Address Space Concept in OPC UA

VO Dezentrale Automation
SS 2010

Thomas Hassler
0425981
e0425981@student.tuwien.ac.at

Inhaltsverzeichnis

OPC Unified Architecture Allgemein

Die OPC-Technologie ist im Bereich der Industrie-Automation weit verbreitet. Sie wird als standardisierte Schnittstelle zwischen Automationssystemen verwendet. Anwender können mit OPC auf einfache Art und Weise Produkte verschiedener Hersteller in die Produktionsanlagen integrieren. Es gab jedoch einige technische Einschränkungen welche die Entwicklung und Anwendung behinderten. Um diese Probleme zu bewältigen wurde die Spezifikation der OPC Unified Architecture (OPC UA) veröffentlicht.
OPC UA wurde von der OPC-Foundation spezifiziert um ein standardisiertes Datenaustausch-Modell für Automationszwecke zu haben. Es sollte eine Plattform-unabhängige Technologie geschaffen werden, mit der auch komplexe Systeme beschrieben werden können. Die wichtigste Anforderung war die Interoperabilität zwischen Systemen von verschiedenen Anbietern. Außerdem war das Modellieren von Daten bisher beschränkt und wurde in OPC UA durch ein objektorientiertes Modell für alle OPC-Daten erweitert. Damit können auch komplexe Systeme beschrieben werden; das Konzept ist aber, dass Clients auch auf kleinste Daten zugreifen können, ohne das ganze komplexe System kennen zu müssen.
Die OPC Unified Architecture wurde von über 30 Firmen über einen Zeitraum von zirka fünf Jahren entwickelt. Während der Entwicklung von OPC UA hat sich herausgestellt, dass sich die Technologie auch über die Industrie-Automation hinaus verbreitet. OPC hat in die Bereiche der Heim- und Gebäude-Automation, Sicherheit und noch andere expandiert.
OPC UA benutzt, wie auch schon das klassische OPC, ein Client-Server-Prinzip. Eine Applikation die ihre Informationen für andere Applikationen zur Verfügung stellt wird OPC UA Server genannt und Applikationen die Informationen von anderen Applikationen abrufen sind UA Clients. Eine Applikation kann auch Server und Client zugleich sein.
Das wichtigste Merkmal von OPC UA ist die Möglichkeit, mächtige Informationsmodelle zu erstellen. Diese Modelle definieren die Organisation der Daten innerhalb des Address Space eines OPC UA Servers in Bezug auf Struktur und Semantik.
Der OPC UA Server wird oft als Teil eines Systems zu Verfügung gestellt, zum Beispiel als Teil eines Kontrollsystems. In der OPC UA Systemarchitektur interagieren Server und Clients miteinander.

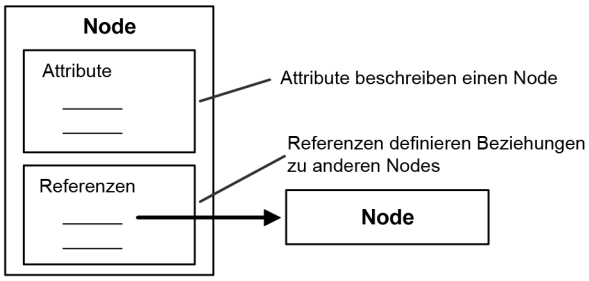
Address Space

Der Address Space von OPC UA ist eine Sammlung von Information. Diese werden vom OPC UA Server für die Clients sichtbar gemacht. Die Menge an Objekten, die der Server für die Clients sichtbar macht wird Address Space genannt. [1] Es werden Informationsmodelle benutzt um den Inhalt des Address Space zu beschreiben. Die fundamentale Komponente des Address Space ist der Node. Der Address Space setzt sich aus Nodes und Referenzen zwischen diesen Nodes zusammen. Der Address Space präsentiert somit ein zusammenhängendes Netzwerk an Nodes.
Die Hauptaufgabe des OPC UA Address Space ist es, einen Standard zu Verfügung zu stellen, der es ermöglicht, dass die Server den Clients Objekte anbieten können.

