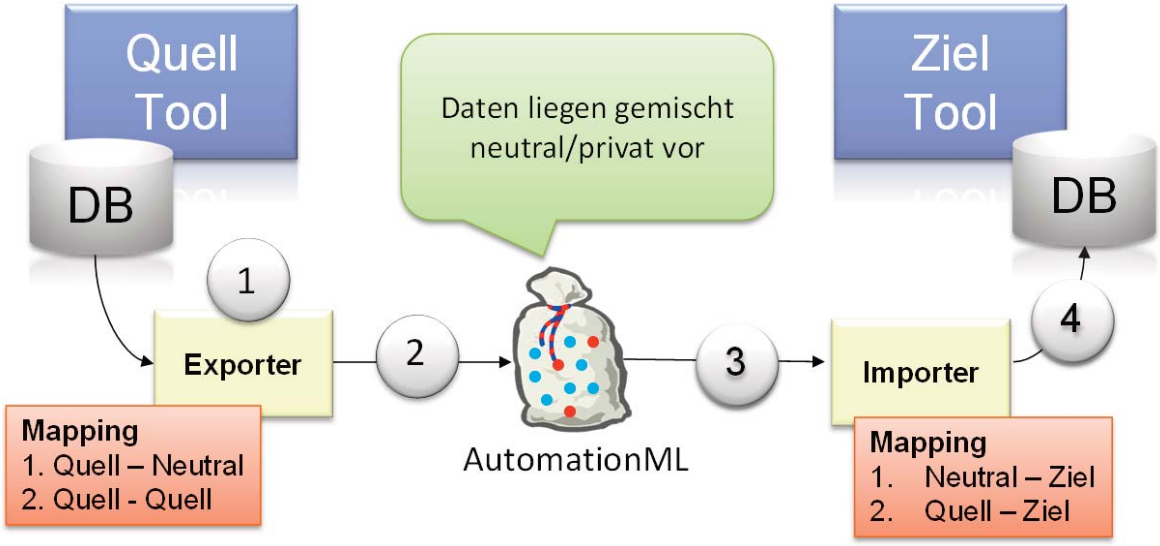

Bild 1: Realer Datenaustausch mit CAEX

AutomationML-Programmierung

Serie AutomationML Teil 3: Effizientes Programmieren mit der AutomationML-Engine

Ziel von AutomationML [1] ist der Datenaustausch zwischen Werkzeugen der Anlagenplanung. Als herstellerneutrales, standardisiertes und XML-basiertes Datenformat kann es derzeit Planungsinformationen bezüglich Anlagentopologie/Struktur, Geometrie/Kinematik sowie Logik/Verhalten abbilden. Dabei bildet CAEX [2] das Objektmodell der Engineeringdaten ab. Die Integration von AutomationML in die Schnittstellen von Werkzeugen wirft jedoch immer wieder die Frage auf: Wie aufwändig ist die Programmierung von Ex- und Importern? Dieser Beitrag beleuchtet dies anhand verschiedener Programmierszenarien.

Parallel zu AutomationML entwickelte der AutomationML e.V. eine freie Software für Entwickler – die AutomationML-Engine [3]. Sie spiegelt das CAEX-Datenmodell in Form einer C#-Klassenstruktur exakt wider und enthält alle Klassen und Methoden, um CAEX-Objekte (Klassen und Instanzen) manipulieren zu können. Der Softwareentwickler wird durch diese Software befreit, die Einhaltung des CAEX-Schemas zu überwachen. Stattdessen operiert er auf Klassenebene, während die Engine im Hintergrund konforme CAEX-Dateien erzeugt oder liest. Entwicklungsvoraussetzung zur Nutzung der AutomationML-Engine ist das Betriebssystem WindowsXP oder höher sowie Visual Studio Version 2008 oder höher. Die Einbindung der Engine in eigene Projekte erfolgt über die Referenzierung der Engine-dlls.

AutomationML-Programmierung

Bild 1 zeigt das Basisszenario: Der Datenaustausch zwischen zwei Werkzeugen benötigt einen Exporter und einen Importer. Sowohl Exporter als auch Importer müssen AutomationML-Dateien erzeugen, lesen und manipulieren können. Daraus lassen sich Basisfunktionen und erweiterte Funktionen ableiten, die im Folgenden schrittweise beleuchtet werden.

