Seminararbeit
zur VO
Dezentrale Automation

OPC UA
Service Sets and Profiles

Rene Weinhappl
Matr. Nr. 0625626, Kennzahl 535
e0625626@student.tuwien.ac.at

Technische Universität Wien

Institut für rechnergestützte Automation
Arbeitsbereich Automatisierungssysteme

Wien, am 5. Mai 2010

Zusammenfassung

OPC Unified Architectre (UA) ist die neueste Spezifikation, veröffentlicht von der OPC Foundation, mit wesentlichen Änderungen gegenüber den alten Versionen. Die Plattformunabhängigkeit und die Kommunikation über HTTP stand im Mittelpunkt der Interessen der Entwickler. Mit den abstrakten Services und deren Gruppierungen (Service Sets), die Verwendung von Informationsmodellen und erweiterter Sicherheitsaspekte bietet OPC UA eine serviceorientierte Architektur (SOA) für industrielle Anwendungen jeglicher Art.

Inhaltsverzeichnis

1 Überblick

1.1 OPC

Seit den frühen 90er Jahren ist die Anzahl der software-basierenden Automationssysteme in der industriellen Automation stark angestiegen. Es entwickelten sich am Markt eine Vielzahl an unterschiedlichen Bus-Systemen, Schnittstellen und Protokollen, die alle herstellerspezifisch implementiert wurden. Wollte man unterschiedliche Komponenten miteinander verbinden, war dies nur durch eine zeit- und kostenintensive Programmierung bzw. Anpassung der Schnittstellen möglich.
So entstand 1995 die sogenannte OPC Task-Force (***O**bject Linking and Embedding for **P**rocess **C**ontrol*), deren Mitgliederorganisationen Fisher-Rosemount, Rockwell Software, Opto 22, Intellution und Intuitive Technology die gemeinsamen Ziele, einen Plug& Play-Standard für Gerätetreiber und einen standardisierten (Echtzeit-) Zugriff auf Automationsdaten, basierend auf der Technologie *Component Object Model* (COM) und *Distributed COM* (DCOM) von Microsoft Windows, verfolgten. Ausgehend von den unterschiedlichen Anwendungsfällen in der industriellen Automation entwickelten sich drei wesentliche, klassische OPC Spezifikationen:

- **Data Access (DA**) zum kontinuierlichen Lesen, Schreiben und Überwachen von Variablen, die Prozessdaten beinhalten,
- **Alarm & Events (A& E)** zur Behandlung von Event- und Alarm-Nachrichten, im Gegensatz zur kontinuierlichen Kommunikation und
- **Historical Data Access** zum Abrufen und Speichern von historischen Daten.

Weitere Standards folgten, darunter OPC Common, OPC Overwiev, OPC Security, OPC Complex Data, OPC Batch, OPC Data eXchange (DX), OPC Data Access Automation und OPC XML Data Access, die erste, plattformunabhängige Spezifikation basierend auf HTTP/SOAP (Hypertext Transfer Protocol/Simple Object Access Protocol).

1.2 OPC UA

Die schlechte Performance von OPC XML Data Access, die Probleme der COM/DCOM-Technologie mit Firewalls, der Wunsch der Hersteller nach Plattformunabhängigkeit und eine Zusammenfassung der Vielzahl an Diensten zu *Service Sets*, wie in [7] beschrieben, führte zur Verabschiedung der bisherigen COM-basierenden Spezifikationen und zur Einführung der **OPC Unified Architecture (OPC UA)** Ende 2006.

Abbildung 1 zeigt die Spezifikation mit ihren 11 Teilen, wie sie in [1] normiert wurde. Weitere zwei Teile *Part 12 Discovery* und *Part 13 Aggregates* sind nach [7] in diesem Jahr zur Veröffentlichung vorgesehen.