Dazu wurde das Objektmodell entworfen. Die Elemente dieses Modells werden im Address Space als Nodes dargestellt. Jeder Node ist einer NodeClass zugewiesen und jede NodeClass stellt ein Element des Objektmodells dar. Die NodeClasses beschreiben außerdem die Metadaten für den OPC UA Address Space.
OPC UA bietet verschiedene Beziehungen zwischen Nodes an und ist nicht nur auf eine einzige Hierarchie beschränkt. Ein OPC UA Server kann die Daten in verschiedenen Hierarchien darstellen, abhängig davon nach welchen Kriterien der Client die Daten betrachten möchte.

Nodes

Objekte und deren Komponenten werden im Address Space durch eine Menge an Nodes repräsentiert. [2] Die Nodes werden durch Attribute beschrieben und durch Referenzen miteinander verbunden. Nodes und Referenzen sind die grundlegenden Modellierungskonzepte von OPC UA. Die Abbildung veranschaulicht dieses Prinzip.



Nodes im OPC UA Address Space werden durch einer der NodeClasses definiert, je nachdem welche Aufgabe der Node hat. Nodes können zum Beispiel Instanzen oder Typen repräsentieren. Jeder Node im Address Space ist eine Instanz einer der *StandardNodeClasses* und es sollen keine anderen NodeClasses verwendet werden um Nodes zu definieren. Demzufolge ist es Clients und Servern nicht erlaubt NodeClasses zu definieren oder die bestehenden NodeClasses zu erweitern.
Die *StandardNodeClasses* definieren Attribute die für alle Nodes gleich sind. Jede NodeClass erbt diese Attribute und kann zusätzlich eigene Attribute definieren.

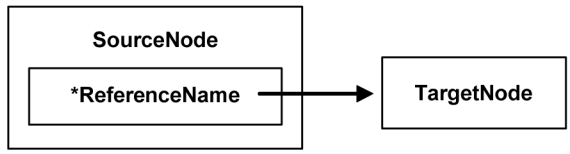
Attribute

Attribute sind Datenelemente die Nodes beschreiben. Abhängig von der NodeClass kann ein Node verschiedene Attribute haben. Einige Attribute sind jedoch bei allen Nodes gleich. Die Definitionen der Attribute sind Teil der NodeClass Definitionen und somit nicht im Address Space enthalten. Die Clients können auf die Attribute mittels den Services *Read*, *Write*, *Query* und *Subscription/MonitoredItem* zugreifen.
Die *NodeId* ist eines der Attribute die jeder Node besitzt und das wichtigste Konzept der Adressierung. [3] Die *NodeId* identifiziert einen Node eindeutig im Server und wird zum Referenzieren benutzt. Der Server gibt die *NodeId* zurück, wenn der Address Space durchsucht oder abgefragt wird. Ein Node kann zusätzlich noch alternative *NodeIds* besitzen, die ebenfalls zur Adressierung verwendet werden können.

Die Definition eines Attributes besteht aus einer ID, einem Namen, einer Beschreibung, einem Datentyp und einem optionalen Indikator. Die Menge an Attributen, die für jede NodeClass definiert wurde soll nicht von Servern oder Clients erweitert werden.

Referenzen

Referenzen werden benutzt um Notes miteinander zu verknüpfen. Wie auch die Attribute sind Referenzen eine grundlegende Komponente von Nodes. Im Gegensatz zu Attributen sind Referenzen jedoch im Address Space sichtbar. Referenzen sind mit Hilfe der *ReferenceType* NodeClass definiert.
Der Node, der die Referenz enthält, wird als Source Node bezeichnet und der Node, der referenziert wird, wird als Target Node bezeichnet. [4] Die Kombination von Source Node der Referenz, der Target Node, der *ReferenceType* sowie die Richtung der Referenz wird in OPC UA verwendet um Referenzen eindeutig zu identifizieren.



