataType="xs:anyURI">
      <Value>//simpledrivecontrol.xml</Value>
    </Attribute>
  </ExternalInterface>
</InternalElement>
```

Bild: AutomationML e.V. c/o IAF

Bild 3: Beispielreferenzen auf PLCopen XML-Dateien

dene Simulationsziele unterschiedliche Modelle der Systemkomponenten bereitstellen, die der Wissenssicherung dienen und die Wiederverwendbarkeit dieser Systemkomponenten vereinfachen. Derartige Bibliotheken könnten aus unterschiedlichen Quellen gespeist werden. Kandidaten dafür sind neben den Systemintegratoren auch Komponenten- und Gerätelieferanten. Dieser Artikel soll zeigen, dass AutomationML sowohl zum neutralen Austausch von Verhaltensinformationen geeignet ist als auch die Voraussetzung für eine Bibliotheksbildung erfüllt.

Ein Beispiel

Für ein besseres Verständnis soll an dieser Stelle ein Antriebsstrang betrachtet werden, wie er in Teil 5 dieser Artikelserie bereits vorgestellt wurde. Dieser soll einen Antrieb mit einer vereinfachten Antriebssteuerung enthalten. Dieser Antrieb kann sowohl im Rechts- als auch im Linkslauf betrieben werden und unterstützt zwei Geschwindigkeiten. Dabei darf er aus dem Stillstand in den Links- oder Rechtslauf gestartet werden. Zwischen den beiden Geschwindigkeiten darf innerhalb derselben Laufrichtung jederzeit gewechselt werden. Um den Antrieb allerdings aus der einen Laufrichtung in die andere zu schalten, muss er zunächst angehalten werden. Wird versucht den Antrieb vom Rechts- direkt in den Linkslauf zu schalten (oder umgekehrt), wird dies ignoriert und ein Fehlerstatus gesetzt, der zurückgesetzt werden kann. Hieraus ergeben sich die Eingänge der Steuerung, wie sie in Bild 6 zusammengefasst sind. Während des Betriebs sollen von der Antriebssteuerung zwei Informationen abgefragt werden können. Zum einen soll die aktuelle Geschwindigkeit und zum anderen der aktuelle Fehlerstatus ausgelesen werden können. Die sich daraus ergebenden Ausgänge der Steuerung sind in Bild 7 zusammengefasst. Das Verhalten dieser Steuerung lässt sich nach IEC61131-3 als Funktionsblockdiagramm (FBD) beschreiben und ist in Bild 1 zu sehen.

Herstellerunabhängige Komponentenbeschreibung

Für eine hersteller- und technologieunabhängige Beschreibung des Antriebstrangs kann AutomationML eingesetzt werden. Dazu werden CAEX zur Beschreibung der Komponentenstruktur und dazugehörige Schnittstellen und PLCopen XML zur Beschreibung des Komponentenverhaltens verwendet. In CAEX werden, wie in Bild 2 zu sehen, die einzelnen Geräte zu einer Gerätestruktur als Hierarchie von InternalElements zusammengefasst und bilden damit eine Komponente. In Bild 2 sind beispielhaft die Antriebsstränge SimpleDriveControl und AdvancedDriveControl gezeigt, die jeweils eine Komponente bilden und einen Antrieb, eine An-