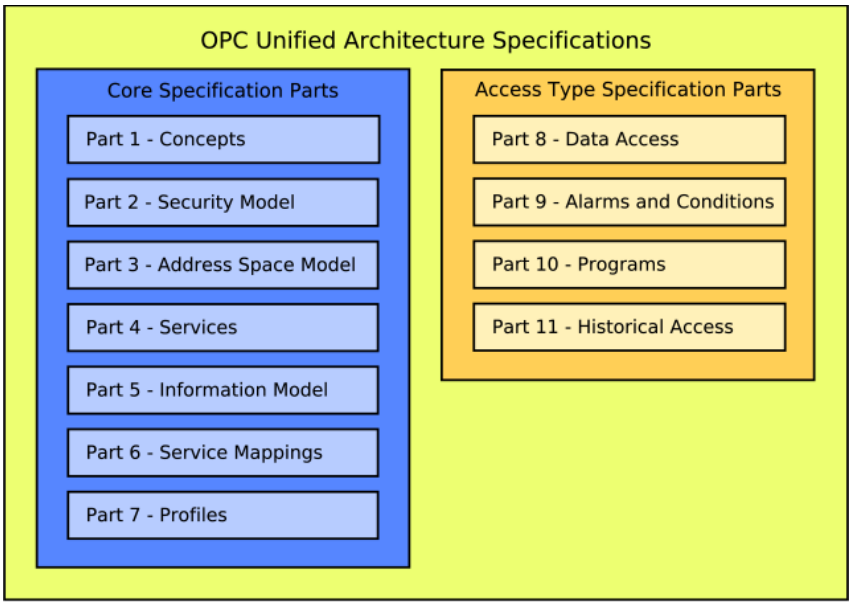

Abbildung 1: OPC UA Spezifikation [5]

Heute zählen über 450 Mitglieder zur OPC-Foundation, darunter alle namhaften Hersteller von sogenannten HMI- (Human Machine Interface) und SCADA- (Supervisory Control and Data Acquisition) Sytemen zur Steuerung und Überwachung technischer Prozesse (vgl. dazu [9, Seite 4]). OPC steht für ***O**peness, **P**roductivity and **C**ollaboration* und hat sich zu dem (universell akzeptierten) Standard zum Austausch von Daten zwischen unterschiedlichen industriellen Automationssystemen etabliert.

Nachfolgend werden **Part 4 Services** und **Part 7 Profiles** der OPC UA Spezifikation genauer betrachtet.

2 Service Sets

2.1 Service Konzept

Damit ein OPC UA Client Zugang zu Daten des Informationsmodells des OPC UA Servers erlangt, stellt OPC UA Methoden, sogenannte **Services** oder **Service Sets**,

Anwendungsfall	**Service Set**
Suchen von Servern	Discovery Services Set
Verbindungsmanagement	Secure Channel Service Set Session Service Set
Anpassung der Struktur des Server-Adressraumes	Node Management Service Set
Suchen von Informationen in Adressraum	View Service Set
Suchen von Informationen in einem komplexen Adressraum	Query Service Set
Lesen und Schreiben von Daten, Metadaten und historischen Daten	Attribute Service Set
Aufruf von Methoden im Server	Method Service Set
Benachrichtigung bei Wertänderungen und Events	Monitored Item Service Set Subscription Service Set

Tabelle 1: Beschreibung der Service Sets anhand von Anwendungsfällen

zur Verfügung. Die abstrakten und protokollunabhängigen Service Sets sind bestimmte Gruppen von Services, die je nach Aufgabengebiet und Nachrichtentyp in OPC UA definiert wurden (siehe [8]). Sie beschreiben also die Datenkommunikation auf Applikationslevel. In Tabelle 1 sind die Service Sets ja nach Anwendungsfall aufgelistet. Die Reihenfolge richtet sich nach der Spezifikation [2] und wird bei der Beschreibung der einzelnen Service Sets beibehalten.

Abbildung 2 gibt einen Überblick über die Einteilung der Schichten in dem OPC UA Kommunikationsmodell. Die Services und Service Sets finden sich in der 2. Schicht von unten wieder. Die Transportschicht packt die Services in ein Protokoll und versendet sie über das Netzwerk. Das generelle Ziel dieser Einteilung war es, auf jedes Datenelement zugreifen zu können, mit einer möglichst geringen Anzahl an Diensten und ohne das Gesamtmodell des komplexen Systems verstehen zu müssen.

Die folgende Erläuterung beschreibt die in der Abbildung verwendeten Abkürzungen: DA definiert Automationsdaten-spezifische Erweiterungen, wie zum Beispiel die Modellierung von analogen und diskreten Daten. AC steht für Alarm & Conditions, ein Modell zum Alarmmanagement und zur Bedingungs-Überwachnung. In HA (Historical Access) sind Mechanismen zum Zugang zu historischen Daten und Ereignissen festgelegt. Mit Prog (Programs) besteht die Möglichkeit, um Programme zu starten, zu verändern und zu überwachen.

