* Primitive Devices:
  + Kann ich angeschlossene Geräte feststellen?
    - Test vom 01.06.2015 :
      * Analoger Druckknopf an EL1004 angeklemmt -> klemme leuchtet wenn knopf gedrückt wird
      * Scan in TwinCat -> Wenn Knopf gedrückt wird, wird Eingangssignal erkannt, aber kein angeschlossener Druckknopf
      * Scan in Twincat Mit aktiviertem Druckknopf -> gleiches Bild: TwinCat erkennt zwar eingangsZustand, aber nicht analoges Gerät dahinter
      * Idee: Drücken von Analogen Gerätenm detektieren und so Adressierung einteachen, statt auf GUI auwählen zu müssen
      * Fall: Sensor kann nicht manuell getriggert werden: möglichkeit zum einteachen über GUI muss auf jeden Fall bestehen bleiben
    - Recherche vom 01.06.2015 hat nichts ergeben
  + Welche Möglichkeiten gibt es diese Geräte in die Systemperipherie aufzunehmen:
    - Direkteingabe über GUI (ID und Anschlussort)
    - Detektierung durch Triggern der Geräte (Vorteil: automatische Adressierung, ID muss nach wie vor eingegeben werden, da das analoge Gerät von sich aus keine Informationen preisgibt (siehe Arbeit vom Veit oder sonst irgendwo)
* CommunicationChannelMaster wie EthernetMaster sollen auch Devices sein ( anstatt CCDs )
  + Versuch: CCDs allgemein Verfassen und in Repository speichern, um diese wiederverwenden zu können scheitert, da die APIs der Hersteller dies nicht zulassen, da diese zu unterschiedlich sind. Lösung: DeviceDriver versuchen erst einen spezifischen CCD zu laden, falls dieser nicht vorhanden ist, wird ein universeller geladen
* SPS ist bisher nicht vom Typ Device, es hat eine Sonderstellung. Wäre ein Match zu Device möglich? -> hierfür sind Properties wie MAC, IpAdresse notwendig
* Überlegen und begründen, welche CC Verallgemeinerbar sind (in Tapako) oder speziell (im PLC driver)
  + Primitives (vom Benutzer per GUI), Ethernet und NGD sind allgemein
  + Primitives ( evtl. durch höheren Innenwiderstand von SPS), Fieldbus, IO-Link sind speziell
  + Tendenz: Alles in den PLC-Driver packen, da dieser 1. Eine topologische Grenze ist und 2. Auch die PLC Informationen über alle Devices benötigt
* Überlegungen zum Broadcast Search:
  + Durchpingen des Netzwerks nach MAC Adressen und Gerätetypen zu den MAC Adressen heraussuchen:
    - Vorteil: Performancevorteile da nicht jeder spezielle Treiber geladen werden muss
    - Nachteil: Extra Datenbank mit Verknüpfung von MAC und Typ notwendig
* Vergleich zwischen CCD-Prinzip und dem modularen Aufbau ( auch CC-Module werden als Devices behandelt)

|  |  |
| --- | --- |
| **CCD** | **Modular (Devices)** |
| **VORTEILE** | |
| + weniger Informationen über die CCMs Notwendig | + Modularer Aufbau gewährleistet komfortable Skalierbarkeit des Systems |
| + der CC jedes Hosts kann spezifiziert auf einen Host zugeschnitten werden | + Rekursive Abfrage nach Subsysteme ist für jedes Gerät gleich |
| + ein allgemeiner CCD muss nur 1 mal geschrieben werden und kann von jedem Host für jedes CCM benutzt werden | + Das System hat physische und nicht-physische Informationen über sämtliche Komponenten, darunter auch CCMs |
|  | + Genauere Rekonstruktion der realen Topologie (notwendig für Subscriptions) |
| **NACHTEILE** | |
| - Logischer Bruch, da CC-Module in der Realität auch als Geräte auftreten, mit physikalischen Eigenschaften | - ein DD benötigt evtl. Informationen über den Host (Bsp. Die IO-Link-Module an einer Beckhoff-SPS können nur per GUI gescannt werden, andere Hosts können aber vielleicht dasselbe CCM verbauen ) |
| - Gewährleistung, dass unterschiedliche CCMs am selben Host immer denselben CCD verwenden können | - CCMs verfügen über keine Skills und sollten somit auch nicht auf dem SMT erscheinen |
| - Da CCDs nicht als Device in den TreeNode oder ins Parent-System mitaufgenommen werden, fällt es schwer den Pfad zum Anlegen einer Subscription zu rekonstruieren | - CC-Geräte besitzen keine Produktivskills, allerdings wurden Devices bisher über diese Produktivskills definiert |

* Topologiedarstellung ist notwendig, damit im Konfigurationsvorgang der Host oder später die Codegenerierung weiß, auf welche Variablen welche Messwerte subscriptet werden sollen. Hierzu gibt es einige Möglichkeiten eine Topologie darzustellen:

|  |  |  |
| --- | --- | --- |
| **TreeNodes (Topologiebaum der die Devices enthält)** | **Parents- und Childs-Properties (Verweise in einem Device auf das Über- und untergeordnete**  **Device) + topologiefreie parallel Auflistung** | **Connections (Path String + IP-Adresse)** |
| **VORTEILE** | | |
| 1:1 Topologiedarstellung | Einfache Navigation | Simple Implementierung |
| Simple graphische Baumdarstellung | Performance, da alle Geräte in einer Geräteliste parallel aufgeführt werden können und nach Hashwerten sortiert werden können | Keine Suche Notwending |
|  | Möglichkeit ID und Code-Referenz zu speichern |  |
|  | Eventuell keine Childs notwendig (sagt der schlaue Tobi) |  |
| **NACHTEILE** | | |
| Performancenachteile bei der Suche nach Geräten | Zusätzliche Implementierung der Properties | Konvertierung bzw. Verallgemeinerung bei der Kommunikation nach oben notwendig |
| Parser zur Pfaderstellung | Parser zur Pfaderstellung | Unflexibel, Informationsarm |
| Navigation | Zyklische Referenzen erscheinen im OpcUaServer und führen entweder zum Absturz, oder bei Beschränkung der Rekursionstiefe zu performanceeinbußen |  |
|  | Die Rekonstruktion der Klassen aus den Informationen des OpcUa Servers im SMT-Opc Clienten wird komplexer |  |

* Somit fällt die Entscheidung vom 16.06.2015 auf das Family-System
* Die Entscheidung in der Umsetzung fällt auf eine Mischung aus TreeNodes und ConnectionPoints
* Test: BeckhoffPlcSearcher benötigt „45“ Sekunden um durchzulaufen, würden nun sämtliche Suchtreiber durchgegangen werden, würde dies zu erheblichen Perfomanceeinbußen führen. Deshalb haben wir beschlossen einen komplett universelles Suchmodul zu schreiben, welches die MAC-Adressen in angegebenen Subnets des Ethernets ermittelt. (-> hierzu vielleicht Literaturverweise) Die MAC-Adresse ist in einem Device-MAC-Repository (DMR) hinterlegt und verweist auf entsprechende Treiber.
* Große Grundsätzliche Frage:
  + Sollen die Treiber sich beenden und immer zu Tapako zurückspringen und die Treiber der Subdevices werden unabhängig vom ersten Treiber geladen und enthalten sämtliches Zeug, wie z.b. VS öffnen
  + ODER:

Ein Treiber lädt den nächsten und Tapako merkt davon nichts. Somit würde Tapako nur vom MasterHostTreiber die Suche starten und keine Schleife durchgehen.

|  |  |
| --- | --- |
| **Tapako ruft Treiber.SearchForSubsystems() auf**  **(rekursiv)** | **Treiber ruft Subtreiber.SearchForSubsystems() auf**  **(prozedural)** |
| **VORTEILE** | |
| Logik in Tapako, damit leichte Anpassbarkeit von Ablauf | Ein Durchlauf von Oben bis unten |
|  | Spezifische Ablaufdetails werden in Treiber ausgelagert |
|  |  |
| **NACHTEILE** | |
| Schleife mit Test auf ungescannte Devices | Unituitiv |
| Tapako kann keinen Kontext injizieren | Zugriff auf DDR muss enthalten sein |
| Nicht kompatibel mit Inklusion von CCs | Zugriff auf VS funktioniert nicht |

* Problem der Identifizierung eines Gerätes:
  + Twincat liefert vendorId (Verkäufer) + deviceId (Modell)
  + -> Es liefert keine Information über die exakte Deviceinstanz zurück, somit muss ein Weg gefunden werden Informationen wie z.B. Betriebsstunden DeviceInstanzen zuzuordnen
  + LÖSUNG: ???
* Problem der Einordnung in das DDR:
  + Zurzeit des Erkennens des Gerätes, gibt es eventuell keine Informationen über den Hersteller, somit kann auch nicht in den entsprechenden Herstellerordnern nach näheren Informationen gesucht werden
* Lösung: Flache Repositorystruktur
* Priorisierung Logischer Treiberfragmente, danach ermitteln der Instanz und laden der VR

Ideen:

* Statt rekursiv: 3 Listen: finishedDevices, DevicesInSearch, searchRequiredDevices
* Implementierungsdetail fäät weg bei der prozeduralen Suche
* Devices ohne Ahnung von SearchForSubSystems händeln Anfrage an Parent weiter
* Parents durchsuchen so viel sie können. Sobald SearchForSubSystems eines Childs aufgerufen wird, ist es sinnlos den Parent erneut aufzurufen, da dieser bereits alle auffindbaren Geräte gefunden hat.
* Durchsuchung topologisch Ebene für Ebene
* Praxisfern, da CCs mehrere Ebenen in einem Schritt ermitteln können (Beckhoff Ethercat)

Meeting mit Veit vom 08.07:

* Problem beim Identifizieren der unterschiedlichen Gerätetypen (Steckkarten, Devices, Power Module) -> zurückkehren zum ursprünglichen Modell -> nur Devices mit Produktivskill werden bei ScanforSubsystem zurückgeliefert und nach weiteren subsystems durchsucht
* Somit bleibt die Scan-Schleife erhalten, Device ohne Produktivskills fallen raus und CCD ersetzen Devices ohne Produktivskill
* Mögichkeit zum Scannen einer IO-LinkDevice-Seriennummer nur über ADS-Befehl, wenn sich SPS-im FreeRunModebefindet

## 14.07.2015

* Idee im Beckhofftreiber: Blacklist, welche Produktfamilien von Beckhoff (EP,EL,EK rausschmeisst) und die restlichen Devices als „echte“ Devices durchlässt -> Geräte ohne Treiber melden, dass keine Treiber gefunden wurden
* funktioniert nicht, da für evtl. D/A Schnittstellenkarten Primitive Devices hinzugefügt werden müssen
* Wichtig für BA : Problem mit der Aufnahme der Primitives (Aktuelle Idee):
  + CCD-Treiber öffnet GUI, da dieser aktuell alle anfallenden Informationen über die Topologie bereitstellen kann
  + INTERN: Skills haben zu den Pins eines Gerätes n:m Verhältnis, d.h. ein Pin geht auf mehrere Skill-Aus oder Eingänge
  + Trotzdem muss beim Mapping nur jedem Realen Fuß ein Realer Steckkarteneingang zugewiesen werden, da der Benutzer keinerlei Informationen über die interne verarbeitung und Verknüpfung zu den Skills hat
* Für die Erstellung der Gui wird das MVVM Pattern verwendet. Hierzu gibt es einige Frameworks die dies unterstützen, ich benutze das PRISM Framework und den Unity-IOC (erklären was IOC ist – dieser Container zum Auflösen der Abhängigkeiten) (alternative wäre der MEF-IOC)
* Problem vom 28.07.2015: Die Methode „SearchForSubsystems“ der IDevice Schnittstelle liefert ohne Zugriff auf das Driver-Repository nur wenige Informationen wie z.B. Seriennummer zurück. Man 2 Möglichkeiten die Treiber zu laden
  + Beim Aufruf der SearchForSubsystems Methode wird ein IRepository übergeben und der Instanziierende Treiberteil vervollständigt das Device-Objekt mithilfe der Repository-Schnittstelle
    - Vorteile:
      * Entlastung des Hauptprogramms insofern, dass sich Tapako nicht mit konkreten Treiber-Objekten näher beschäftigen muss
      * Es existieren zu keinen Zeitpunkt 2 Objektinstanzen die dasselbe reale Objekt repräsentieren
      * Man muss sich keine Gedanken über das Übertragen und eventuelles Überschreiben anderer Informationen machen
    - Nachteile:
      * Durchschleifen der Schnittstelle bis in die unterste Ebene
      * Komplexere Implementierung des Herstellertreibers
      * Logik ist dort, wo sie nicht hingehört: Treiber finden nur Geräte, sollen diese aber nicht für Tapako vorbereiten müssen (warum gehört die Logik da nicht hin?)
      * Eigentlich gehört diese Logik ja da schon hin, da zum Finden der Untergeräte auch deren digitale Bereitstellung gehört
  + Tapako erhält unvollständige IDevice-Objekte, welche von einer ManagerKlasse vervollständigt werden müssen. Diese ManagerKlasse ist Knotenpunkt sämtlicher Wissenquellen
    - Vorteile:
      * Logik ist nicht verteilt
      * Entkopplung der IDevice-Instanziierung von Treibern der Hersteller
      * Flexible Registrierung von Wissenquellen möglich
    - Nachteile:
      * Komplexe Handhabung der Daten durch die Reihenfolge in der Informationen eintreffen
      * Assemblieren der gesplitteten Device-Informationen
      * Zu einem Zeitpunkt kann es mehrere digitale Repräsentationen eines realen Device geben
    - Überlegung zur Assemblierung: Hat neues oder altes Gerät Priorität?
      * Entscheidung: neues Gerät hat Vorrang, da neues Wissen auf bereits Vorhandenem aufbauen kann (Virtual Representation erst zum Schluss). Das Problem zielt v.a. auf Primitive Datentypen ab, da diese nicht mit null, sondern mit einem konkreten Wert initialisiert werden, denn allgemein Informationen sollte in den Treibern nur an einer Stelle vorkommen und nicht doppelt
      * Entscheidung: Vorgehen bei z.B. String-Arrays: Sollen doppelte Einträge übernommen werden, oder nicht? Stand von 2015-09-04 -> Einträge werden nicht kopiert
    - Entscheidendes Problem: 2 Objekte mit demselben Interface können zwar kombiniert werden, können intern aber zusätzliche Funktionen und Properties haben, die sich im zweiten Objekt nicht wiederfinden. Wenn nun ein IDevice-Objekt die Logik für SearchForSubsystems() und einige damit gekoppelte Methoden mit sich bringt und in ein vorhandenes Device kopiert werden soll, dann kann (sollte die Interface-Methode SearchForSubsystems sogar kopiert werden) evtl. nicht auf die erforderlichen Methoden zugegriffen werden, da diese möglicherweise nicht im Zielobjekt vorhanden sind (und zur Runtime auch unmöglich kopiert werden können -> Literaturverweis).
      * Lösung 1: Strategy-Pattern -> SearchForSubsystems ist keine Methode, sondern ein Strategy-Objekt-Property, welches sämtliche Logik enthält und leicht ausgetauscht werden kann. Diese Strategy wird mit einem Attribut versehen, dass diese Klasse nicht vom DIM gemerged werden darf:
        + Nachteil: Definition des Strategy-Inferfaces und Anpassung des IDevices
        + Nachteil: Strategy benötigt u.U. Referenz auf das zugehörige Device (um z.B. SubDevices hinzufügen zu können) -> Zyklische Abhängigkeit, welche auch im OPC UA auftritt. Gwählte Lösung -> referenz auf ParentDevice in der Strategy ist protected
      * Lösung 2: Der DeviceInformationManager (DIM) muss als Basis-Objekt den Treiber mit der notwendigen Logik Laden und die anderen Ergänzer-Objekte in diesen Treiber hineinkopieren
        + Nachteil 1: der DIM muss entweder erkennen, welches Objekt die Logik enthält, oder eine feste Rangordnung muss mit den IInformationSources mitgegeben werden.
        + Nachteil 2: unflexibel, da verteilte Logik nicht kompatibel ist, denn Logik der Ergänzer-Objekte geht verloren
      * Praxisfern, da bereits eine Geräteinstanz ohne Logik erstellt werden muss, um dem DIM Seriennummer und Modellnummer bereitzustellen. Dann müsste im ersten Schritt die geladene Logik als Objekt bevorzugt werden, und im Zweiten Schritt müssen das bereits vorhandene Objekt bevorzugt werden.
      * Lösung 3: Dekoration
        + Nachteil: Nur Erweiterung, kein ersetzen der Logik -> Hierarchie wieder nicht bekannt
        + Enorme Komplexität bei vielen Treiberfragmenten
        + Viel unnötige Delegierung an ein inneres dekoriertes Objekt
* Auslesen von DeviceSerialNumber:
  + Problem: FreeRunState muss aktiviert sein, damit Io-linkdevices ADS-befehle annehmen
  + Komisches verhalten, da IO-Link Geräte trotz „Toggle Free Run State“ immer im FreeRunState verweilen, sobald dieser einmal aktiviert wurde -> Notwendig ist „Reload Devices“
* Sequenzdiagramm für DeviceInformationManager
  + Ablauf, Rekursion
  + Welche Daten werden wann übernommen / überschrieben / warum -> Literatur?
  + Wie sieht die Schnittstelle zu Informationsquellen aus?
* Event-Aufruf-Hierarchie bei Property-Änderung eines Devices (Was ist hiermit gemeint? Etwa das Subscripten auf ein bestimmtes Device Property ? )
* Parametrierung -> Use Case Ablauf:
  + Möglichkeit 1: Opc-Ua -> Device -> Tapako -> Master -> Runterblubbern ? Vergleich gegenüber
  + Möglichkeit 2: Opc-Ua -> Device -> Hochblubbern bis Host
  + Möglichkeit 3: Opc-UA -> Parameterobjekt -> ValueChangedEvent -> Device -> DeviceHost