```
<body>
  <FBD>
    <inVariable localId="1" height="23" width="33">
    <inVariable localId="2" height="23" width="33">
    <inVariable localId="3" height="23" width="33">
    <inVariable localId="4" height="23" width="33">
    <outVariable localId="5" height="23" width="37">
    <outVariable localId="6" height="23" width="37">
    <block localId="7" width="47" height="40" typeName="R_TRIG" instanceName="RE_DI01">
    <block localId="8" width="53" height="60" typeName="AND">
      <position x="270" y="38" />
      <inputVariables>
        <variable formalParameter="IN1">
        <variable formalParameter="IN2">
      </inputVariables>
      <inOutVariables/>
      <outputVariables>
        <variable formalParameter="OUT">
      </outputVariables>
    </block>
    <block localId="9" width="47" height="40" typeName="R_TRIG" instanceName="RE_DI00">
    <block localId="10" width="53" height="60" typeName="AND">
    <block localId="11" width="41" height="60" typeName="SR" instanceName="DI01_Nach_DI00">
    <block localId="12" width="41" height="60" typeName="SR" instanceName="DI00_Nach_DI01">
    <block localId="13" width="53" height="60" typeName="MUL">
    <inVariable localId="15" height="23" width="79">
    <inVariable localId="16" height="23" width="78">
    <block localId="17" width="53" height="60" typeName="ADD">
    <block localId="18" width="80" height="40" typeName="BOOL_TO_INT">
    <block localId="19" width="80" height="40" typeName="BOOL_TO_INT">
    <block localId="20" width="53" height="60" typeName="ADD">
    <block localId="21" width="53" height="60" typeName="MUL">
    <block localId="14" width="80" height="40" typeName="BOOL_TO_INT">
    <block localId="22" width="53" height="60" typeName="ADD">
    <block localId="23" width="53" height="60" typeName="XOR">
    <block localId="24" width="80" height="40" typeName="BOOL_TO_INT">
    <block localId="25" width="53" height="60" typeName="MUL">
    <block localId="27" width="53" height="60" typeName="ADD">
  </FBD>
</body>
```

Bild: AutomationML e.V. c/o IAF

Bild 4: FDB in PLCopen XML

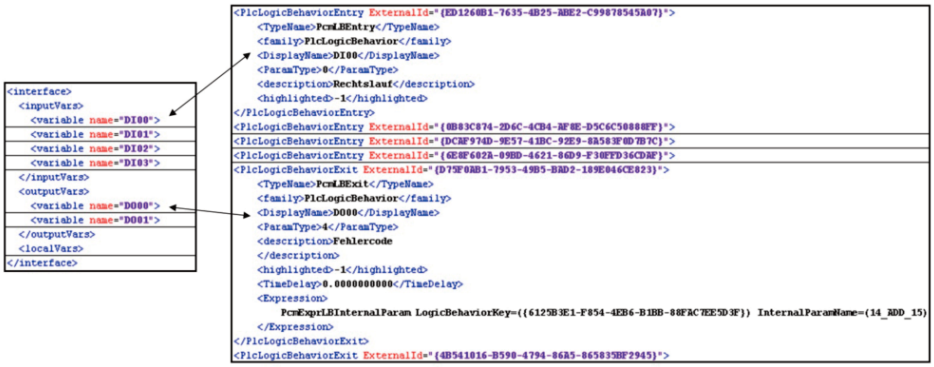

Bild: AutomationML e.V. c/o IAF

Bild 5: Analogie zwischen PLCopen XML und Process Simulate

triebssteuerung und die Applikationsbeschreibung enthalten. Die einzelnen InternalElements der Hierarchie können notwendige Eigenschaften als Attribute enthalten. Zudem wird aus CAEX heraus auf PLCopen XML Dateien verwiesen, um darüber die Verhaltensbeschreibungen einzubinden. Wie Bild 3 verdeutlicht, dienen dazu entsprechende ExternalInterface Elemente, die auf Objekte (ganze POEs oder einzelne Variablen) in den PLCopen XML Dateien verweisen.