De Abarbeitung der Services erfolgt in der bekannten ”Request and Response”-Methode, wie bei Webservices. Damit ergibt sich automatisch ein asynchroner Nachrichtenaustausch bzw. Aufruf der Services. Die UA Stack APIs unterstützen

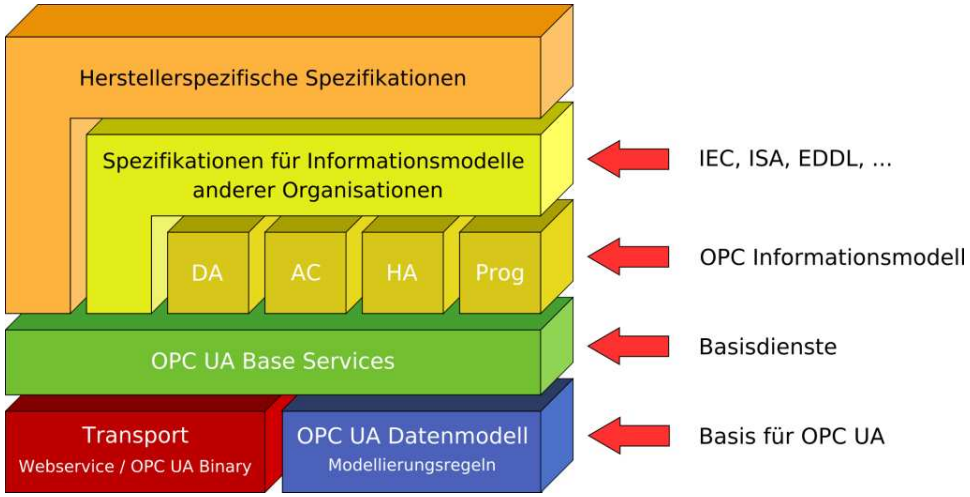

Abbildung 2: OPC Unified Architecture [4]

allerdings auch synchrone Varianten.

Die Fehlerbehandlung erhält einen hohen Stellenwert bei OPC UA. Um den Ausfall des Netzes früh zu erkennen, ist für jedes Service ein eigener Timeout einstellbar. Bei einem verteilten System mit Komponenten unterschiedliche Hersteller können Fehler auf verschiedenen Levels und Szenarien entstehen. OPC UA sieht zwei Typen von Fehlerinformation in den Services vor: Erstens, der *StatusCode* (32-Bit unsigned integer), die höherwertigen 16 Bits repräsentieren den numerischen Wert des Fehlers, wobei die zwei ersten Bits den Zustand des Systems beschreiben (Good, Uncertain oder Bad), die niederwärtigen tragen weitere Information. Zweitens, die *DiagnosticInformation* mit zusätzlicher, herstellerspezifischer Information über den Fehlerzustand.
Die Fehlerausgabe basiert auf zwei Stufen, dem Ergebnis des Aufrufen eines Services und jeden einzelnen Operationen, die innerhalb eines Services ausgeführt werden, da manche erfolgreich, andere fehlgeschlagen sein können.

Jede Service-Anfrage besitzt einen *RequestHeader* und die Antwort dementsprechend einen *ResponseHeader* (vgl. [2]).
Für den *RequestHeader* ergeben sich folgende Attribute:

- AuthenticationToken: Ein geheimer Sitzungs-Identifier zur Zuordung der Anfrage an die Sitzung.
- Timestamp: Zeitpunkt, zu dem der Client die Anfrage gestellt hat.
- RequestHandle: Vom Client definiert Handle für die Anfrage.
- ReturnDiagnostic: Indikator für den Server, um Diagnosedaten im Fehlerfall zu übermitteln.

- AuditEntryId: Ein String, der ein Auditing, dem Tracking von Anwendungen von OPC UA, initiiert.
- TimeoutHint: Dauer, nachdem der Server die (”im Kreis laufende”) Nachricht löschen darf, definiert vom Client.

Der *ResponseHeader* beinhaltet folgende Parameter:

- Timestamp: Zeitpunkt, zu dem der Server geantwortet hat.
- RequestHandle: Der vom Client im RequestHeader definierte Handle.
- ServiceResult: Ein vordefinierter Code für den Aufruf eines Services.
- ServiceDignostic: Die vom Client angeforderten Diagnosedaten.

2.2 Beschreibung der Services und Service Sets

Im Folgenden werden die einzelnen Services und Service Sets genauer beschrieben. Die jeweiligen Parameter der Requests und Responses, die Ergebnis- und Status-Codes, zusätzlich zu den oben genannte Headern, sind in [2] angegeben und werden hier nicht erwähnt.

2.2.1 Suchen von Servern

Mit dem **Discovery Service Set** sind Clients in der Lage, verfügbare Endpunkte, implementiert durch OPC UA Server zu finden und Informationen über diese (häufig sicherheitsrelevante Daten) zu ermitteln. Definiert wird ein Endpunkt als das zu verwendende Netzwerkprotokoll und die nötigen Sicherheitseinstellungen, um sich mit einen Server verbinden zu können. Der Discovery-Prozess ist in Abbildung 3 dargestellt. Server können sich bei sogenannten **Discovery-Servern** im Netz mit dem Service *RegisterServer* periodisch registrieren. Damit der Client eine verfügbare Liste an Servern von einem Discovery-Server erhält, ruft er die Methode *FindServers* auf. Mit *GetEndpoints* liefert ihm der Server in einem eigene, ungesicherten Kanal (Discovery-Endpoint) die oben genannten Daten zur Verbindung mit einem Endpunkt. Die Aufgabe des Clients besteht nun darin, die erhaltenen Daten mit denen bei der Antwort des Servers auf den anschließenden Methodenaufruf *CreateSession* zu verifizieren (in Abbildung 3 nicht eingezeichnet). Mit *CreateSecureChannel* baut der Client eine sichere Sitzung mit dem Server auf.