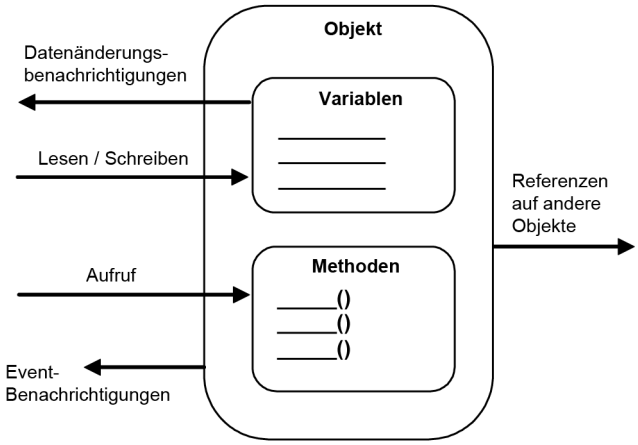
* Name des ReferenceType der Referenz

Die Abbildung veranschaulicht das Prinzip der Referenzierung.
Der Target Node einer Referenz kann im gleichen Address Space sein wie der Source Node oder auch im Address Space eines anderen OPC UA Servers enthalten sein. Es kann auch sein, dass die Referenz auf einen nicht existenten Node zeigt, da OPC UA nicht fordert, dass der Target Node existieren muss. Man kann sich eine Referenz als Zeiger in einem Node vorstellen. Dieser Zeiger enthält die *NodeId* des referenzierten Nodes, den OPC UA Server des referenzierten Nodes, den *ReferenceType* und die Richtung der Referenz.
Es gibt symmetrische und asymmetrische Referenzen. Symmetrische Referenz bedeutet, dass die Referenz in beide Richtungen die gleiche Bedeutung hat. Ein Beispiel für einen symmetrischen *ReferenceType* wäre „Kommuniziert mit“. Dieser hat in beide Richtungen die gleiche Semantik, weile beide Teilnehmer kommunizieren. Beispiele für eine asymmetrische Referenz wären „Empfängt von“ und „Sendet an“. Hier hat die Richtung der Referenz eine Bedeutung, weil es darauf ankommt wer empfängt oder sendet.
Referenzen verbinden zwar zwei Nodes, die OPC UA Server müssen jedoch nicht immer beide Richtungen der Referenz darstellen. Wird nur eine Richtung der Referenz gezeigt, so nennt man die Referenz unidirektional. Werden beide Richtungen der Referenz gezeigt, so nennt man diese bidirektionale Referenzen.
Auf Referenzen kann nicht direkt zugegriffen werden. Clients können aber die Referenzen eines Node durchstöbern, eine Referenz auswählen und somit zum Target Node der Referenz gelangen um wiederum die Referenzen dieses Nodes zu durchstöbern. Auf diesem Weg kann ein Client den Address Space durchstöbern. Der Client muss jedoch damit rechnen, dass der Target Node nicht existiert oder dass er einer Referenz nur in einer Richtung folgen kann. Referenzen können auch Schleifen bilden, es liegt hier an den Clients mit inakkuraten Daten umzugehen.

Die Semantik einer Referenz wird vom *ReferenceType* festgelegt. Ein *ReferenceType* ist ein Untertyp der *Standard ReferenceTypes*. Ein OPC UA Server kann auch eigene *RefereceTypes* definieren. Es gibt hier einerseits hierarchische Referenzen sowie nicht-hierarchische Referenzen. Hierarchische Referenzen werden verwendet wenn eine Hierarchie modelliert werden soll. Ein Beispiel für eine hierarchische Referenz ist *HasComponent* und ein Beispiel für eine nicht-hierarchische Referenz ist *HasTypeDefinition*. Diese beiden Referenzen werden später beim Punkt über *TypeDefinitionNodes* noch erwähnt.
Die *ReferenceTypes* werden im Address Space als Nodes dargestellt obwohl Referenzen keine Nodes sind und auch keine Attribute enthalten. Dies ermöglicht es aber, dass Clients Informationen über die Referenzen, die im OPC UA Server verwendet werden, erhalten können.