PLCopen XML

Das FBD ist eine der fünf in der IEC61131-3 definierten Sprachen zur Programmierung von Speicherprogrammierbaren Steuerungen (SPS). PLCopen XML wurde entwickelt, um die Steuerungslogik gemäß IEC61131-3 in einem herstellerunabhängigen Format abbilden zu können. Ähnlich wie in der grafischen Repräsentation in Bild 1 wird in PLCopen XML jeder einzelne Funktionsbaustein als eigenes Element beschrieben. Die Eingangs- bzw. Ausgangsvariablen werden durch die Elemente <inVariable> und <outVariable> modelliert und die einzelnen Funktionsblöcke, die nach IEC61131-3 standardisiert sind, durch die Elemente <block> dargestellt. Die einzelnen Funktionsblöcke und Variablen werden nun über sogenannte Konnektoren miteinander verbunden, wodurch sich die Antriebssteuerung vollständig in der neutralen Sprache PLCopen XML modellieren lässt. Ein Ausschnitt aus der resultierenden Beschreibung ist in Bild 4 zu sehen. Um festzustellen, ob diese Darstellung auch für den herstellerunabhängigen Datenaustausch geeignet ist, muss untersucht werden, wie die gleiche Steuerung in einem Simulationssystem zu modellieren und ob eine automatische Überführung der Darstellungen möglich ist.

Process Simulate

Für diese Untersuchungen soll das Simulationssystem Process Simulate beispielhaft herangezogen werden. Es ist ein Repräsentant einer Werkzeugklasse, für die nach Ansicht der Autoren die nachfolgenden Aussagen ebenso zutreffen. Für die Beschreibung einer Steuerung steht im Simulationssystem Process Simulate ein Editor zur Verfügung, mit dessen Hilfe sich Eingänge, Ausgänge und Parameter definieren lassen. Eingängen und Ausgängen können Datentypen zugewiesen werden. Parameter definieren Funktionen, die aus n Variablen ein Ergebnis ermitteln. Als Variablen dienen entweder die Eingänge oder die Ergebnisse von anderen Parametern. Als Funktionen stehen neben arithmetischen und booleschen Operationen auch einige vordefinierte Funktionen wie z.B. 'RisingEdge', 'FallingEdge' oder 'SetReset' zur Verfügung. Diese stellen ein Subset der in der IEC61131-3 standardisierten FBD Bausteine zur Beschreibung von Steuerungsverhalten dar. Gespeichert wird das Verhalten in einer XML Datei. Ein Blick in die erzeugte Datei (siehe Bild 5) zeigt, dass hier die Eingänge und Ausgänge sowie die einzelnen Funktionsblöcke in Listen gespeichert und über ihre IDs miteinander verknüpft werden.

Beliebig austauschbar!

Im direkten Vergleich ist ein analoger Aufbau der beiden XML Formate zu erkennen.

- Die Daten werden in Eingänge, Ausgänge und Funktionen bzw. FBD Bausteinen gruppiert.
- Eingänge sind die Variablen der Funktionen.
- Das Ergebnis einer Funktion ist entweder eine Variable für eine folgende Funktion oder der Wert eines Ausgangs.
- Alle Funktionen entsprechen denen in der IEC61131-3 beschriebenen.

Name des Eingangs	Typ	Bedeutung
DI00	Boolean	True: Rechtslauf starten False: Rechtslauf anhalten
DI01	Boolean	True: Linkslauf starten False: Linkslauf anhalten
DI02	Boolean	False: Betrieb mit Geschwindigkeit 1 True: Betrieb mit Geschwindigkeit 2
DI03	Boolean	Fehlerstatus zurücksetzen

Bild: AutomationML e.V. c/o IAF

Bild 6: Eingänge der Antriebssteuerung

Der Informationsgehalt in beiden Formaten ist somit identisch. Da es sich bei beiden Formaten um XML Dokumente handelt, wäre eine Überführung im einfachsten Fall durch eine XML Transformation mittels XSLT möglich. Process Simulate bietet zwar nur ein Subset der in der IEC61131-3 beschriebenen Funktionen an, allerdings steht es dem Anwender frei über eine Plugin-Schnittstelle weitere Funktionen im Editor zur Verfügung zu stellen.

Aber sicher!

Dem Bestreben Daten herstellerneutral abzubilden steht auch immer der Wunsch gegenüber, den Inhalt vor dem Zugriff Unbefugter und vor nicht intendierter Veränderung abzusichern. In diesem Zusammenhang stellen sich im Wesentlichen zwei Fragen:

- Von wem sind die Daten und wurden sie auch nicht verändert?