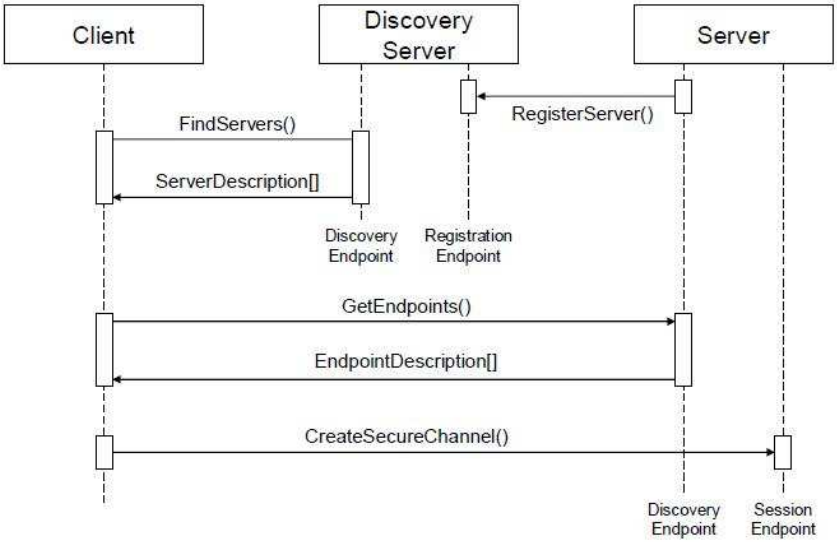

Abbildung 3: OPC UA Discovery-Prozess [2]

2.2.2 Verbindungsmanagement

Das **Secure Channel Service Set** sorgt für Richtigkeit und Vertraulichkeit aller Daten, die zwischen Client und Server ausgetauscht werden. Dieser Service unterscheiden sich von anderen dadurch, dass er im UA Communication Stack implementiert ist. Lediglich für den Sitzungsaufbau, dem Handshake, ist auf Applikationsebene ein Security-Handling notwendig. Die zwei Stack API Methoden sind *OpenSecureChannel* zu Öffnen und Erneuern eines SecureChannels und *CloseSecureChannel* zum Schließen.
Des Weiteren existiert ein **Session Service Set**, der für das erwähnte Handshaking zwei Services, *CreateSession* und *ActivateSession*, zum Beenden der Sitzung *CloseSession* anbietet. Wie im Punkt "Suchen von Servern" beschrieben, ist *CreateSession* notwendig, damit der Client die Validierung der vom Server erhaltenen Informatonen durchführen kann. Eine Sitzung ist in jenem Fall nicht aufrecht, bis das Service *ActivateSession* korrekt durchgeführt wurde. Abbildung 4 verdeutlicht den Zusammenhang zwischen SecureChannel und Session.
ActivateSession wird auch verwendet, um den Benutzer der Sitzung, die Spracheinstellungen zu verändern oder einen neuen SecureChannel zu generieren.
Werden Sitzungen beispielsweise im Fehlerfall nicht beendet, übernimmt dies der Server (über eine Time-Out Periode), damit dementsprechend die Ressourcen freigegeben werden können.
Ein *Cancel* Service dient dem Abbrechen von noch ausständigen Requests, wie zum Beispiel länger andauernde Queries.

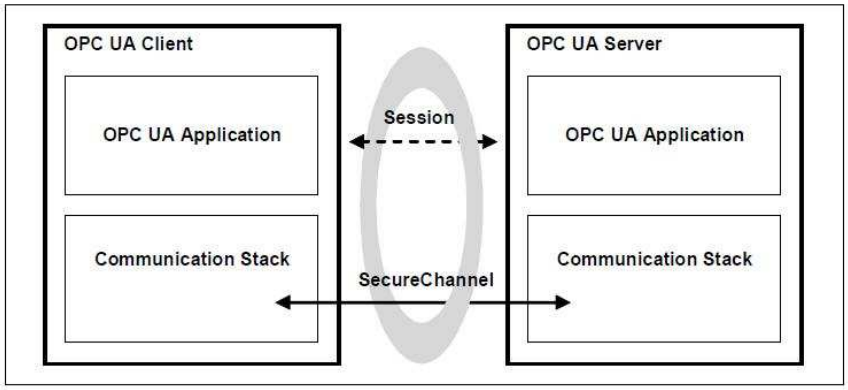

Abbildung 4: Session vs. SecureChannel [2]

