PAPAS

PLUG AND PLAY ANTRIEBS- UND STEUERUNGSKONZEPTE
FÜR DIE PRODUKTION VON MORGEN

Forschung für die Produktion von morgen
Schlüsselkomponente Handhabungstechnik

Abschlussbericht

Autoren

Georg Plank, Detlef Reintsema, Dr. Gerhard Grunwald (DLR)

Prof. Dr. Martin Otter, Matthias Kurze, Matthias Löhning, Matthias Reiner (DLR)

Dr. Uwe Zimmermann, Dr. Günter Schreiber (KUKA Roboter)

Dr. Martin Weiss, Rainer Bischoff (KUKA Roboter)

Bruno Fellhauer (Schunk)

Thomas Notheis (TRUMPF Laser)

Thomas Barklage (Lenze Drive Systems)

Projektträger Forschungszentrum Karlsruhe (PTKA)

GEFÖRDERT VOM Bundesministerium für Bildung und Forschung

Inhaltsverzeichnis

Liste der Abbildungen

1 Einleitung

Das Verbundprojekt „Plug And Play Antriebs- und Steuerungskonzepte für die Produktion von morgen“ (PAPAS) wurde vom Bundesministerium für Bildung und Forschung im Rahmen des Handlungsfeldes „Forschung für die Produktion von morgen“ im Themenfeld „Schlüsselkomponente Handhabungstechnik“ in der Zeit von Mai 2003 bis Juli 2006 durchgeführt.

Die zunehmende Integration von Mechanik, Sensorik, Aktorik zu mechatronischen Systemen und deren Steuerung und Regelung durch Software ermöglicht die Entwicklung neuer Handhabungssysteme, die schneller, genauer und intuitiver programmierbar sind. Voraussetzung dafür ist ein integrierter Systementwurfsprozess, der die Modellbildung, Simulation und Optimierung beinhaltet und alle Teildisziplinen der Mechatronik einschließt. Der integrierte Entwurfsprozess ermöglicht es, zukünftige Handhabungssysteme modular aufzubauen und damit Innovationen der Antriebs- und Peripheriehersteller schneller und effizienter in Systeme zu integrieren.

Ziel des Projektes PAPAS war es, diese Entwurfstechnologien Herstellern des Maschinen- und Anlagenbaues sowie deren Komponentenherstellern zugänglich zu machen und basierend auf einem „Plug-And-Play“-Konzept modulare, industrietaugliche Leichtbauroboter für die Produktion von morgen zu realisieren.

Zur Erreichung des Projektziels arbeiten die Projektpartner an drei Schwerpunkten:

1. Auf Kommunikationsebene: „Plug-And-Play“-Technik basierend auf einem offenen Protokoll und auf einem standardisiertem, hochleistungsfähigem Datenbus, sowie dessen Konfiguration, Überwachung und Diagnose.
2. Auf Entwurfsebene: Integrierter mechatronischer Systementwurf von Handhabungssystemen, Anwendung von Methoden Know-how aus der Forschung.
3. Auf Anwendungsebene: Nachgiebige und einfach zu programmierende Produktionsroboter für neue Anwendungen und Märkte.