## 2015-09-04: Peripherieanalyse – Primitive Devices

Es wird davon ausgegangen, dass Primitive Devices nur weitere Primitive Devices angeschlossen haben können

Folgende Schritte sind durch zu führen:

* 1. Das Hinzufügen von Primitiven Devices muss eingeleitet werden, vorzugsweise von einem Modul, welches Informationen darüber hat, ob ein Primitives Devices irgendwo angeschlossen werden kann
* 2.a Eingeben der Modellnummer
* 2.b Optional: Eingeben der Seriennummer, damit Geräteinstanz bezogene Informationen geladen werden können
* 3. VR und andere Device-Informationen müssen geladen werden, damit Informationen über die Pins zugänglich werden
* 4. Vorhandene Pins müssen Zugewiesen werden

Hierzu müssen folgende Überlegungen angestellt werden

* + 1. Welches Modul initialisiert das Hinzufügen und ordnet das Primitive Devices in den Peripheriebaum ein
    - Tapako:
      * Vorteile:
        + Einmaliger Aufruf, da andererseits für jede einzelne D/A-Steckkarte eine erneute Benutzereingabe erforderlich wird
        + Tapako hat eine GUI
      * Nachteile:
        + Tapako benötigt Topologieinformation des Primitiven Gerätes (Nachteil ist nichtig, da durch die Verkettung von Primitiven Devices kein Modul diesen Nachteil nicht hat)
        + Der Benutzer hat zusätzlichen Aufwand um entsprechende Primitive CCs zu identifizieren
    - CCD
      * Vorteile:
        + Topologie ist klar (außer Primitive Geräte haben primitive Subdevices -> Vorteil ist nichtig, da Ventilinseln diese Forderung stellen)
      * Nachteile:
        + Bei mehreren Primitiven CCDs wird diese Vorgang mehrmals widerholt (Redundanznachteil)
  + 2. Welches Modul übernimmt das Auswählen des Modells (bzw. der ID)
    - Dies ist beliebig, da diese Informationen nur vom User kommen können. Falls verlangt wird, dass das Modul eine Auswahl von verfügbaren Modellen bieten kann, muss Tapako-Schnittstelle oder der DIM muss für Treiber zugänglich sein
  + 3. Welches Modul lädt Device-Informationen
    - Der DIM, dieser kann aber als Schnittstelle angeboten werden
  + 4. Welches Modul übernimmt das Einleiten einer Pin-Zuweisung
    - Tapako:
      * Vorteile:
        + Auch andere Geräte wie beispielsweise IO-Link Geräte ohne CCD können Primitives Gerät gewiesen bekommen -> erforderlich bei Ventilinseln
        + Tapako hat eine GUI
      * Nachteile:
        + Tapako braucht Informationen über CCDs -> CCDs müssen Property des Devices werden, oder CCD muss die Device-Property – Connections erweitern (Erweiterungsnachteil)
        + Tapako weiß nicht, ob überhaupt ein Primitive-CCD vorhanden ist (Informationsnachteil)
    - CCD:
      * Vorteile:
        + dieser hat Informationen darüber, welche Anschlüsse verfügbar sind -> VRs für CCDs müssen geladen werden ->
      * Nachteile:
        + Referenz auf DIM notwendig, da Modellinformationen wie z.B. Anzahl der Pins geladen werden müssen bzw. CCD muss nach DIC(DeviceInformationCompletion)-Vorgang wieder aufgerufen werden (Referenznachteil)
        + Fragwürdig, ob überhaupt ein Primitive-CCD Sinn macht -> Ja, da Anschlussinformationen geladen werden müssen
  + 5. Welches Modul enthält die Logik wie die Portzuweisung abläuft
    - Kein Modul, denn Ablauf ist durch die Implementation selbst gegeben und ist abhängig von dem obigen Entscheidungen
  + 6. Welches Modul enthält die Schnittstelle zum Benutzer für die Portzuweisung
    - Ein Modul, welche Informationen in das Primitive Devices Schreiben kann und dieses an Tapako übermitteln kann -> Beliebig

Zusätzlich muss Recherchiert werden -> welche Schritte kann man aufteilen und welche nicht

* Definition CCD: Ist ein Primitive-CCD überhaupt ein CCD? -> Es findet keine Kommunikation im eigentlichen Sinne statt -> was ist Kommunikation in der Industrieanlage?