Objekte

Das Objektmodell definiert Objekte in Bezug auf Variablen und Methoden. Objekte werden verwendet um den Address Space zu strukturieren. Objekte können auch andere Objekte referenzieren. Mit Hilfe der Objekte können Variablen, Methoden und andere Objekte gruppiert werden. Variablen und Methoden gehören immer zu einem Objekt.
Variablen repräsentieren den aktuellen Status des Objekts und Methoden bieten eine Funktionalität des Objekts. Objekte werden mit der NodeClass *Object* definiert. Objekte selbst enthalten keine Daten, bis auf die Attribute die den Node beschreiben (zum Bespiel das Attribute *NodeId*, das den Node eindeutig identifiziert oder das Attribut *BrowseName*, das einen für Menschen lesbaren Namen darstellt, was hilfreich ist wenn man den Address Space durchstöbert). Die Werte der Objekte werden durch die Variablen dargestellt.



Die Abbildung zeigt ein Objekt mit zugehörigen Variablen und Methoden. Das Objekt enthält Referenzen auf andere Objekte. Clients können sich für Variablen anmelden und werden infolgedessen benachrichtigt wenn sich der Wert der jeweiligen Variable ändert. Des Weiteren können Clients die Variablen lesen oder auch darauf schreibend zugreifen. Es ist den Clients auch möglich die Methoden des Objektes aufzurufen. Diese Methode wird dann ausgeführt und das Ergebnis zurückgegeben

(mehr dazu später im Abschnitt über Methoden). Clients können sich auch für Events anmelden und werden dann ebenso benachrichtigt, wenn ein Event auftritt. Angenommen bei dem Objekt im Beispiel handelt es sich um das Objekt „Motor“, dann kann ein Event generiert werden wenn der Motor nicht mehr richtig funktioniert. Mehr zu Events folgt ebenfalls noch weiter unten im Text.

Variablen

Variablen werden verwendet, um Werte repräsentieren zu können. Alle Daten im Address Space, die nicht von Referenzen oder Attributen der Nodes abgedeckt sind, werden von Variablen repräsentiert. Dabei kann es sich auch um Konfigurationsdaten oder zusätzliche Metadaten, die den Node beschreiben, handeln. Eine Variable kann zum Beispiel benutzt werden um die Temperatur darzustellen, die von einem Sensor gemessen wird.
Clients können die Werte der Variablen lesen, schreiben und sich für Änderungen der Werte anmelden. Eine Variable ist des Weiteren immer Teil von zumindest einem Node, sie kann aber auch zu weiteren Nodes gehören.
Es gibt zwei verschiedene Arten von Variablen im Address Space und zwar *Properties* und *DataVariables*. Sie unterscheiden sich darin welche Art von Daten sie darstellen und ob sie andere Variablen beinhalten können. Bei der Modellierung von Daten ist die Unterscheidung zwischen den beiden Konzepten jedoch nicht immer einfach.
Beide Arten von Variablen werden mit der NodeClass *Variable* definiert.

Properties

Properties beschreiben die Charakteristiken eines Objekts, von *DataVariables* und anderen Nodes, die nicht von den Attributen des Nodes festgelegt sind. Um beim Beispiel mit der Temperatur zu bleiben, kann die Property dazu verwendet werden um festzulegen in welcher Einheit die Temperatur gespeichert wird.
Properties sind in der Hierarchie immer die Blätter. Das bedeutet auch, dass Properties keine Properties für sich definiert haben können. Dies verhindert das Auftreten von Rekursionen. Um Properties eindeutig zu identifizieren, muss der *BrowseName* des Properties im Kontext des Nodes, der das Property enthält, eindeutig sein. Properties sind immer simpel. Es existieren keine komplexen Properties, die Untervariablen enthalten können.
Ein Node und seine Properties sollen immer am gleichen Server abgelegt werden.

Data Variables

DataVariables repräsentieren den Inhalt eines Objekts. Dies kann zum Beispiel die Temperatur sein die ein Temperatursensor gemessen hat. Im Gegensatz zu Properties können *DataVariables* auch komplex sein. In diesem Fall beinhalten sie zusätzliche Untervariablen die Teile der Daten enthalten. Ein Beispiel hierfür wäre die Durchschnittstemperatur, die aus Werten von mehreren Temperatursensoren ermittelt wird. Die Werte der einzelnen Sensoren könnten zusätzlich zum ermittelten Durchschnittswert als Untervariablen dargestellt werden.