CAEX-Basisprogrammierung

1. Erzeugen eines CAEX-Dokumentes

Für das Erzeugen eines leeren CAEX-Dokumentes genügt eine Zeile Quelltext (siehe Bild 2). Im Hintergrund, für den Anwender verborgen, erzeugt die AutomationML-Engine alle benötigten XML-Knoten [4].

2. Öffnen/Speichern einer CAEX-Datei

Das Öffnen und Speichern eines CAEX-Dokumentes erfordert ebenfalls jeweils nur eine einzige Zeile Quelltext (siehe Bild 3 und Bild 4). Dies funktioniert mit beliebig komplexen CAEX-Dateien.

3. Erzeugen von CAEX-Hierarchien

CAEX unterstützt vier Arten von Hierarchien: eine Instanz-Hierarchie für die eigentlichen Projektdaten sowie drei Bibliotheksarten. Im AutomationML-Editor [3] kann jede dieser Hierarchien mit einem einzigen Mausklick erzeugt werden (siehe Bild 5). Bild 6 zeigt den zugehörigen C#-Quelltext, um alle vier Hierarchien zu erzeugen. Dieses Beispiel unterstreicht eine Erfahrung des Autors: Das Programmieren eines einfachen CAEX-Browsers gelingt innerhalb von 30min.

4. Erzeugen von Hierarchie-Elementen

Im nächsten Schritt wird die Instanz-Hierarchie mit Objekten gefüllt. Ihre Architektur ermöglicht das Modellieren beliebig tiefer Objektstrukturen. Bild 7 zeigt dies beispielhaft anhand von zwei Objekten Tank und Nozzle im AutomationML-Editor sowie drei Attributen des Tanks. Bild 8 zeigt den zugehörigen C#-Code: Zunächst werden Variablen IE1 und IE2 deklariert. Anschließend werden beide Objekte

Tank und Nozzle erzeugt, der Tank wird mit Attributen versehen.

Zwischenfazit
Der Aufwand für die grundlegenden Schritte beim Programmieren von AutomationML/CAEX ist tatsächlich gering. Dies ist nicht auf die beschriebenen Basisanforderungen beschränkt, auch komplexere Funktionen wie das Löschen von Objekten, das Ändern von Attributen, das Referenzieren von Objekten, Funktionen zum Kopieren und Klonen ganzer Teilhierarchien zwischen verschiedenen CAEX-Dokumenten gelingen ähnlich einfach.

Erweiterte Techniken

1. Umgang mit semantischer Viel-falt
AutomationML führte 2011 ein neues Konzept zur Beherrschung semantischer Vielfalt ein. Dies ermöglicht die Realisierung von Datenaustausch zwischen Planungswerkzeugen, ohne dass hierzu ein standardisiertes Datenmodell existieren muss. Das Konzept basiert auf der Idee, proprietäre Daten-Teilmodelle in ihrer ursprünglichen Semantik in CAEX abzubilden. Dazu wird die AutomationML-Datei mit einem Etikett versehen, das Auskunft über den Urspung der Datei gibt. Importer können diese Informationen nutzen, um quelltoolspezifische Sub-Routinen zum Import aufzurufen. Das Konzept wird in [5] im Detail beschrieben, im Folgenden soll die Programmierung erläutert werden. Einzige Voraussetzung für die Anwendung dieses Konzeptes ist die Offenheit der adressierten Planungswerkzeuge [6].

2. Etikettierung von CAEX-Dateien
Bild 9 zeigt, wie das Etikettieren erfolgt: Zunächst wird exemplarisch ein Objekt m vom Typ Metainformation erzeugt. Dann werden die von AutomationML definierten Informationen festgelegt und abschließend wird das Objekt m im CAEX-Dokument eingebunden.

3. Lesen von CAEX-Etiketten
Bild 10 illustriert, wie das vorhandene Etikett einer CAEX-Datei ausgewertet werden kann. Die Intellisense-Funktion von Visual Studio hilft, die relevanten Properties zu sichten.