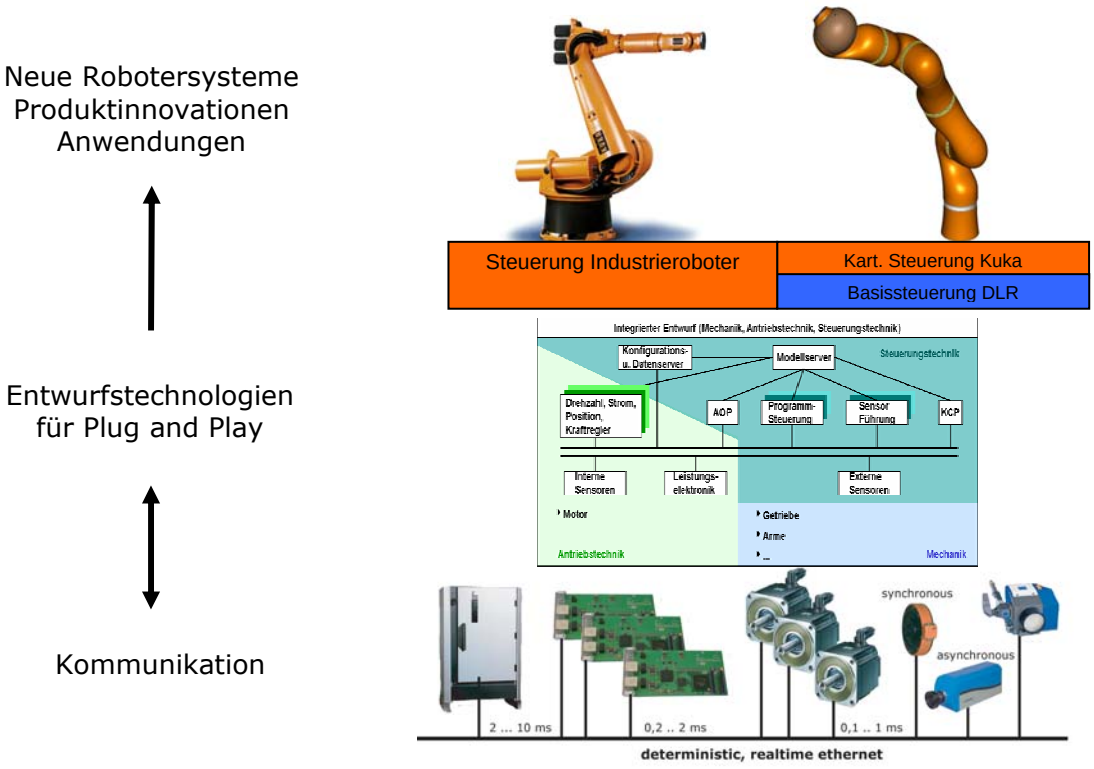


Abb. 1-1 Schwerpunktthemen des Projektes

1.1 Motivation

Ausgangspunkt und Motivation der Arbeiten war der Wunsch aller beteiligten Partner nach einem Technologietransfer der Schlüsselkomponenten bzw. –technologien, die für die Produktion von morgen unabdingbar erschienen:

Erstens ein abgestimmter Systementwurfsprozess, der Modellbildung, Simulation und Optimierung von mechatronischen Komponenten sowie den daraus aufgebauten Systemen umfasst. Zweitens die größtmögliche Integration von Mechanik, Sensorik und Aktuatorik, um möglichst leistungsfähige Systeme, z.B. Leichtbauroboter, aufbauen zu können und drittens die Vernetzung der Komponenten in „Plug-And-Play"-Technik für ein effizientes Konfigurationsmanagement. Darüber hinaus war es ein wesentliches Ziel des Projektes, die technischen Rahmenbedingungen und wirtschaftlichen Erfolgsaussichten für einen industrietauglichen nachgiebigen Leichtbauroboter aufzuzeigen, indem der an eine Industrierobotersteuerung anzubindende DLR-Leichtbauarm Systempartnern zu Test-zwecken zur Verfügung gestellt wurde.

Die globalen Projektziele wurden auf wirtschaftliche und technologische Teilziele herunter gebrochen:

1.1.2 Wirtschaftliche Ziele

- Beschleunigter Innovationsprozess durch durchgängige Modellierbarkeit, Simulation und Optimierung von mechatronischen Systemen
- Reduzierung des Kosten- und Zeitaufwandes zum Entwurf, zur Entwicklung und zur Herstellung von komplexen mechatronischen Komponenten und Systemen durch vollständig modularen Ansatz
- Kostengünstiger, schneller Austausch und Wechsel von Komponenten eines bestehenden Handhabungssystems durch „Plug-And-Play"-Technik
- Leichte Anbindung an und Integration von zusätzlicher Sensorik, Aktuatorik und mechatronischen Komponenten in ein bestehendes Handhabungssystems durch „Plug-And-Play"-Technik
- Erweiterung des Anwenderspektrums von Robotersystemen und wirtschaftliche Fertigung auch bei kleinen Losgrößen und großem Variantenreichtum durch eine neue Generation von flexiblen und intuitiv (da nachgiebig) programmierbaren Handhabungssystemen
- Weite Verbreitung der Ergebnisse durch die Beteiligung eines repräsentativen Spektrums von Unternehmen aus unterschiedlichen Branchen des Maschinen- und Anlagenbaus