Im Gegensatz zu den Properties muss bei den *DataVariables* der *BrowseName* nicht eindeutig sein weil jede *DataVariable* Teil eines Objekts sowie vom Typ *BaseDataVariableType* (oder eines Untertyps davon) ist. Die Semantik einer *DataVariable* ist somit von ihrem Typ definiert ist.

Methoden

Jede Methode wird von einem Node der NodeClass *Method* beschrieben. Methoden sind immer im Besitz eines Objekts. Methoden werden von einem Client aufgerufen, laufen bis zur Fertigstellung am Server und geben das Ergebnis wieder an den Client zurück. Die Lebenszeit der Instanz des Methodenaufrufs beginnt, wenn der Client die Methode aufruft und endet mit der Rückgabe des Ergebniswertes.
Methoden haben keinen eigenen Status, können aber den Status des besitzenden Objekts verändern. Jede Methode spezifiziert die Eingabe-Argumente die ein Client verwenden soll sowie die Rückgabewerte, die der Client erwarten kann. Der Node, der die Methode beschreibt, enthält die Metadaten welche die Argumente der Methode sowie ihr Verhalten beschreiben.
Clients rufen Methoden mit Hilfe des Services *Call* auf (alle Interaktionen zwischen OPC UA Clients und Servern erfolgen mit Hilfe von Services). Die Antwort dieses Service Calls enthält den Rückgabewert der Methode. Methoden sollen relativ schnell ausführbar sein. Wenn der Server lange laufende Prozesse zur Verfügung stellen muss, die vom Client aufgerufen und kontrolliert werden, sollten anstatt von Methoden besser Programme verwendet werden. [5]
Jede Methode wird innerhalb des Kontexts einer existierenden Session aufgerufen. Eine Session ist eine Verbindung zwischen einem Client und einem Server. Wenn die Session während der Methoden-Ausführung beendet wird, dann können die Ergebnisse der Methode nicht zurückgegeben werden und werden somit verworfen. In diesem Fall ist die Methoden-Ausführung undefiniert, das bedeutet, dass die Methode trotzdem bis zum Ende ausgeführt werden kann oder die Ausführung sofort abgebrochen wird.
Das besitzende Objekt einer Methode wird im Service Call spezifiziert wenn der Client die Methode aufruft. Clients können die Methoden, welche der Server unterstützt, entdecken indem sie die Referenzen des besitzenden Objekts durchsehen. Diese beschreiben die unterstützen Methoden.
Ein Beispiel für Methoden wäre eine Methode die einen Motor startet. Es sind jedoch auch komplexere Methoden möglich, zum Beispiel eine Methode die mit den zur Verfügung gestellten Eingabewerten eine Simulation durchführt.

Typen für Objekte und Variablen

OPC UA Server sollen Typdefinitionen auch für Objekte und Variablen enthalten und nicht nur für Datentypen. Es ist also möglich nicht nur die Information zu Verfügung zu stellen, dass eine Variable von einem bestimmten Typ ist (zum Beispiel String oder Integer). Genauso kann Information über den Typ eines Objekts zu Verfügung gestellt werden. Dies ermöglicht es beispielsweise mitzuteilen, dass eine Einheit einen Temperaturwert zu Verfügung stellt. Dies wird ermöglicht indem für Temperatursensoren ein bestimmter Typ definiert wird und dann Objekte von diesem Typ erzeugt werden.