Zu Recherchieren -> Können andere Programmteile als Tapako Gui-Elemente starten?

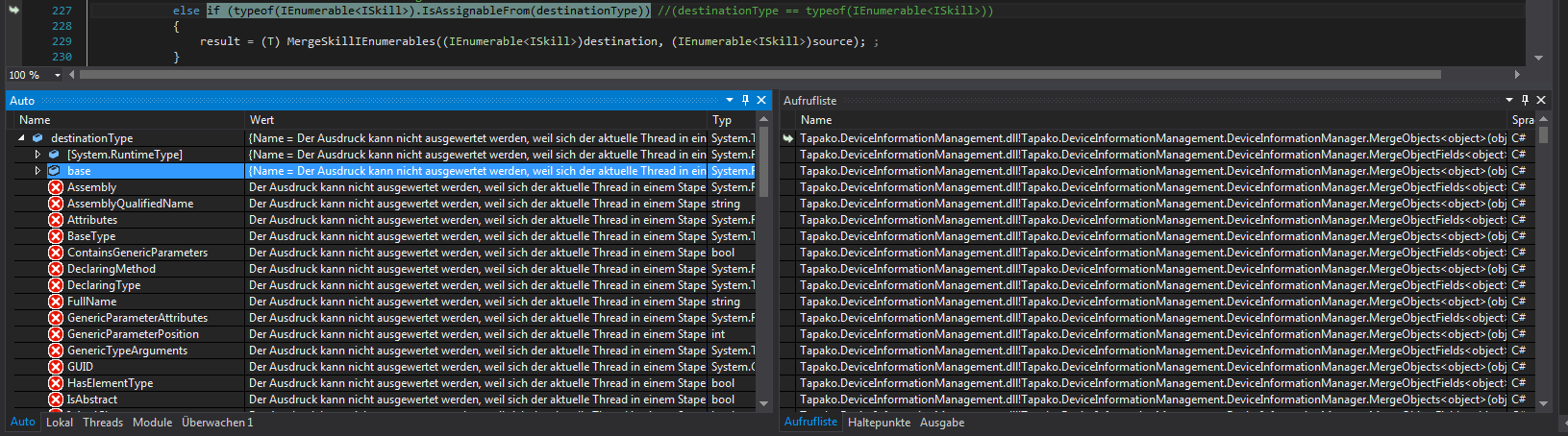
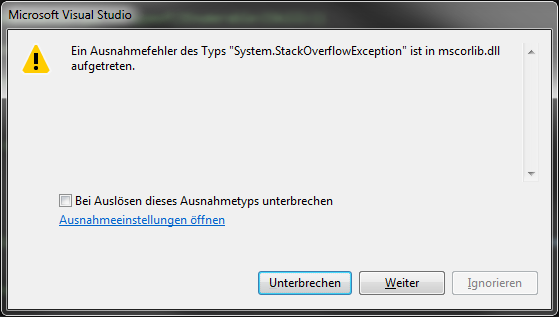
Abwägung (evtl. Gewichtung der Nachteile und Vorteile in einer Matrix): siehe Excel Tabelle

## 09-09-2015

Getestet wurde der Device Information Manager mit 2 unterschiedlichen Klassen. Nachdem er den Merge-Test mit 2 Identischen Klassen und unterschiedlichen Inhalten bestanden hat, wurden Tests mit 2 unterschiedlichen Klassen und der gleichen Schnittstelle durchgeführt. Es hat sich herausgestellt, dass sobald Funktionen in den Klassen definiert werden, der DIM beim Mergen Abstürzt -> siehe TestCopyLogikBetweenDifferentClasses. Dies geschieht dadurch, dass die Klassen Objecte vom Typ Action bzw. Func enthalten haben, welche auf die jeweiligen Methoden verweisen.

Somit ist der Schritt gerechtfertigt eigene Strategy-Objekte für das IDevice einzuführen.

* Der DIM ist nun soweit abzuändern, dass er nicht mehr alle Felder und Properties der Klassen durchsucht (man stelle sich vor 2 Properties verschiedenen Klassen tragen denselben Namen), sondern man muss nach dem nächsten Gemeinsamen Typen der 2 Klassen suchen, und die Properties dieses Typs kopieren.
* Der Fehler der durch Properties vom Typ Func<> oder Action entsteht ist unbedingt zu vermeinden, da dieser Sehr schwer zu finden wäre. (nichtssagende Debuginformationen in einer unwesentlichen Codezeile

Überlegung: Wenn der Hersteller nun (durch schlechten) Programmierstil nicht nur die Properties des IDevice implementiert und mit weiteren Logikfunktionen erweiter, auf die die DeviceStrategy Zugriff braucht, dann schlägt verursacht das Mergen des DIMs unvorhersehbares verhalten, da der Treiber evtl. auf nicht vorhandene Methoden einer falschen Instanz zugreifen will. Abhilfe wäre geschafft durch folgende Wege:

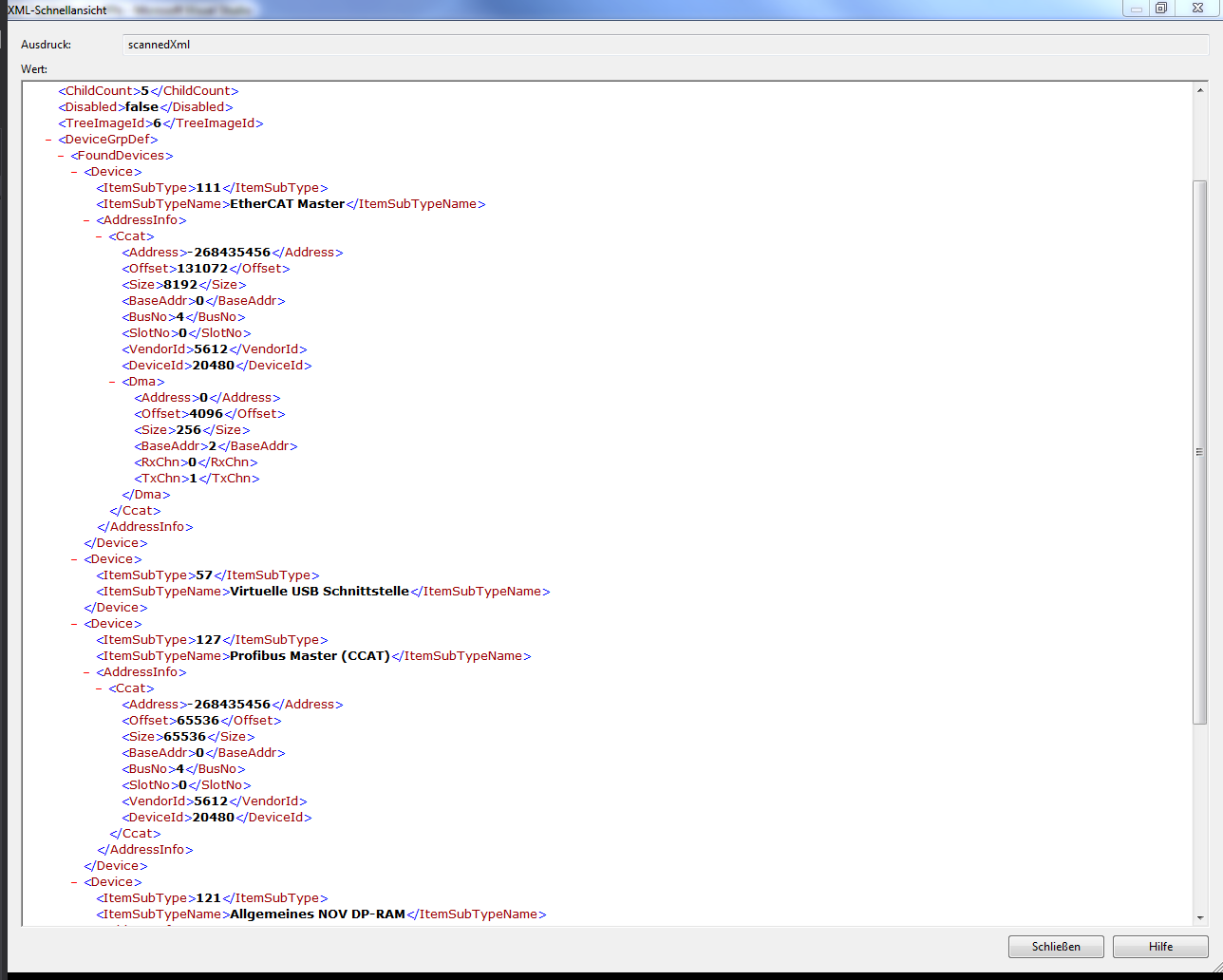
* Der DIM erstellt, statt zu mergen, eine neue Klasse mit sämtlichen gefundenen Properties ( Problem ist, dass hier gleichnamige Properties, die eventuell unterschiedliche Aufgaben erfüllen, gegenseitig überschrieben werden ( unverhältnismäßiger Aufwand, denn dies ist eine Wissenschaft für sich )
* Der IInformation Sources haben Prioritäten und werden entsprechend der Prioritäten ausgeführt (sehr einfache Implementation, jedoch nicht so sicher, da nicht gewährleistet ist, dass Logikcode nur aus dem DDR kommt)

Todo: Untersuchen, was im DIM passiert, wenn eine klasse eine zusätzliche Property hat, und diese von der SchnittstellenMethode aufgerufen wird

Im DIM todo: SetValueDirect ausprobieren, vielleicht umgeht dies die Setterlogik

Nach Einführen des Strategies hat sich ein weiteres Problem herausgestellt: Die Strategy darf nicht mit einer referenz auf das Parent Device instnanziiert werden, da dieser Referenz nach dem Mergen unverändert bleibt. Wird nun die Execution der Strategy aufgerufen, wird eine Veraltete Device-Referenz zum Informationsaustausch verwendet.

## 2015-09-14

Frage von Tobi: Wie können neue Geräte beim erneuten Scannen (oder Scannen bei geöffneten Project mit Geräten) unterschieden werden von bereits hinzugefügten?  


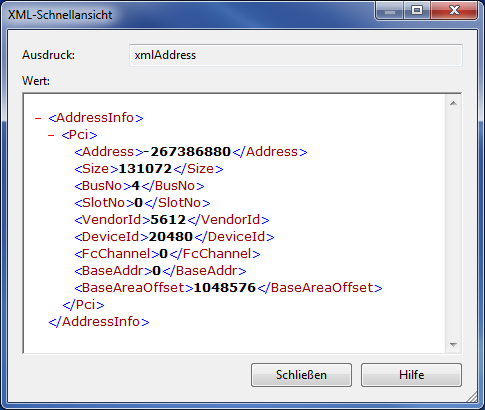
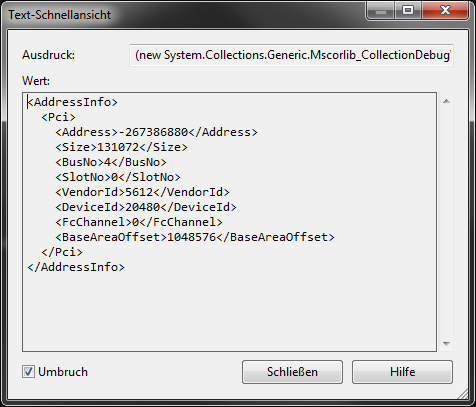
Nicht möglich über Vendor- und DeviceID

ItemSubType (-Name) nicht eindeutig

Adress auch nicht eindeutig.

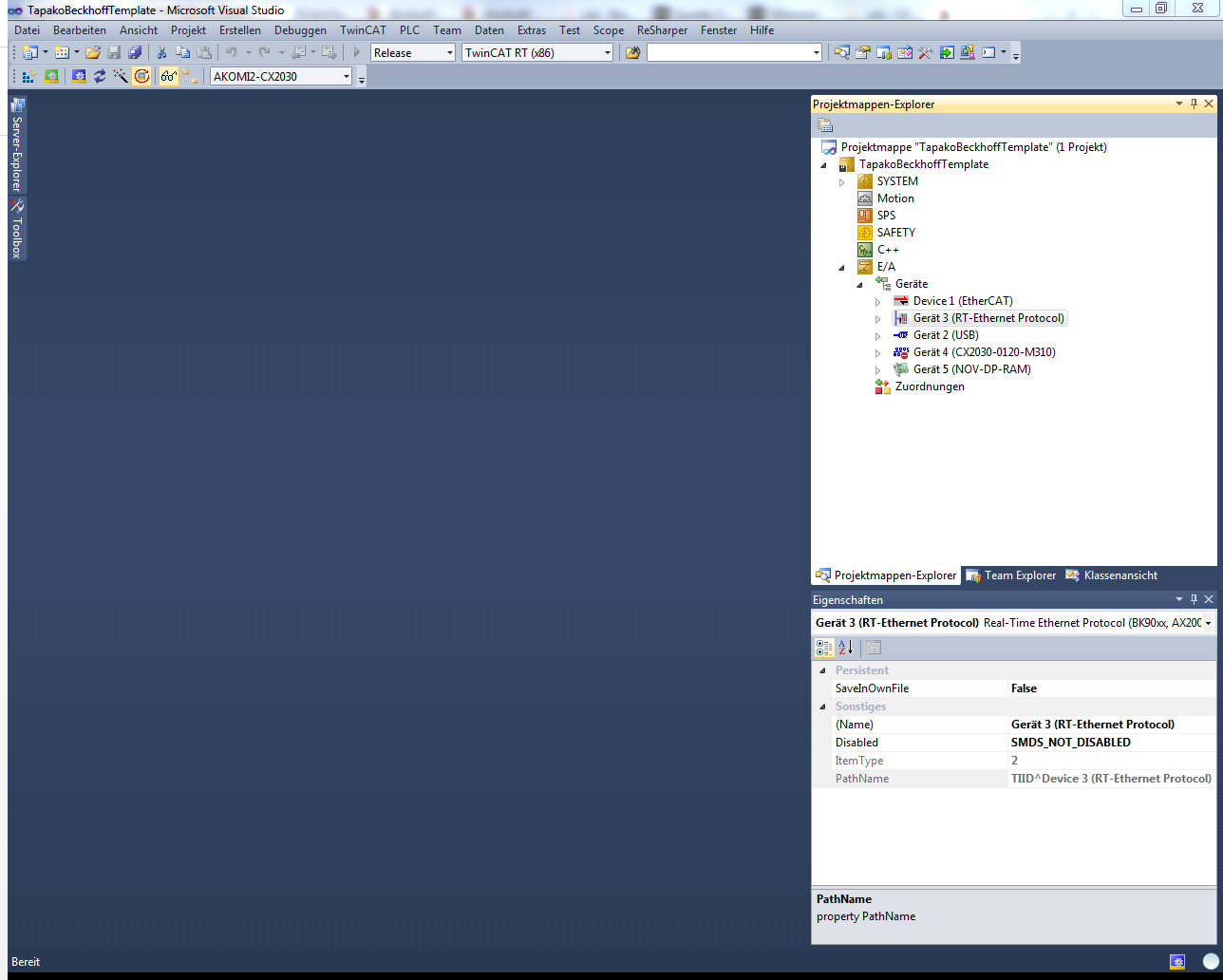
Evtl kombination von Address und Offset eindeutig -> evtl eindeutig aber nicht überall vorhanden