1.1.3 Technologische Ziele

- Werkzeuge zum integrierten mechatronischen Systementwurf (geeignet für alle mechatronischen Produkte, nicht nur für Roboter)
- Neuartige Aktuatorik- und Sensorik-Komponenten, leicht konfigurierbar („Plug-And-Play") für neue Systeme (auch über die „klassische" Handhabungstechnik hinaus)
- Kommunikation zwischen und Verwaltung von Aktuatorik- und Sensorik-Komponenten zur leichten Anbindung an und Integration von zusätzlicher Sensorik, Aktuatorik und mechatronischen Komponenten in ein Handhabungssystem
- Nachgiebiger Manipulator als Pilotdemonstrator zur Anwendungs- und Systementwicklung sowie zur Abschätzung des zukünftigen Marktpotentials

1.2 Ausgangspunkte

Das Projekt startete im Mai 2003 zu einem Zeitpunkt, an dem in der Automatisierungsindustrie Diskussionen über neuere, leistungsfähigere Kommunikationssysteme aufkamen. Hauptursache dafür war neben der mangelnden Performance auch die hohen Kosten für die bis dahin aktuellen Feldbussysteme. Neue Komponenten wie z.B. hoch auflösende Sensorik benötigten höhere Bandbreiten und Kommunikationsgeschwindigkeiten.

Digitale Feldbus-Technologien bieten enorme Nutzenpotenziale für die Automatisierungstechnik. Im Idealfall sollte es die digitale Feldbus-Kommunikation den Benutzern ermöglichen, alle intelligenten Feldgeräte und die von ihnen gelieferte Fülle von Informationen problemlos in ihre Steuerungen einzubinden. Moderne Steuerungen umfassen dabei eine

enorme Vielfalt von Einrichtungen wie Sensoren, Antriebe, SPS, sicherheitsrelevante Systeme, ... Die Papas – Technologie standardisiert dabei die Kommunikationsschnittstelle zwischen Feldgeräten und Systemen. Das zentrale Merkmal ist dabei ihre Unabhängigkeit vom Kommunikationsprotokoll und von der Software-Umgebung der Geräte und Systeme.

1.2.1 Kommunikationsebene

Die Anforderungen an elektronische Datenübertragungssysteme im Produktionsumfeld variieren stark. Je nach Einsatzgebiet ergeben sich unterschiedliche Anforderungen an:

- Bandbreite
- Echtzeitfähigkeit / Determinismus
- Maximale Kabellänge
- Stör- bzw. Ausfallsicherheit / Redundanz

Um diese Anforderungen abzudecken, wurde eine Vielzahl unterschiedlicher Kommunikationsmechanismen entwickelt, die sich hinsichtlich der verwendeten Hard- und Software unterscheiden. Die damals und auch heute noch im Einsatz befindlichen Produktionsanlagen müssen oftmals eine breite Palette an unterschiedlichen Kommunikationsmechanismen unterstützen, um allen Anforderungen gerecht zu werden. So bietet eine typische Industrierobotersteuerung standardmäßig bis zu fünf Kommunikationsschnittstellen, die je nach Eignung für unterschiedliche Zwecke eingesetzt werden können:

- Ankopplung von Sensoren (z. B. Initiatoren, Winkelgeber, Bildverarbeitungssysteme)
- Ankopplung von Aktuatoren (z. B. Zusatzachsen)
- Ankopplung an das zentrale Datenarchivierungssystem
- Ankopplung an das Prozessvisualisierungssystem
- Ankopplung an Leit- oder Steuerrechner

1.2.2 Entwurfstechnologien

Der Entwurf von Handhabungsmaschinen und anderen mechatronischen Systemen stützte sich noch auf die getrennte Behandlung der beteiligten Fachgebiete Mechanik, Steuerungs- und Regelungstechnik und Antriebstechnik. In der industriellen Praxis vertraute man jedoch fast ausschließlich auf die Expertise von Spezialprogrammen für die mechanische oder regelungstechnische Auslegung. Zum Teil haben diese Programme schon Module zur Optimierung, deren Funktionalität jedoch eingeschränkt war. Prominente Vertreter dieser Art von Programmsystemen sind Adams, Matlab/Simulink, Ansys und Spice.