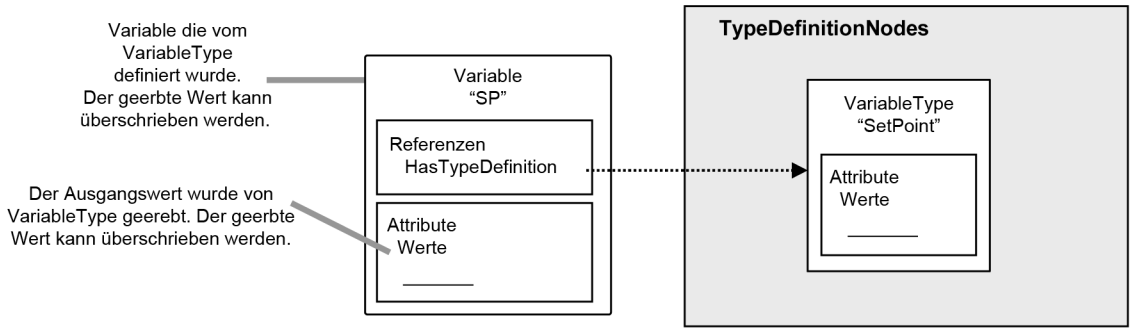
Es können Standardtypen definiert werden und mittels Vererbung können speziellere, herstellerspezifische Typen davon abgeleitet werden. Der abgeleitete, spezialisierte Untertyp kann den Standardtyp erweitern. Dieses Prinzip lehnt sich wieder an objektorientierter Programmierung an.
Für das Erzeugen von Typdefinitionen stellt der OPC UA Address Space die NodeClass *ObjectType* und *VariableType* zur Verfügung. Mit Hilfe dieser NodeClasses können Typdefinitionen von Objekten und von Variablen erstellt werden. Für Methoden besteht nicht die Möglichkeit einer Typdefinition.

Das Bilden von Untertypen von *ObjectTypes* und *VariableTypes* erlaubt folgendes:
- Clients die nur den Supertyp kennen, können eine Instanz vom Untertyp handhaben wie eine Instanz vom Supertyp
- Instanzen vom Supertyp können von Instanzen vom Untertyp ersetzt werden
- Spezialisierte Typen die gemeinsame Eigenschaften vom Basistyp erben können

Mit anderen Worten spiegeln Untertypen die Struktur ihrer Supertypen wider, es können aber zusätzliche Eigenschaften hinzugefügt werden. Ein Hersteller kann beispielsweise einen allgemeinen *VariableType* „Temperatursensor“ um den nächsten Wartungstermin erweitern. Dies erfolgt indem ein neuer *VariableType* erzeugt wird und dieser um das gewünschte Property erweitert wird.

TypeDefinitionNodes

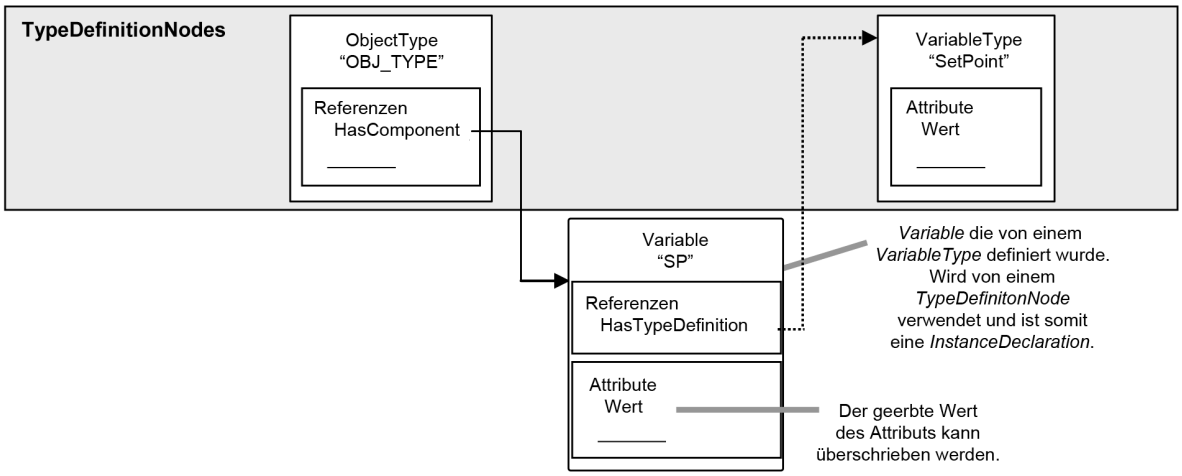
Um ein Objekt oder eine Variable mit einer Typdefinition (die durch einen *TypeDefinitionNode* dargestellt wird) zu verbinden, wird die *HasTypeDefinition* Referenz verwendet. Diese Referenz bindet ein Objekt bzw. eine Variable an seinen *ObjectType* bzw. an seinen *VariableType*. Der Source Node ist hierbei das Objekt bzw. die Variable und der Target Node der *ObjectType* bzw. der *VariableType*. Objekte und Variablen erben die Attribute die von ihrem *TypeDefinitionNode* spezifiziert wurden. Diese können aber später überschrieben werden.
Im Beispiel in der Abbildung wird ein *VariableType* „SetPoint“ definiert. Des Weiteren gibt es die *DataVariable* namens „SP“ (diese ist Teil eines in der Abbildung nicht gezeigten Objektes). Die Variable „SP“ beinhaltet die Referenz *HasTypeDefinition*, die auf den *VariableType* „SetPoint“ zeigt. Somit ist der Typ der Variable „SP“