Fehlersuche in CAEX-Dateien
AutomationML-Dokumente sollen fehlerfrei sein. Fehler sind stets auf das Quellwerkzeug oder den Exporter zurückzuführen. Beim Entwickeln von Exportern treten typische Fehler auf: ID's werden doppelt vergeben, referenzierte Klassen oder externe Dateien existieren nicht. Die AutomationML-Engine bietet Funktionen, um solche Fehler leicht zu finden. Bild 11 zeigt, wie diese Fehlerprüfroutinen aufgerufen werden können. Die Fehler werden in Listen gespeichert, deren Einträge jeweils eine menschenlesbare Fehlerbeschreibung sowie einen Pointer auf das fehlerhafte Objekt enthalten.

```
CAEXDocument myDoc = CAEXDocument.New_CAEXDocument();
```

Bild 2: Beispielcode zum Erzeugen eines leeren CAEX-Dokumentes

```
myDoc.SaveToFile(@"c:\temp\test.aml", true);
```

Bild 3: Beispielcode zum Speichern eines CAEX-Dokumentes

```
myDoc = CAEXDocument.LoadFromFile(@"c:\temp\test.aml");
```

Bild 4: Beispielcode zum Laden eines CAEX-Dokumentes

```
//variabe declaration
CAEXFileType myCAEXFile = myDoc.CAEXFile;
InstanceHierarchyType IH;
SystemUnitClassLibType SUCL;
RoleClassLibType RCL;
InterfaceClassLibType ICL;

//create instance hierarchy
IH = myCAEXFile.New_InstanceHierarchy("DemoPlant");
SUCL = myCAEXFile.New_SystemUnitClassLibHierarchy("MySUCLibrary");
RCL = myCAEXFile.New_RoleClassLibHierarchy("MyRoleClassLib");
ICL = myCAEXFile.New_InterfaceClassLibHierarchy("MyInterfaceClassLibType");
```

Bild 6: Beispielcode zum Erzeugen der Hierarchien

```
//create internal element
InternalElementType IE1, IE2;
IE1 = IH.New_InternalElement("Tank");
IE1.New_Attribute("Length");
IE1.New_Attribute("Width");
IE1.New_Attribute("Height");
//create child internal element
IE2 = IE1.New_InternalElement("Nozzle");
```

Bild 8: Beispielcode zum Erzeugen von CAEX-Objekten

```
MetaInformation m = new MetaInformation();
m.WriterName = "a Source Tile";
m.WriterID = "a Source Tool";
m.WriterVendor = "a Company";
m.WriterVendorURL = "www.acompany.com";
m.WriterVersion = "1.0";
m.WriterRelease = "1.0";
m.LastWritingDateTime = "2012.09.21";
m.WriterProjectTitle = "TestProject";
m.WriterProjectID = "TestProject";
myDoc.CAEXFile.SetMetaInformation(m);
```

Bild 9: Beispielcode zum Etikettieren von CAEX-Dateien

```
MetaInformation m = myDoc.CAEXFile.GetMetaInformation().First();
m.
```

LastWritingDateTime
ToString
WriterID
WriterName
WriterProjectID
WriterProjectTitle
WriterRelease
WriterVendor
WriterVendorURL
WriterVersion

Bild 10: Beispielcode zum Lesen von Etiketten

```
//performing the check
myDoc.CheckFastMultipleIDs(ref IDErrorList);
myDoc.CheckFastReferenceConsistency(ref RefErrList);

//list declaration
public Dictionary<string, CAEXBasicObject> IDErrorList = new Dictionary<string,CAEXBasicObject>();
public Dictionary<string, CAEXBasicObject> RefErrList = new Dictionary<string, CAEXBasicObject>();
```