Die ersten Schritte hin zu einer Integration und Kopplung im Bereich mechatronischer Systeme wurden bereits auf mehreren Ebenen unternommen. Im Esprit Projekt ODESIM (Optimum design of Multi-Body Systems) wurden Software-Tools für ein optimiertes Design der mechanischen Struktur entwickelt. Dabei kommen in Verknüpfung von CAD und Mehrkörpermodellierung sowohl interaktive als auch vollautomatische Optimierungsbasierte Design-Methoden zur Anwendung.
Einen Schritt weiter ging man im EU Projekt RealSim (Realtime Simulation for Multi-Physics Systems). Dort wurde nach Ideen des objektorientierten Multi-Domain-Modeling die Integration von mechanischen, elektrischen und regelungstechnischen Aspekten schon auf Modellbildungsebene vorgenommen. Auf diese Weise wird das gesamte mechatronische Systemmodell einer einheitlichen mathematischen Behandlung durch Werkzeuge wie Simulatoren, Optimierern usw. zugänglich gemacht. Eine Transformation auf symbolischer Ebene stellt die Verbindung zu einer Echtzeit-Hardware in einer „in the loop-Simulation" dar.
Sehr erfolgreich im Einsatz für den Entwurf von Regelungssystemen sind modellbasierte numerische Optimierungsverfahren. Das eigentliche mechatronische System wird aber meist nicht oder nur sehr eingeschränkt als freier Parameter in die Optimierung miteinbezogen und wird dementsprechend bei der Problemlösung als fest vorgegeben behandelt.

2 „Plug-And-Play" Kommunikation für die Handhabungs- und Automatisierungstechnik

Aus Sicht von PAPAS zeichnen sich zukunftsweisende Anwendungen in der Antriebs- und Steuerungstechnik für die Produktion von Morgen durch eine größtmögliche Integration von Mechanik, Aktuatorik und Sensorik aus, wobei gerade der Einsatz innovativer „Plug-And-Play"-Ansätze auf der Kommunikationsebene die Flexibilität und Produktivität von Anwendungen erheblich beeinflussen wird. Heutzutage werden Roboterarbeitszellen für die jeweilige Anwendung statisch programmiert (dedizierte/spezifischen Komponenten). Bei einem Komponentenausfall, der durch den Austausch mit einer gleichwertigen (z.B. unterschiedliche Geometrie) Komponente kompensiert wird, müssen die gerätespezifische Teile der Anwendung überarbeitet und neu erstellt werden. Mit dem Einsatz neuartiger „Plug-And-Play"-Technologien in der Automationstechnik, wie im Rahmen von PAPAS untersucht, kann die Stillstandszeit der Anlage erheblich reduziert werden. Idealerweise sehen derartige Ansätze eine automatische Re-Konfiguration der Robotersteuerung vor, was die standardisierte Bereitstellung von Komponentendaten voraussetzt.

Der Begriff „Plug-And-Play" stammt aus dem Englischen und bedeutet soviel wie „Anschließen und loslegen". Er kommt aus dem Gebiet der Computertechnologie, mit dem man die Eigenschaft eines Computers beschreibt, neue Geräte –meist Peripheriegeräte– anschließen zu können, ohne anschließend Treiber zu installieren oder andere Einstellungen vornehmen zu müssen. Ein weit verbreitetes und sehr anschauliches Beispiel für „Plug-And-Play" ist der auf USB basierende Memory-Stick. Man nimmt dieses Gerät, steckt es in einen freien USB-Slot, der Computer erkennt autonom, um welches Gerät es sich handelt, und leitet entsprechende weitere Aktionen wie z.B. das Öffnen eines Dateimanagers ein.

Dieses aus dem Office-Bereich stammende Prinzip wird im Rahmen von PAPAS in die Automatisierungs- und Handhabungstechnik überführt. Ziel ist es, dass es für einen Anwender immer einfacher wird, neue Geräte in seine jeweilige Umgebung einzubinden, ohne dafür ein Expertenwissen haben zu müssen.