definiert. Alle Variablen, die vom *VariableType* „SetPoint“ sein sollen beinhalten die Referenz *HasTypeDefinition* die auf eben diesen *VariableType* zeigt. Auf diese Weise wurde der gemeinsame Typ „SetPoint“ definiert, der von verschiedenen Variablen referenziert werden kann.

TypeDefinitionNodes können auch komplex sein. Eine komplexe *TypeDefinitionNode* definiert auch Referenzen zu anderen Nodes als Teil der Typdefinition.

Die Abbildung zeigt ein ähnliches Beispiel wie zuvor, dieses Mal handelt es sich jedoch um komplexe *TypeDefinitionNodes*. Es wird der *ObjectType* „OBJ_TYPE“ definiert, der die Referenz *HasComponent* enthält. Der Target Node solch einer Referenz ist ein Teil des Source Nodes. Der *ObjectType* referenziert hier auf seine Variable „SP“. Die Variable „SP“ ist (wie zuvor) vom Typ „SetPoint“. „OBJ_TYPE“ kann aber nicht nur diese eine Variable haben, es sind auch weitere Variablen vom gleichen Typ möglich (diese müssen sich vom Namen unterscheiden lassen). Es ist



auch möglich, einen Standard-Wert für „SP“ zu definieren. In diesem Fall hätte jede Instanz vom Typ „OBJ_TYPE“ eine Variable „SP“ der dieser Wert zugewiesen wird.
Die Instanzen, die für die Typdefinition benutzt werden, werden *InstanceDeclarations* genannt. Im Beispiel der Abbildung ist dies die Variable „SP“. Auf diese Weise ist es möglichen zwischen den Instanzen die von *TypeDefinitionNodes* benutzt werden und jenen die echte Daten enthalten zu unterscheiden. Ob eine Instanz eine *InstanceDeclaration* ist, ist aber nur im Address Space sichtbar wenn man ihren Referenzen folgt. Ein *TypeDefinitionNode* und seine *InstanceDeclarations* sollen immer am gleichen Server abgelegt sein.

Events

Events stellen spezielle, vorübergehende Ereignisse dar. Systemfehler sind zum Beispiel Events. Im Abschnitt über Objekte wurde beispielsweise bereits das Beispiel mit dem Objekt „Motor“ angeführt welches ein Event im Fehlerfall des Motors generiert. Clients können sich über das Auftreten von Events benachrichtigen lassen. Um derartige Benachrichtigungen zu erhalten, muss sich der Client bei *EventNotifier* anmelden.
Events sind nicht direkt im OPC UA Address Space sichtbar. Es können aber *Objects* und *Views* benutzt werden, um sich für Events anzumelden. Die Nodes besitzen das Attribut *EventNotifier*, das angibt ob es möglich ist, sich bei diesem Node für Events anzumelden. Die *EventNotifier* zeigen die Events zwar an, müssen aber nicht zwingend die Quelle dieser Events sein. Ein OPC UA Server der Events unterstützt soll zumindest einen Node als *EventNotifier* zu Verfügung stellen.