Komplette AddressInfo XElement vergleichen!? -> mit „==“ schlecht, da nicht immer gleiche Infos

Bisherige hässliche Lösung: manuelles codebasiertes anpassen der XML Daten

## 2015-09-15

Problem: Gerätenamen entspricht nicht unbedingt Teil des Gerätepfades. Z.B. :

dies ist bei über Tapako neuhinzugefügten kein Problem, aber beim Einlesen eines vorhandenen Projekts. Da TwinCAT in der deutschen Version, standartmäßig einen deutschen Namen gibt und einen englischen Pfadnamen. Zugriff über TwinCAT API folgt über Pfadnamen, aber bei IO-Link erfolgt Zugriff über VS API und diese sieht nur den Namen.

Markus:

* Auslagerung des SearchForSubsystems-Algorythmus durch Command-Pattern oder Strategy Method-Pattern ? Abwägung welches Pattern verwendet wird: Quelle: <http://stackoverflow.com/questions/3883692/strategy-pattern-vs-command-pattern>
* Strategies encapsulate algorithms. Commands separate the sender from the receiver of a request, they turn a request into an object.
* If it's an algorithm, how something will be done, use a Strategy. If you need to separate the call of a method from its execution use a Command. Commands are often used when you queue up messages for later use, like a task or a transaction.
* Somit wird in unserem Fall das Strategy-Pattern angewendet, da verschiedene Verhaltensweisen mit demselben ergebnis gefordert sind, dabei ist die unabhängigkeit vom Aufrufer nicht notwendig.

# 2015-09-17

* Wiring-Tool wird im CCD aus folgenden Gründen implementiert werden:
  + Kein Verlassen des SearchForSubsystems-Loops, da nicht zu Tapako gesprungen werden muss
  + Steckkarten treten im DeviceTree nicht als Devices auf, somit muss der CCD das hinzufügen einleiten, da Tapako niemals von der Existens der Steckkarten erfahren wird
  + Keine zusätzliche Schnittstelle für Informationsaustausch zwischen Tapako und CCD benötigt
  + Eventuell zusätzliches Wiring Tool in Tapako für Primitive Devices, welche an anderen Geräten als CCDs hängen (z.B. Luftdruckgerät an Ventilinsel)

# 2015-09-17

* Wiring Tool nimmt Liste mit allgemeinen Connection-Objekte an und verknüpft diese – Diese Schnittstelle kann von den Herstellern verwendet werden –> Darauf in BA eingehen, was sonst noch gegangen wäre
* Klassifizierung von Connections (Beispiel RJ-45 -> Mechanische, Elektrische und IT- Eigenschaften)

# 2015-09-29

* Die Schnittstelle zwischen Wiring-Tool und CCD sind Listen von Connections, welche miteinander verbunden werden können.
* Zusätzlich kann das HmiImage mitgegeben werden ( bessere Orientierung mit den Connections )
* Möglich wäre als Schnittstelle auch eine Liste von Objekten (zur erhöhung der Allgemeinen Kompatibilität), aber dann fehlen Namen zu den einzelnen Connections
* Container mit Übergabe-Parameter in Strategy, damit referenz auf DIM erhalten wird
* Adapter Pattern für Tapako ? -> wie oft ändert sich IDevice? -> Hinten angestellt, da der Aufwand zu nicht gerechtfertigt ist, da ist 1. das IDevice nicht mehr ändern sollte und 2. durch die Implementation des abstrakten DeviceBase sowieso eine Art Adapter vorhanden ist
* In BA spezifizieren, welche Arten von Connections es gibt und wie man sie spezifizieren kann (was ist eine Connection? aktuell ist eine Connection eine Verbindung, die zwar aus mehreren einzelkabeln bestehen kann, aber nur im Ganzen Sinnvoll mit einer anderen Verbindung verbunden werden kann, oder nur als Ganzes seinen Zweck erfüllen kann -> was ist mit optionalen Anschlüssen?)

# 2015-09-30

* DIM: Fragestellung, ob beim Aufruf von MergeObjects<T1,T2,T3> mit T2 und T3 als typ der übergabeparameter und T1 als erwarteten rückgabe-Typ implementiert werden soll oder
* MergeObjects<T1, T2> und als Rückgabetyp wird der Typ des ersten Übergabeparameters verwendet. -> aktuell wird Lösung 2 implementiert, da als erwarteter Typ auch interfaces möglich sind, Interfaces als Instanz aber eine Klasse haben müssen. Bei Lösung 1 wäre die Frage Problematisch, welche Implementierung des Interfaces verwendet wird, deshalb verwende ich die übergabeinstanz des ersten Übergabeparameters als Basisinstanz

# 2015-09-01

* Der DIM kopiert nun Array, Skill-Listen und wurde gerefactored
* der DIM ersetzt noch nicht die Verweise der Strategy -> Lösung 1: Strategy doch z.t. mergen, Lsg. 2: Strategies nicht kopieren lassen, Lsg. 3: bestimmten, dass das Device mit der Strategy das Zielobjekt des Mergens ist
* todo im DIM: Bit-Masken für Merginverhalten verfügbar machen