2.1 Formen von „Plug-And-Play"

Das Konzept kann aber nur dann in der Handhabungs- und Automatisierungstechnik übernommen werden, wenn deren spezifischen Anforderungen und Rahmenbedingungen berücksichtigt werden. Betrachtet man den Vorgang des „Plug-And-Plays" genauer, so kann man drei „Plug-And-Play"-Ebenen ausmachen:

1. „Plug-And-Play" für die Kommunikations- bzw. Bussystemebene, in der das An- und Abmelden von Kommunikationsteilnehmern stattfindet;
2. „Plug-And-Play" für die Steuerungs- bzw. Betriebssystemebene, in der die Komponenten in die eigene Umgebung eingebunden bzw. ausgelöst werden;
3. „Plug-And-Play" für die Anwendungsebene, in der die Funktionen der Teilnehmer auch tatsächlich genutzt werden.

Innerhalb einer Ebene kann man darüber hinaus zwischen verschiedenen „Plug-And-Play" Varianten und „Plug-And-Play" Stufen unterscheiden.

2.1.2 „Plug-And-Play" aus Sicht der Anwendung

Anhand prototypischer Szenarien aus der Automatisierungs- und Handhabungstechnik wurden unterschiedliche Anwendungen eines „Plug-And-Plays" herausgearbeitet.

Beim Systemanlauf kommt es darauf an, die Anwendung mit möglichst geringem Konfigurationsaufwand in Betrieb zu setzen. Unabhängig vom Hersteller sollen Geräte unterschiedlicher Geräteklassen untereinander Kommunikationsbeziehungen aufbauen und die für die Anwendung erforderliche Kommunikation möglichst automatisch aufnehmen. Es sollen da-

durch Fehlkonfigurierungen vermieden und Inbetriebnahmezeiten entscheidend verkürzt werden.

Beim Gerätetausch kommt es darauf an, defekte Geräte möglichst schnell durch Ersatzgeräte auszutauschen. Das Ersatzgerät sollte nicht vom gleichen Hersteller sein müssen. Es sollte jedes Gerät des gleichen Gerätetyps einsetzbar sein können. Das Gerät sollte möglichst ohne Konfiguration und ohne Programmierung die Aufgabe des defekten Gerätes übernehmen können. Damit können Stillstandszeiten verkürzt und somit die Verfügbarkeit der Anlage erhöht werden.

Bei der Diagnose geht es darum, entsprechende Geräte in das System integrieren zu können, ohne den Regelbetrieb zu beeinflussen. Ohne Neuanlauf muss es den Diagnosegeräten möglich sein, alle erforderlichen Systeminformationen zu erfassen. Damit können z. B. Wartungsprozesse unterstützt werden, ohne die Fertigung zu unterbrechen.

Im Folgenden werden die betrachteten Szenarien kurz vorgestellt, die durch die PAPAS „Plug-And-Play"-Spezifikation abgedeckt werden sollen. Die Szenarien beziehen sich auf einen Roboter, an den Geräten wie

- zusätzliche Antriebe,
- Greifer,
- Kameras,
- Kraft-Momenten-Sensoren,
- Programmierbare Fokussieroptiken und
- Regler (als Teil eines modularen Steuerungskonzeptes)

angeschlossen werden können.

Szenario 1: Lastdatenermittlung
Die dynamischen Daten (Masse, Schwerpunkt, Trägheitstensor) eines am Flansch des Roboters befestigten Gerätes werden durch entsprechende Bewegung des Gerätes ermittelt. Die ermittelten Daten sollen im Geräte gespeichert werden.
Der Anschluss eines entsprechenden Gerätes ist dem System anzuzeigen, einschließlich der Information zur Geräteklasse, zum Gerätetyp bzw. zu den relevanten Geräteeigenschaften (Geräte-Parametersatz). Die Mechanismen zum Abspeichern der Lastdaten im Gerät sowie zur Bereitstellung der Daten durch das Gerät bei einem Wiederanlauf sind zu spezifizieren.