EventTypes

OPC UA bietet verschiedene *EventTypes* an. Die *EventTypes* können auch erweitert werden. Jeder *EventType* wird aber vom *BaseEventType* abgeleitet. *EventTypes* sind abstrakt und werden somit im AddressSpace nie instanziert. Die *EventTypes* die von einem Server unterstützt werden, werden im Address Space des Servers angezeigt. Dies ist nötig damit die Clients diese Informationen abrufen können. Auf diese Weise können die Clients auch Filter definieren. Diese Filter beinhalten Bedingungen die den *EventType* von Events spezifizieren. Sie ermöglichen es also die Menge an Events auf bestimmte Typen einzuschränken.
Um die *EventTypes* im Address Space anzuzeigen wird keine eigene NodeClass verwendet. Es wird für diesen Zweck die NodeClass *ObjectType* verwendet. Dies hat die Vorteile, dass manche Events als Objekte im Address Space dargestellt werden und deswegen müssten für diese Events sowieso *ObjectTypes* erstellt werden. Außerdem muss auf diese Weise keine zusätzliche NodeClass eingeführt werden, was dazu führt, dass Clients *EventTypes* wie *ObjectTypes* behandeln können und keine neuen Mechanismen dazu benötigen.

Views

Die Systeme sind oft groß und komplex und Clients sind oft nur an einem bestimmten Teil der Daten interessiert. Deshalb können OPC UA Server den Address Space in Views unterteilen. Views werden verwendet um die Größe des Address Space zu beschränken. Zum Beispiel kann der Server eine View für die Serverwartung anbieten. In dieser wären nur Nodes sichtbar, die Wartungsinformationen enthalten.
Die Standard-View ist der gesamte Address Space. Es können jedoch weitere Views erstellt werden. Das bedeutet, dass der Server nur einen Teil aller Nodes für den Client sichtbar macht und andere Nodes und Referenzen des Address Space verbirgt. Des Weiteren können Server den Zugriff auf Views auch nur für bestimmte Benutzergruppen beschränken.
Die Views sind im Address Space sichtbar und Clients können die Views durchsuchen. Clients greifen auf die Views zu wie wenn sie voneinander unabhängige Address Spaces wären. Views sind oft Hierarchien, damit sie für die Clients einfacher zu durchsuchen und als Baum darstellbar sind.
Views werden im Address Space als Nodes dargestellt und sie werden mit Hilfe der Klasse *ViewNode* definiert. Der *ViewNode* ist dabei die Wurzel der Nodes in der View. Alle Nodes innerhalb der View sind vom *ViewNode* aus erreichbar wenn man die View durchstöbert. Es muss jedoch nicht jeder Node ausgehend vom *ViewNode* referenziert sein. Es sind also nicht alle Nodes der View vom *ViewNode* aus direkt erreichbar und somit kann es sein, dass beim Durchstöbern der Views Sprünge notwendig sind.
Ein *ViewNode* wird außerdem dazu verwendet den Address Space zu organisieren indem der *ViewNode* der einzige Einstiegspunkt in eine Untermenge des Address Space ist. Clients sollten also die *ViewNodes*, die von einem Server im Address Space gezeigt werden, nicht ignorieren.

Literatur

[1] DIN Deutsches Institut für Normung e.V. *DIN EN 62541-1, OPC Unified Architecture – Teil 1: Übersicht und Konzepte*. Dezember 2008.

[2] DIN Deutsches Institut für Normung e.V. *DIN EN 62541-3, OPC Unified Architecture – Teil 3: Adressraummodell*. Dezember 2008.

[3] DIN Deutsches Institut für Normung e.V. *DIN EN 62541-3, OPC Unified Architecture – Teil 5: Informationsmodell*. Dezember 2008.

[4] Lee Neitzel. *OPC Unified Architecture Internals*. Presented at the ISA, 2004.

[5] Wolfgang Mahnke, Stefan-Helmut Leitner, Matthias Damm. *OPC Unified Architecture*. Springer-Verlag, ISBN 978-3-540-68898-3, 2009.