2.2.3 Anpassung der Struktur des Server-Adressraumes

Mit dem **Node Management Service Set** hat der OPC UA Client die Möglichkeit, Knoten und Referenzen aufeinander im Adressraum des Servers hinzuzufügen und zu löschen. Dieses Feature dient vorrangig den Servern zur Konfiguration und den betroffenen Clients. *AddNodes* fügt einen oder mehrere Knoten in den Adressraum ein, *AddReferences* fügt eine oder mehrere Referenz(en) zwischen zwei Knoten ein. Mittels der Services *DeleteNodes* und *DeleteReferences* werden ein(e) oder mehrere Knoten bzw. Referenz(en) wieder vom Adressraum entfernt.

2.2.4 Suchen von Informationen im Adressraum

Um einen bestimmten Typ von Daten im Adressraum des Servers oder in einem Subset (View) dessen zu finden, werden zwei Services *Browse* und *Read* im **View Service Set** bereitgestellt. Mit *Browse* wird eine Liste von Ausgangsknoten übergeben und der Server antwortet mit einer Liste von zusammengeschlossenen Knoten zu jedem einzelnem Ausgangsknoten, also die Referenzen eines Ausgangsknoten. Zusätzlich sind mit einem Aufruf von *BrowseNext* jene Teile des Ergebnisses von *Browse* abrufbar, die beispielsweise aufgrund des Umfanges nicht übertragen werden konnten.
Das Service *TranslateBrowsePathsToNodeIds* ist dann vonnöten, wenn ein Zugang zu einem Objekt basierend auf dem Wissen über dessen Typ angestrebt wird. Da die ID des Knoten zum Beispiel zur Variablenerfassung nötig ist, und Knoten auch Objekte instantiieren können, soll deren ID ebenfalls zugänglich sein. Mit Kenntnis des entsprechenden Objekttyp ist dies möglich.
Ein weiteres, hilfreiches Service *RegisterNodes* wird von Clients verwendet, um häufig wiederholt angesprochene Knoten (zum Beispiel durch Aufruf- oder Schreib-Befehle) im Server vorzumerken. Dieser veranlasst jede Möglichkeit auszuschöpfen,

den Zugang zu den entsprechenden Knoten zu optimieren (natürlich nur innerhalb einer Sitzung). Das Gegenstück dazu ist das *UnregisterNodes*-Service zum Freigeben der Ressourcen.

2.2.5 Suchen von Informationen in einem komplexen Adressraum

Die *Browse*-Funktion wurde eingeführt, um durch den Adressraum des OPC UA Servers zu navigieren. Sie findet vor allem in einem kleinen, bekannten Umfeld relevante Daten. Für den Fall, dass es sich um einen sehr großen oder dynamischen Adressraum handelt, wird ein anderer, effizienterer Ansatz vorgezogen. Queries erlauben durch die Verwendung von Filterkriterien die Liste der Knoten erheblich zu verringern. Im SELECT-Teil der Abfrage werden die zu retournierenden Knoten spezifiziert, die durch den im WHERE-Teil angegebenen Attribute eingeschränkt werden. Das **Query Service Set** bietet dafür die Methoden *QueryFirst* und *QueryNext* an. Erstere dient dem Starten der Abfrage, die andere, um verbleibende Reste eines zu langen Ergebnisses abzurufen.

2.2.6 Lesen und Schreiben von Daten, Metadaten und historischen Daten

Das **Attribute Service Set** ist eines der wichtigsten, implementiert in OPC UA. Es beinhaltet Services, nicht nur zum Lesen und Schreiben von Variablen, sondern auch um Attribute von Knoten, also Metadaten, im Adressraum zu bearbeiten. Mit dem Service *Read* lassen sich ein oder mehrere Attribut(e) von einem oder mehreren Knoten lesen. Es ist für eine Menge an Leseoperationen optimiert. Wichtige Parameter, wie Auswählen eines Elementes eines Arrays oder ein Subset von Elementen, Lesen nur von neuesten Daten (um Lesevorgänge zu reduzieren), sind einstellbar. Leserechte (AccessLevel) verhindern unerlaubte Lesezugriffe.
Das *Write*-Service beschreibt analog zum *Read* die selben Werte und ist ebenfalls zur mehrfach-hintereinander-Ausführung optimiert. Typischerweise sind nur Variablen beschreibbar, andere Attribute nur während der Konfigurationsphase, erlaubt durch den Server. Daher zeigen die AccessLevel, welche Variablen, getrennt von den WriteMasks an, welche anderen Attribute veränderbar sind.
Zugang zu historischen Daten und Events erhält man mit *HistoryRead*. Dabei lässt sich über einen erweiterungsfähigen Parameter der Typ von Daten einstellen, wie Rohdaten, modifizierte Daten, berechnete Daten (durch Interpolation, Durchschnitt, etc.) durch Daten zu spezifischen Zeitstempel oder historische Events.
Mit *HistoryUpdate* erfolgt das Einfügen, Löschen, Austauschen oder Updaten von historischen Daten und Events, ebenfalls über einen erweiterbaren Parameter.