Szenario 2: Anschluss von Antrieben oder der Programmierbaren Fokussieroptik (Vorkonfigurierte Geräte)
Zusätzliche Antriebe (evtl. mit Getriebe) für einen Hubtisch oder eine Programmierbare Fokussieroptik (PFO) für eine Laserschweiß-Anwendung sind zu konfigurieren. Diese Geräte sind durch eine hohe Flexibilität gekennzeichnet und erfordern dadurch die Einstellung von einer Vielzahl von Parametern. Für die Konfiguration der Geräte wird eine gerätespezifische Software auf einem externen PC genutzt. Bei der Verbindung der Geräte mit der Robotersteuerung wird deren Konfiguration an die Steuerung übertragen.
Die Konfiguration von Geräten die dauerhafte Speicherung der Daten muss gewährleistet werden. Geräteklasse, Gerätetyp und die konfigurierte Anwendung müssen identifizierbar sein.

Szenario 3: Anschluss von Kameras und/oder externen Kraft-Momenten-Sensoren (Geräte mit Funktionsbausteinen)
Die Geräte sind durch Funktionsbausteine gekennzeichnet, die zur Realisierung der Anwendung nicht in den Geräten selbst, sondern in der Steuerung ablaufen. Die anfallenden Daten können dabei in rohem oder aber auch vorverarbeiteten Zustand sein. Die Inbetriebnahme der Geräte beinhaltet neben der Konfiguration der Geräte auch die Auswahl der passenden Funktionalität, die in der Steuerung ablaufen soll, und deren Konfiguration.
Neben der Identifikation des Gerätetyps sind Informationen über die externen Funktionsbausteine (Treiber) erforderlich. Nach Anschluss der Geräte ist die Treiberauswahl und die konsistente Konfiguration von Gerät und Treiber zu unterstützen.

Szenario 4: Gerätetausch
Ein Gerät kann aufgrund eines Defektes durch ein anderes (nicht zwangsläufig identisches) ersetzt werden. Die Steuerung muss erkennen, ob das Ersatzgerät in der Lage ist, die Funk-

tionalität des zuvor angeschlossenen Gerätes zu übernehmen. Das Ersatzgerät muss entsprechend der Funktion in der Anwendung konfiguriert werden.
Der Ausfall eines Gerätes muss angezeigt werden, wobei Informationen über Gerätetyp und Anwendungsfunktion bereitzustellen sind. Der Anschluss des neuen Gerätes ist zu erkennen und der Gerätetyp bzw. die relevanten Parameter zu identifizieren. Die Übertragung der Konfiguration von der Steuerung zum Gerät ist zu unterstützen.

Szenario 5: Greiferwechsel
Der Roboter nutzt Greifer mit unterschiedlichen dynamischen Daten und unterschiedlichen Tool-Center-Points (TCP). Ein Gegenstand soll durch verschiedene (zufällig angeflanschte) Greifer adäquat gegriffen werden können. Die Robotersteuerung ist hierfür beim Anschluss des Greifers über die entsprechenden Daten zu informieren. Die automatische Konfiguration der Robotersteuerung ist zu unterstützen.

Szenario 6: Alternative Geräte
An den Roboterarm werden völlig verschiedene Werkzeuge für unterschiedliche Bearbeitungsaufgaben angebaut. Beim Anschluss des Gerätes sind Informationen zur Anwendung und zum Gerätetyp zwischen der Robotersteuerung und dem Werkzeug auszutauschen.

Szenario 7: Diagnose
Zur Geräte- und Systemdiagnose kann ein Standard-PC (z. B. Laptop) mit dem System verbunden werden. Es kann die Kommunikation verfolgt, Konfigurationsdaten geändert und sogar Geräte gesteuert werden.
Dem Diagnosetool müssen alle Informationen über Gerätekonfigurationen sowie über Geräte- und Systemzustände zugänglich gemacht werden. Weiterhin muss das Diagnosetool über Informationen zu den relevanten Geräte- und Anwendungsprofilen verfügen.

Szenario 8: Reglerauslegung für Roboter mit Rapid-Prototyping
Für die einfachere, schnellere und weniger fehlerträchtige Entwicklung neuer, aufwendigerer Reglerstrukturen wird ein Rapid-Prototyping realisiert, wobei die Algorithmen statt auf einem Digital-Signal-Processor (DSP) auf einem Standard-PC ablaufen. Zu diesem Zwecke wird der PC, auf dem auch die Algorithmen zur Steuerung des Testablaufes abgearbeitet werden, an das System angeschlossen.
Der Rapid-Prototyping-PC wird als solcher im System erkannt und kann Aufgaben des Anwendungsmasters, einschließlich der Netzwerksmanagementaufgaben, übernehmen.