# 2015-09-01

* Aufgrund des Beckhoff Treibers besteht die Forderung nach einer Referenz auf den DIM. Denn wenn der primitive CCD läuft, und man ein modell auswählt, muss gleich der Treiber dazu geladen werden, um herauszufinden, wie viele Connections das angegebene Device aufweist. Ausserdem benötigt auch der IO-Link-CCD eine vervollständigung seiner Devices, da gewisse XML Dateien in das Beckhoffverzeichnis kopiert werden müssen, damit mit den IO-Link geräten kommuniziert werden kann und ihre Seriennummer herausgefunden werden kann.
* Folgende Möglichkeiten stehen zur Auswahl:
  + Der Beckhofftreiber instanziiert einen eigenen DIM und registriert eigene Information Sources darin, welche das IO-Link-Zeug vervollständigen
    - Problem: bereits registrierte IInformation-Souces stehen dem Treiber nicht zur Verfügung
  + Parameter werden bei Aufruf von Strategy.Execute() mit übergeben:
    - Problem: OpcUa Server kann manche Methoden mit komplexen Methodenparameter nicht registrieren und stürzt ab
    - Eindeutige Parameteranforderungen sind erkennbar
  + Das Strategy Objekt hält ein Property vom Typ List<object> in dieses werden sämtliche optionale Parameter vor aufruf reingeschrieben.
    - Problem: eventuell muss kommuniziert werden, welche optionalen Argumente vom Treiber erwartet werden
    - Vorteil: Keine Änderungen an der Infrastruktur nötig, falls ein zusätzliches Objekt übergeben werden muss
  + DENNOCH: In der Praktischen Umsetzung wurde der DIM erstmal als Static-Class behandelt, wodurch eine Instanz-referenz unnötig wird
* Problem mit dem DIM in kombination mit strategy muster: Das Strategy- Objekt benötigt eine Referenz auf sein deklarierendes Objekt, damit die Strategy gefundene SubDevices darin eintragen kann. Der DIM kopiert allerdings ein Strategy als komplettes objekt und merged dieses nicht. Nun ist es bei falscher Reihenfolge der fall, dass das Strategy in ein neues IDevice-Objekt kopiert wurde, das Strategy selber allerdings als Parent-Device noch das alte Device referenziert hat. Lösungsvorschläge:
  + Parameterübergabe -> Passt nicht zur aktuellen architektur
  + Übergabe per InputArgument über den InputArgumentContainer eines Strategies -> So haben wir das gemacht, und es hat auch auf Anhieb funktioniert, juhu. -> Man muss hier noch eine Allgemeine Lösung finden, woher Tapako weiß, dass der Treiber das aktuelle Device als Parameter benötigt ( oder man übergibt es immer )
  + Definierte Reihenfolge der Objekte ( wissen darüber, welches objekt das Strategy instanziiert hat ist erforderlich )
  + Mit einem C#-Hack-Vorgehen in einem Strategy dynamisch das deklarierende Parent-Objekt herausfinden ( Überprüfung der Möglichkeit in C# notwendig – nichts besseres als <http://stackoverflow.com/questions/27825575/get-invoked-methods-parent-object-type> gefunden)

# 2015-11-03

* Entwurfsmuster: DispatchingObservableCollection benutzt SynchronisierungsProxy

# 2015-11-05 - Parametrierung

* Folgende Funktionalitäten sind gefordert:
  + Anzeigen aller Parameter im Skill Modelling Tool
  + Schreiben der Parameter durch setzen eines Wertes im Skill Modelling Tool
* Folgende Voraussetzungen sind zu erfüllen:
  + Write Parameter und Read Parameter Funktionen haben Referenz auf HostDevice
  + Write Parameter und Read Parameter müssen den DIM übersetehen
* Folgende Lösungsvorschläge:
  + Direkter Zugriff auf HostDevice, dabei wir das entsprechende zu parametrierende IDevice übergeben
    - Problem: Herausfinden der Parameternamen, da diese in einer Liste des Devices gespeichert werden müssen -> undurchsichtige und quere Abhängigkeiten
  + Aufruf einer Funktion „Setze Parameter“ in IDevice
    - Problem: IDevice muss Referenz auf HostDevice halten
    - Logik direkt im IDevice
  + Einführen einer ParameterListe im IDevice (Liste von Parameter-Objekte)
    - Falls nun ein Parameter gesetzt werden soll, wird der Value-Wert des Parameter Objektes gesetzt und ein Event wird ausgelöst. Dieses Event ruft die Parametrierungsfunktion des Hosts auf.
      * Woher hat das Parametrierungsobjekt die HostReferenz?
      * Der Host injiziert seine eigene Referenz -> callback
      * Todo: weiterschreiben!!!

# 2015-11-06 - Parametrierung

* Die Parameter dürfen keine Beckhoffspezifische Schnittstelle zur Parametrierung bzw. AMS-Befehlsausführung verwenden, da sonst eine Abhängigkeit vom HostSystem entstünde
* Möglichkeit1 -> Vormerken eines Hosts und dieser wird angesprochen
  + Nachteil: unflexibel, falls Kommunikation über zwischenstellen stattfinden muss. Szenario : Alle CCDs werden als IDevices hinzugefügt, dann könnte ein IOL-Master-Device evtl. die ADS-Kommunikation übernehmen
* Möglichkeit 2 -> Aufrufen einer Methode des Device Parents.
  + Parametrisiere würd dazu zu einem allgemeinen IDevice Member
  + Jedes device leitet die parametrisierung an das Parent weiter
  + Nachteil -> zyklische strukturen im OPC UA Server –> parent hat Referenz auf child und child hat referenz auf Parent
  + Lösung 1: Einführen von Member Attributen, um bestimmte Member im OPC Ua Server auszuschließen. Dies hätte eine allgemeine Steigerung der Flexibilität der Architektur zur Folge, somit sowieso sinnvoll
  + Nachteil : Jedes Device muss diese Methode implementieren, andererseits wäre die Kette unterbrochen
* Möglichkeit 3 -> **Event System:** Jedes Device wird von parametrisierenden Parents abonniert
  + Event.Handled flag wird gesetzt, sobald die parametrisierung stattgefunden hat
  + Keine Referenz von Child auf Parent nötig
  + Kein unnötiges durchschleifen einer Methode durch den kompletten hierarchiebaum
  + Flexible Übergabemöglichkeit von vielen Informationen zur Parametrisierung per EventArgs
  + Observable Pattern und lose Kopplung !!!

# 2015-11-06 – Parametrierung

* Aufbau der Parameter XML:

<Parameter>

<Name>Direct Parameters 2</Name>

<ObjId>V\_DirectParameters\_2</ObjId>

<Index>0x1</Index>

<ByteLenth>16</ByteLenth>

<Items>

<Name>Device Specific Parameter 1</Name>

<Subindex>0x1</Subindex>

<Datatype>UINT8</Datatype>

<Length>8</Length>

<BitOffset>120</BitOffset>

<Value></Value>

<DefaultValue></DefaultValue>

<Access>rw</Access>

</Items>

<Items>

<Name>Device Specific Parameter 2</Name>

<Subindex>0x2</Subindex>

<Datatype>UINT8</Datatype>

<Length>8</Length>

<BitOffset>112</BitOffset>

<Value></Value>

<DefaultValue></DefaultValue>

<Access>rw</Access>

</Items>

<Items>...

* Problem: Mehrere Items in einem Parameter -> Was ist nun eine Parameter -> Definition muss her:
  + Parameter ist ein einzelner Wert eines bestimmten Datentyps
  + -> ein Parameter bei Beckhoff sind manchmal mehrere Parameter -> Parametergruppe
* Access in Beckhoff lautet „ro“ „rw“ „wo“ „ro s“ -> keine Allgemeingültige beschreibung -> Auswertung des ausdrucks in Beckhoff in setzen der entsprechenden properties in DeviceParameter

# 2015-11-17 – Architektur

* <https://msdn.microsoft.com/en-us/library/ee658098.aspx>
* <http://www.codeproject.com/Articles/703634/SOLID-architecture-principles-using-simple-Csharp>
* Zitate zu kosten der Inbetriebnahme (Konfiguration, Programmierung, Parametrierung) aus Feldmann, Spur - 2014 - [Buch] Handbuch Fügen, Handhaben, Montieren