Bild 11: Beispielcode zum Validieren von CAEX-Dateien

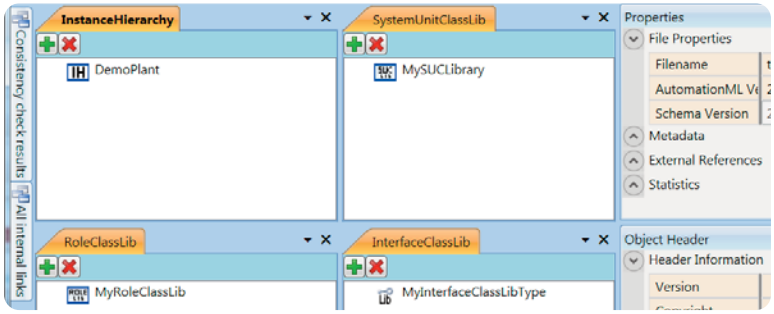

Bild 5: Hierarchien im AutomationML-Editor

Weitere Funktionen

Basierend auf der praktischen Nutzung der AutomationML-Engine wurde diese in 2011 um eine Vielzahl weiterer Funktionen erweitert. Dazu gehören Funktionen zum Importieren von CAEX-Daten aus anderen CAEX-Dateien, Funktionen für das Klonen, Ersetzen und Kopieren von CAEX-Daten, Festlegen von Meta-Daten sowie erweiterte Funktionen zum Splitten und Zusammenführen von CAEX-Dateien.

Zusammenfassung

Die AutomationML-Engine vereinfacht die Arbeit mit CAEX-Dateien erheblich und wird beispielsweise vom AutomationML-Editor bereits verwendet [7]. Der realistische Aufwand zum Programmieren eines AutomationML-Exporters wird auf wenige Tage reduziert, der Hauptaufwand liegt in der Kommunikation mit dem Engineeringwerkzeug. Das Programmieren von Importern ist aufwändiger und erfordert das Mappen der empfangenen Daten mit dem Datenmodell des Zielwerkzeuges. Diese Funktionalität liegt jedoch außerhalb der AutomationML-Programmierung und wird in einem weiteren Beitrag dieser AutomationML-Serie beleuchtet.

Literaturhinweise

[1] IEC 62714-1 CD norm draft AutomationML Architecture, www.iec.ch, 2011.
[2] IEC 62424:2008, Representation of process control engineering – Requests in P&I diagrams and data exchange between P&ID tools and PCE-CAE tools
[3] www.automationml.org
[4] W3C: Extensible Markup Language (XML), URL: http://www.w3.org/XML/ (last access: 19.04.2010).
[5] Drath R., Barth M: Beherrschung von Semantikvielfalt mit AutomationML, Vorgehensmodell für die semantische Standardisierung heterogener Datenmodelle – wie der Umgang mit unterschiedlichen Datenmodellen beim Datenaustausch im heterogenen Werkzeugumfeld gelingt. In atp 12/2012. Oldenbourgverlag 2012.
[6] Drath R., Barth M., Fay A.: Offenheitsmetrik für Engineering-Werkzeuge. In: atp 09/2012, S 46-55. Oldenbourgverlag, 2012.
[7] www.youtube.com/AutomationML

www.automationml.org

Autor: Dr.-Ing. Rainer Drath, Senior Principal Scientist, ABB AG Forschungszentrum Deutschland

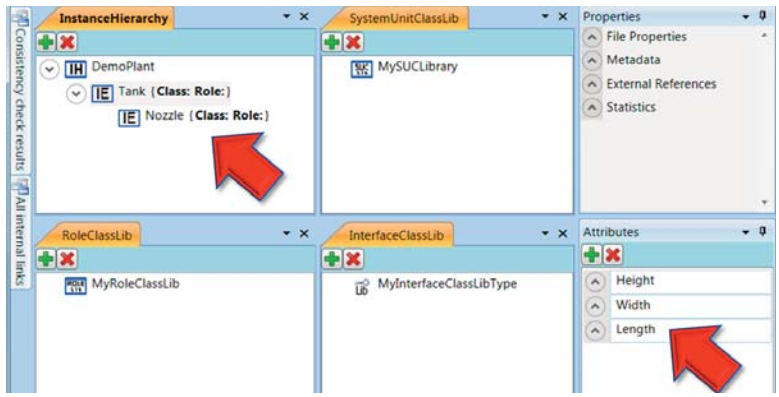

Bild 7: CAEX-InternalElements und Attribute