Szenario 9: Anschluss von nicht projektierten Anwendungen
Um das System für zukünftige Erweiterungen offen zu halten, sollen auch Anwendungen in der PAPAS-"Plug-And-Play"-Spezifikation berücksichtigt werden, die bei der Projektierung noch nicht bekannt sind. Das Hinzufügen entsprechender Geräte ist zu erkennen, Anwendung und Parameterwerte zu identifizieren. Die Einordnung der Anwendung muss durch Nutzerinteraktion erfolgen.

2.1.2.1 „Plug-And-Play" Varianten

Aus den oben aufgeführten Szenarien lässt sich erkennen, dass es unterschiedliche Varianten des Konzeptes „Plug-And-Play" gibt. Sie lassen sich folgendermaßen klassifizieren:

Cold Connect:

Die Geräte werden zunächst im ausgeschalteten Zustand miteinander verbunden. Anschließend werden sie nacheinander oder gleichzeitig eingeschaltet. Die Geräte können dabei vorkonfiguriert sein oder während des Systemanlaufs durch entsprechende Werkzeuge konfiguriert werden. Der Systemanlauf kann zum einen vollautomatisch erfolgen oder für einzelne Geräte eine Nutzerinteraktion erfordern.

Hot Connect:

Ein Gerät wird zu einem bereits laufenden System hinzugefügt. Die Gerätekonfiguration wurde entweder bereits offline festgelegt oder erfolgt automatisch während der Anlaufphase. In keinem Fall darf die laufende Anwendung in ihrem betriebsgemäßen Ablauf beeinflusst werden. Im Einzelnen:

- ein Gerät im eingeschalteten Zustand hinzufügen oder entfernen

- auf dem Bus zu diesem Zeitpunkt zirkulierende Telegramme dürfen dabei verloren gehen
- Teilnehmer am Bus dürfen temporären Störzustand einnehmen
- Teilnehmer und Bus müssen sich selbständig regenerieren können
- Anwendung mit Buszugriff darf durch Störung nicht beeinträchtigt werden

Coordinated Connect:

Hierbei handelt es sich um eine Variante von „hot connect", welche die Zufälligkeit des Ein- bzw. Aussteckens eines Teilnehmers ausschließt. Wie oben beschrieben können durch das Ein- bzw. Aushängen von Kommunikationsteilnehmer Telgramme verloren gehen, was die Systemstabilität beeinträchtigt. Dies kann z.B. bei hochdynamischen Anwendungen zu Gefahrensituationen führen, die unbedingt vermieden werden müssen. Aus diesem Grund werden die Kommunikationsteilnehmer durch Benutzer und/oder programmgesteuert koordiniert hinzugefügt oder abgemeldet

2.1.2.2 „Plug-And-Play"-Stufen

Der Grad der Autonomie variiert in den unterschiedlichen Szenarien. Der Vollständigkeit halber werden diese drei Stufen ebenfalls vorgestellt:

- Vollständiges „Plug-And-Play":
  Wenn ein Gerät nach dem Anlauf kommunikationsfähig ist, wird der Gerätetyp vom System erkannt und das Gerät wenn erforderlich automatisch konfiguriert. Dazu gehört auch die automatische Auswahl, Konfiguration und Einbindung eines Treibers im Anwendungsmaster. Ist das Gerät dem System bereits bekannt, kann die Konfiguration von Gerät und Treiber entfallen, da die Daten nichtflüchtig gespeichert sind. Das Gerät steht der Anwendung zur Verfügung.
- Halbautomatisches „Plug-And-Play":
  Wenn ein Gerät nach dem Anlauf kommunikationsfähig ist, wird der Gerätetyp vom System erkannt und das Gerät wenn erforderlich automatisch konfiguriert. Die Auswahl eines Treibers im Anwendungsmaster erfordert eine Nutzerinteraktion. Die Konfiguration und Einbindung des Treibers wird automatisch ausgeführt.
- Konfigurierbares „Plug-And-Play":
  Wenn ein Gerät nach dem Anlauf kommunikationsfähig ist, wird der Gerätetyp vom System erkannt. Die Konfiguration des Gerätes bzw. die Auswahl, Konfiguration und Einbindung eines Treibers im Anwendungsmaster erfordert eine Nutzerinteraktion.