# 2015-11-18 – Speicherung

* Primitive Geräte und der damit verbundene Zuweisungsaufwand machen ein Speichern sinnvoll
* Hierzu müssen die Benutzereingaben zuerst abgespeichert werden und daraus später wieder eine Device Instanz mit Inhalt und richtiger Hierarchieposition erstellt werden, falls der Anwender dies wünscht.
* Allgemeine Herausforderungen bzw. Fragestellungen:
  + 1: Ein Primitive Device wird in der Analyse nicht gefunden, somit reicht es nicht nur die Connections zu speichern, es muss das gesamte Device instanziiert werden
  + 2: Steckkarten, welche Zuweisungsknoten für primitive Geräte darstellen können, sind nicht Teil der Device Hierarchy und tauchen somit nicht in der fertigen Analyse auf. Andere Primitive Devices hingegen haben durchaus ein Parent Device, welches in der Device Hierarchy auftauchen kann (z.B. VentilInsel), deshalb müssen die Eltern abgespeichert werden, wenn die Hierarchie nicht verfälscht werden soll.
  + 3: Wie erkennt man, ob Geräte Primitiv sind, und somit gespeichert werden sollen?
  + 4: In welchem Format werden Informationen gespeichert?
  + 5: Wie bestimmt der Anwender den Modus?
* Lösungsvorschlag 1: **Serialisierung des kompletten Device-Baumes mittels Serializable Attribut**
  + Problem: Das Attribut Serializable ist nicht vererbbar und muss jeder Klasse explizit zugewiesen werden
* Inkompatibilität von Treiberimplementierungen dritter vorprogrammiert.
* Selbst bei lösungseigenen Implementierungen Umsetzung von „Serializable“ schwierig (Verzicht auf XElements, Verzicht auf Events)
* Angesichts dieser gravierenden Einschränkung und des hohen Riskos von Fehlimplementierungen wird im Folgenden nach anderen Lösungen gesucht
* Lösungsvorschlag 2: **Treiberspezifische Hybridlösung -> Wirings automatisch (Serialisierung), DeviceModell und Seriennummer manuell**
  + Die Lösung bestünde darin, den Benutzer weiterhin dazu aufzufordern die Geräte hinzuzufügen. Allerdings wird dem WiringDialog ein Ladeversuch der Connections vorangehen
* Simple Implementierung
* Keine Beeinflussung der Device Implementierung
* Nachteil : Unkomfortable Lösung, da das Eintrage der Seriennummer und des Modells einen erheblichen Zuweisungsaufwand darstellen
* Lösungsvorschlag 3: **Umstrukturierung des Device-Begriffs**
  + Die Lösung bestünde darin, CCDs als vollwertige Devices mit Logik- und VR-Treiber in die Wissensquellen mit aufzunehmen. Daraus ergibt sich die Möglichkeit eine zusätzliche Wissensdatenbank einzuführen, in welcher SubDevices eines gg. Devices gespeichert und verwaltet werden können
* Erheblicher Implementierungsaufwand durch Umstrukturierung des Systems, Implementierung neuer Treiber und erstellen der zusätzlichen Wissensdatenbank
* Gewinn großer Flexibilität in der Verwaltung, welche Geräte gespeichert werden sollen
* Gewinn von zusätzlichen Anlageninformationen auf der Serverebene
* Bessere Kontrolle des Anlagenaufbaus durch Benutzer möglich, da die DeviceHierarchy besser nachvollziebar ist, da sie sich nun nicht mehr an Skills, sondern an physischen Bauteilen orientiert
* Vereinheitlichung aller Anlagenkomponenten zu IDevices
* Möglichkeit der gezielten nachträglichen Beeinflussung von Connections
* Neue Schnittstelle „Save“ in IInformationsSources, wodurch sich jedes IInformationsSources entsprechende Informationen extrahieren kann.
* Lösungsvorschlag 4: **Treiberspezifisches speichern**
  + Speichern der eingegeben Informationen (Model, Serie, Wirings) von Treiber selber
* Mittelmäßiger Aufwand
* Gefahr von doppelter Implementierung derselben Logik in verschiedenen Treibern
* Kompliziertes Umstellen zwischen „Green field“ und „Brown field“ Modus, da den Treibern zusätzliche Parameter übergeben werden müssen.
* Lösungsvorschlag 5: **XmlSerializer**
* Nicht alle Klassen müssen als Serilaizable deklariert werden
* Alle Klassen müsse leer Konstruktoren aufweisen
* Teilweise Konflikte mit Nativen Objekte welche keine Leeren Konstruktoren haben

# 2015-11-18 – Wiring Speicherung

Als vorübergehene und leicht erweiterbare Lösung, wird ein neues IInformation Sources eingeführt, welcher das wiring unter einer Kombination aus Modelnummer und seriennummer abspeichert.

Problematisch ist dies bei Geräten, welche keine Seriennummer haben (Beispielsweise ein Taster) -> Bei solchen Geräten muss entweder weiterhin gewired werden, oder man weißt eine eigene seriennummern (vom Benutzer aus) zu.

Als eindeutiger Speichername wird Modellnummer\_Seriennummer verwendet

* Serialisierung des kompletten Devices wäre auch möglich, allerdings müsse dann jede von IDEvices verwendete Klasse ISerializable implementieren.
* Das Attribute Serializable ist nicht verwendbar, da manche nativen Klassen nicht Serialisierbar sind, diese aber in IDevice Verwendung finden (ich glaube das waren Listen oder so)

# 2015-11-30– Meeting mit Veit

Ausblick: Einführen von Phantomskills, um Frequenzumrichter + Förderband nicht mehr als 1 Blackbox anzusehen, sondern 2 verschiedene Geräte

* Problem mit induktionssensoren: Nicht alle Sensoren unterstützen die funktion der Seriennummer (Siehe email vom hersteller: [Sonstiges\Emailverkehr mit Balluff.pdf](Sonstiges/Emailverkehr%20mit%20Balluff.pdf)
  + Lösung: Verwenden des Fertigungsdatums

# 2015-12-07

Sind die ProductNames bzw. die ProductIds der IoLink Geräte eindeutig? wenn nicht -> Gibt es eine Möglichkeit mit einer Ordnerstruktur Eindeutigkeit zu schaffen?

# 2015-12-08

Wie Identifiziere ich die unterschieldichen Geräte ?:

* IO-Link über Seriennummer Parameter
  + Falls keine SN vorhanden, alternative ?
* Ethercat

Problem im DIM:

* Default Constructors müssen eigentlich ähnlich wie null behandelt werden, da bei zwar neuen IDevices objekten grundsätzlich die Werte zu bevorzugen sind, eventuell default konstruktoren dazu führen,

# 2015-12-09

Nicht vergessen: DLL Loader in BA mit aufnehmen

+ genaue beschreibung der Probleme mit primitives -> Phantomskills etc. -> Ventilinsel als vermittler zwischen iolink port und luftdruckanschluss

Doku + SCHAUBILD für Repository regel

# 2015-12-10

OPC UA macht probleme mit nullable Objekten.

# 2015-12-22

* Strategies : Unbedingt Ablauf mit den Argumenten aufzeigen
* Primitive ?? hab ich vergessen ??
* Template Method Pattern in RepositoryBase benutzt, sowie in StrategyBase

# 2016-01-08 Was ist noch nicht schön gelöst

Beim Implementieren einer Strategy ist aufgefallen:

* Man vergisst den Rückgabewert dem Device zuzuordnen
* Ohne Lesen der Dokumentation ist es nicht ersichtlich, wie die Werterückgabe funktioniert
* Man sollte dies über Events lösen (Über Finished event holt sich das Idevice die subdevices)