- Wer darf die Daten sehen?

Die erste Frage beschäftigt sich also mit der Authentizität und der Integrität der Daten und die zweite mit dem Schutz vor Missbrauch. Beide Fragen sind so alt wie ihre Lösungsansätze. Authentizität und Integrität kann durch eine digitale Signatur gewährleistet und der Schutz vor Missbrauch kann durch Verschlüsselung erreicht werden. Welcher Standard dafür genutzt wird, sei es PKCS oder XML Signature/Encryption, ist reine Geschmackssache. Vielmehr sollte klar sein, welche Folgen die Nutzung von Verschlüsslungen bzw. Signaturen haben könnten. Eine digitale Signatur ist in jedem Fall sinnvoll. Die Daten sind lesbar, die Integrität kann jederzeit überprüft werden und die Entscheidung, ob der Signatur vertraut wird, kann frei getroffen werden. Sind die Daten hingegen (auch) verschlüsselt, so kann nur der vorgesehene Empfänger die Daten sehen. Wird eine Verschlüsselung als Transportsicherung für eine Übermittlung beispielsweise per Mail verwendet oder dient sie dem Wissensschutz bei einer Langzeitarchivierung, so ist dies mit Sicherheit sinnvoll. Hier muss jedoch für jeden Anwendungsfall genau überprüft werden, wer wann welche der verschlüsselten Informationen lesen bzw. verändern darf und wie ihm die dafür notwendigen Schlüssel zur Verfügung gestellt werden können. Insbesondere gilt dies, wenn der Empfänger ein Programm ist, das die Daten weiterverarbeitet. Hier muss durch alle Beteiligten sichergestellt sein, dass der Ansatz eines herstellerneutralen Datenformates nicht ad absurdum geführt wird. Für beide Anwendungsfälle, Verschlüsslungen und Signatur, sind für die Verarbeitung von XML Dateien entsprechende frei verfügbare Softwarekomponenten vorhanden, so dass ihre Anwendung nur vor organisatorischen, nicht jedoch vor technischen Herausforderungen steht.

Und nun?

Fasst man die beschriebenen Strukturen zusammen, so können herstellerunabhängige Bibliotheken wiederverwendbarer Systemkomponenten entstehen, die passend zu unterschiedlichen Anwendungsfällen Verhaltensmodelle enthalten. Diese Modelle können, wie erste Entwicklungsansätze zeigen, automatisch zur Erstellung von simulierbaren Anlagenmodellen kombiniert werden. Für deren Simulation könnte dann entweder auf bestehende Werkzeuge zurückgegriffen oder neue Simulationswerkzeuge unter Anwendung von IEC61131-3 basierten Soft-SPSen entwickelt werden. In beiden Fällen können insbesondere Komponentenlieferanten und Systemintegratoren profitieren. Erstere können sich durch die Bereitstellung von Simulationsmodellen einen Marktvorteil sichern und letztere können Wissen sichern und wiederverwenden und damit ihre Entwurfsqualität deutlich erhöhen. ■

Name des Ausgangs	Typ	Bedeutung
DO00	Integer	Abfrage Fehlercode: 0: kein Fehler 1: versuchte Änderung vom langsamen Rechtslauf in Linkslauf 2: versuchte Änderung vom schnellen Rechtslauf in Linkslauf 3: versuchte Änderung vom langsamen Linkslauf in Rechtslauf 4: versuchte Änderung vom schnellen Linkslauf in Rechtslauf
DO01	Integer	Abfrage Geschwindigkeit: 0: Motor angehalten 1: Antrieb im Betrieb Geschwindigkeit 1 2: Antrieb im Betrieb Geschwindigkeit 2

Bild: AutomationML e.V. c/o IAF

Bild 7: Ausgänge der Antriebssteuerung

www.automationml.org

Autor: Steffen Lips, Projektleiter, NetAllied Systems GmbH