2.2.7 Aufruf von Methoden im Server

OPC UA Server können Methoden im Adressraum integrieren. Methoden sind Komponenten von Objekten und können sowohl einzeln, als auch als Bundle in Form von Listen aufgerufen werden. Zum Aufruf durch den Client stellt das **Method Service Set** das entsprechende Service *Call* zur Verfügung. Die Syntax zum Aufruf und die Eingabe- und Ausgabeparameter sind in der jeweiligen Methodenbeschreibung festgelegt, die durch *Browse* und *QueryFirst* ermittelt werden kann. Ein Methodenaufruf steht im Kontext mit der zugehörigen Session, d. h. nach Beendigung der Session und sind noch ausstehende Ergebnisse verloren.

2.2.8 Benachrichtigung bei Wertänderungen und Events

Clients haben die Möglichkeit, über Werteänderungen und Events regelmäßig informiert zu werden, also eine Art Abonnement (Subscription) von interessanter Information abzuschließen. Definiert werden die Elemente in *MonitoredItems*, die Attribute eines beliebigen Knotens sein können. *MonitoredItems* können Variablen, Events oder berechnete bzw. aufgezeichnete Werte anhand mehrerer Variablen in bestimmten Zeitintervallen sein. Bei all diesen zu überwachenden Elementen sind die Parameter Sampling-Intervall (Rate, nach welcher der Wert aktualisiert wird), Monitoring-Mode (aktives oder inaktives Monitoring), Filter und Größe der Queue (Anzahl der zum Client gesendeten Elemente) einstellbar. In Abbildung 5 sind die unterschiedlichen *MonitoredItems* und deren veränderlichen Parameter schematisch dargestellt. Ist die Queue voll, wird eine Mitteilung als *NotificationMessage* zum Client gesendet. Da es unter Umständen etwas länger dauern könnte, bis so eine Queue gefüllt ist, sind bei OPC UA sogenannte *Keep Alive Messages* eingeführt worden, um Kommunikationsprobleme erkennen zu können. Dazu sendet der Server regelmäßig eine solche Nachricht, falls keine *NotificationMessage* ansteht. Auch die Überprüfung verloren gegangener Nachrichten wurde mittels Sequenznummer-Verfahren implementiert.

Zum Erstellen und Managen von Subscriptions ist bei OPC UA das **Subscription Service Set** vorgesehen. Mit dem *CreateSubscription*-Service wird eine solche generiert und die initialen Einstellungen vorgenommen. Relevant ist die Aktivierung und das Intervall, in dem Nachrichten gesendet werden (*PublishingInterval*).
Eine Änderung der Subscription erfolgt mit *ModifySubscription*, Löschen mit *DeleteSubscriptions*.
Mit *SetPublishingMode* wird das Senden von Nachrichten unterbunden, ohne dass das Sammeln der gewünschten Daten und Events deaktiviert wird.
Das Service *TransferSubscriptions* erlaubt das Übertragen der Subscription-Liste zu einem anderen Client, falls der gerade eingesetzte nicht mehr verfügbar ist oder zu einer anderen Sitzung, falls die gerade verwendete abgelaufen ist.
Publish wird vom Client für zwei unterschiedliche Zwecke eingesetzt: Erstens,

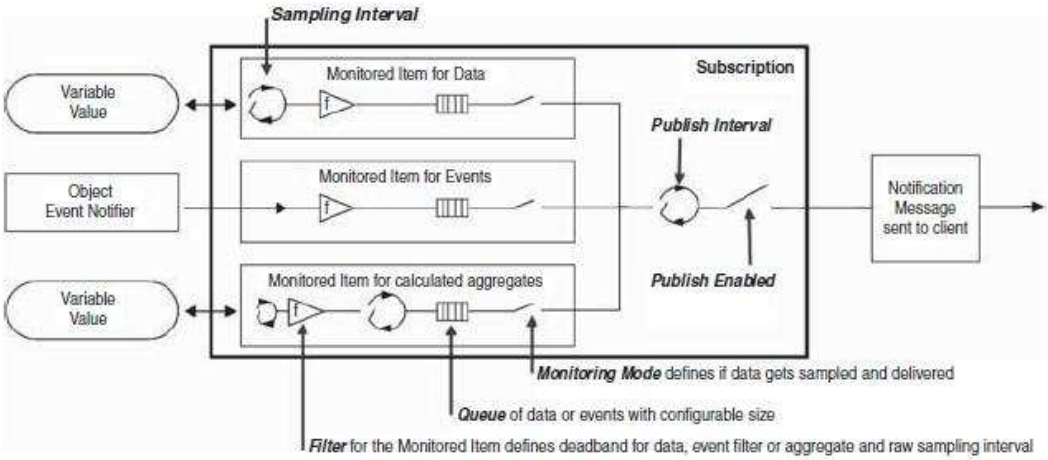

Abbildung 5: OPC UA Subscription [9]

um eine Bestätigung für den Erhalt einer *NotificationMessage* zu senden und zweitens, um vom Server eine *NotificationMessage* oder eine *Keep Alive Message* anzufordern. Letzteres wird etwa dann angewendet, wenn Netzwerke eine hohe Latenz aufweisen und der Client gewisse Pipelining-Konzepte verfolgt.
Eine in einer Retransmission-Queue abgelegten Nachricht wird mit *Republish* nochmals angefordert. Befindet sich keine Nachricht in der Queue, wird eine Fehlermeldung retourniert.

Bisher wurden Subscriptions und dementsprechende Services behandelt. Zur Auswahl, welche *MonitoredItems* in einer Subscription an den Client übermittelt werden, ist im **MonitoredItem Service Set** festgelegt. Die gewünschten *MonitoredItems* und deren erstmaligen Einstellungen werden mit *CreateMonitoredItems* vorgenommen. Zu den Einstellungen zählen das Sampling-Intervall, wie oft die Daten auf Änderungen geprüft werden (unabhängig vom *PublishingInterval*), der Monitoring-Mode (*disabled*, *sampling* zum Aufzeichen ohne Benachrichtigungen, *reporting* mit Benachrichtigungen an den Client), der Filter und die Queue-Parameter, zur Festlegung der Speichergröße und anderer Kriterien, die weiterzuleitende Nachrichten erfüllen müssen.
Zum Bearbeiten und Entfernen der *MonitoredItems* existieren ebenfalls die beiden Services *ModifyMonitoredItems* und *DeleteMonitoredItems*.
Der erwähnte Monitoring-Mode ist auch unabhängig mit *SetMonitoringMode* einstellbar.
SetTriggering erlaubt Benachrichtigungen über Elemente, nur dann, wenn ein anderes Element (Trigger) abgerufen wird, mit Hilfe von Verbindungen zwischen den zu triggernden und dem getriggerten Element(en).

3 Profiles

3.1 Motivation

Die Spezifikation von OPC UA beschreibt eine Vielzahl an Services und Informationsmodellen, die als Funktionalitäten von den Servern und Clients bezeichnet werden können. Folglich ist nicht jeder Server/Client in der Lage, jede Funktionalität zu unterstützen, wie zum Beispiel ein Embedded Server, dessen Aufgabe es nicht ist, historische Informationen bereitzustellen. Aus diesem Grund sind Profile (*Profiles*) spezifiziert worden, die den Herstellern zusätzlich erlauben, diese für Marketing-Zwecke einzusetzen. Der Zusammenhang zwischen Profilen, den nachfolgend erläuterten Conformance Units, Test Cases und dem Zertifizierungsprozess (*UA Compliance Test*) ist in Abbildung 6 gegeben.

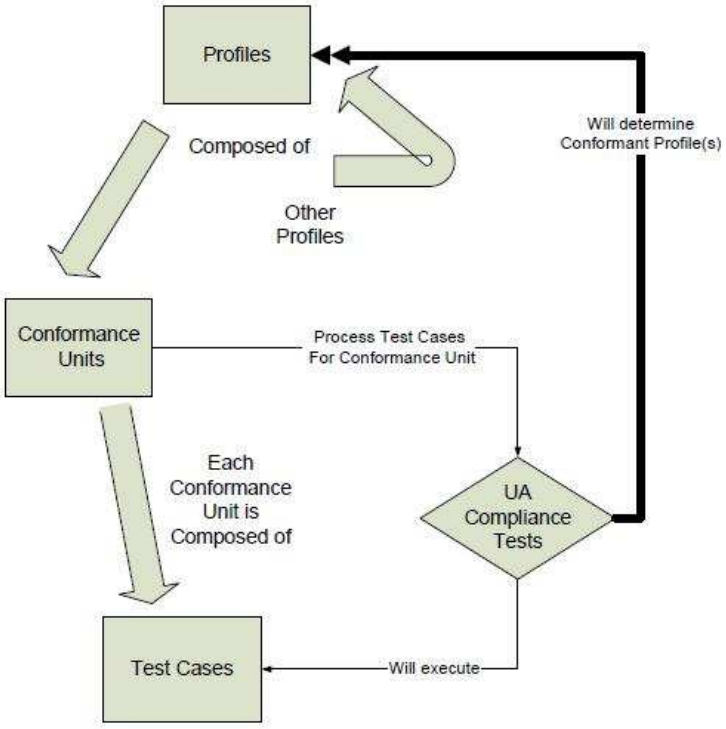

Abbildung 6: OPC UA Subscription [3]

3.2 Profil-Kategorien

Profile wurden kategorisiert, um Herstellern und Endverbrauchern deren Anwendungsbereiche verständlich zu machen. Folgende Kategorien wurden eingeführt:

- **Client** Profil zur Beschreibung der kompletten Funktionalität eines OPC UA Clients. Die URI (Uniform Resource Identifier) ist mit dem *CreateSession*-Service abrufbar und ist Teil des Zertifizierungsprozesses.
- **Security** Profil, das sicherheitstechnische Aspekte (Algorithmen, Schlüssellänge etc.) behandelt. Die URI befindet sich bei der Beschreibung des Endgerätes und lässt sich mit *GetEndpoint* ermitteln.
- **Server** Profil mit Funktionalitäten des Servers. Der URI ist genau so wie bei den Clients ermittelbar.
- **Transport** Profil mit den unterstützen Kommunikationsprotokollen einer Applikation. Der URI ist identisch mit dem des Security-Profils auffindbar.

3.3 Conformance Units

Eine **Conformance Unit** ist ein Teil von einem Profil und definiert eine gewisse Anzahl von Funktionalitäten, die einzeln getestet werden können. Sie beinhaltet also **Test Cases** zum Testen der einzelnen Funktionalitäten. Ein Test Case zum Überprüfen des Call Service bewirkt beispielsweise einen oder mehrere Methodenaufrufe. Zum besseren Verständis und zur Übersichtlichkeit sind die Conformance Units in sogenannte *Conformance Groups* eingeteilt, die mit den oben beschriebenen Service Sets und den Informationsmodellen von OPC UA annähernd korrespondieren. Die jeweiligen Gruppen lassen sich in der Spezifikation [3] nachschlagen.

3.4 Zertifizierung

Das Ziel des Zertifizierungsprozesses ist die Senkung der Integrationskosten für den Endverbraucher, der zertifizierte Produkte erwirbt. Dazu bietet die OPC Foundation [6] zwei unterschiedliche Zertifizierungsmethoden an, welche die korrekte Funktion der von den Herstellern angebotenen Produkte bestätigt. Man unterscheidet bei den Produkten zwischen ”Self-Tested” und ”Compliance Certified”, je nach dem, welches Logo (siehe Abbildung 7) darauf angebracht ist.
Zum **Self-Testing** werden durch die Foundation Tests bereitgestellt, die einerseits die verfügbare Features und andererseits deren korrekte Implementierung überprüfen. Der *Compliance Test* verifiziert ein Produkt nach seiner Spezifikation, der *Interoperability Test* stellt sicher, dass die Interoperabilität mit anderen (getesteten) OPC Produkten gewährleistet ist.
Beim **Compliance-Certified-Testing** prüfen unabhängige Testlabors im *Enhanced 3rd Party Certification Program* auf Korrektheit des Produktes. Zusätzlich zu den in Self-Testing angewandten Methoden werden Verhaltens-, Performance- und Usability-Tests durchgeführt, was einem ”höheren Zertifizierungslevel” entspricht.

Abbildung 7: OPC Product Certification Logo: Self-Tested (links) und Certified (rechts) [6]

Literatur

[1] *DIN EN 62541-1*, 2008.

[2] *DIN EN 62541-4*, 2008.

[3] *DIN EN 62541-7*, 2008.

[4] ascolab GmbH. Opc ua architektur. `http://www.ascolab.com/de/unified-architecture/architektur.html`, 2010.

[5] consultans online. Industrial control protocols on tcp/ip. `http://www.consultants-online.co.za/pub/itap_101/html/ch05s07.html`, 2007.

[6] OPC Foundation. Opc product certification. `http://www.opcfoundation.org/Certification.aspx`, 2010.

[7] OPC Foundation. Opc unified architecture. `http://www.opcfoundation.org/Default.aspx/01_about/UA.asp?MID=AboutOPC`, 2010.

[8] Branko Katalinic Markus Stopper. Proceedings of the international multiconference of engineers and computer scientists vol ii. International Association of Engineers, 2009.

[9] Matthias Damm Wolfgang Mahnke, Stefan-Helmut Leitner. *OPC Unified Architecture*. Springer, 2009.