2.1.3 „Plug-And-Play" aus Kommunikationssicht

Neue, hochleistungsfähige Kommunikationslösungen in der Automatisierungstechnik müssen unabhängig von der hier betrachteten „Plug-And-Play"-Fähigkeit unterschiedlichste Anforderungen erfüllen, die durch folgende Eigenschaften charakterisiert sind:

- Verteilte Automatisierung: räumlich verteilte Einheiten arbeiten gemeinsam an einer Aufgabe.
- Zusammenschaltung von Geräte mit unterschiedlichen Funktionen und Zeit- und Datenmengenanforderungen in einem Netzwerkverbund.
- Sehr hohe Taktzahl: Maschinen und Anlagen werden immer schneller, die notwendigen Steuerungsaufgaben müssen in kürzeren Zeiteinheiten erledigt werden.
- Sehr hohe Datenmengen: Aufgaben werden komplexer und umfangreicher, daraus resultieren große Datenmengen die zwischen den beteiligten Geräten ausgetauscht werden müssen.
- Reduktion der Kosten.
- Reaktionszeit auf unvorhersehbare Ereignisse
- Weitestgehende Verwendung von standardisierter, bewährter und damit preiswerter Technologie.
- Wartung oder Erweiterung der Anlage: Idealerweise sollen beliebige Geräte im laufenden Betrieb entfernt oder angeschlossen werden können, ohne dass die Funktion des Gesamtsystems gestört wird.

Die meisten der oben angeführten Bedingungen werden von den auf dem Markt verfügbaren „Echtzeit" - Ethernet Lösungen erfüllt. Da deren echtzeittechnischen Belange hinlänglich bekannt sind, wird hier im Besonderen auf die in PAPAS interessante Frage eingegangen:

> Wie beeinflusst die Topologie des Netzwerkes, sowie die physikalische Verbindung (Busanschaltung) der Netzteilnehmer die „Plug-And-Play"-Fähigkeiten eines Netzwerkes und welche „Plug-And-Play"-Varianten sind realisierbar?

In der Kommunikationstechnik existieren verschiedene Topologien wie Stern, Ring, Baum oder Bus, um Komponenten miteinander zu verbinden (vgl. Abb).

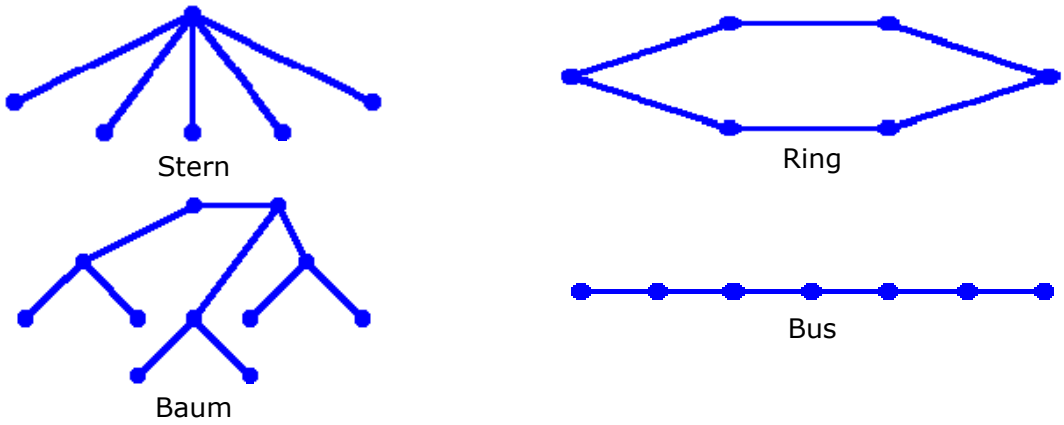


Abb. 2-1 Topologien

Unter einer Netzwerk-Topologie versteht man die Anordnung von Netzwerk-Knoten und Kabeln. Sie bestimmt die einzusetzende Hardware, sowie die Zugriffsmethoden auf das Medium.

Die **Ring-Topologie** (z. B.SERCOS) ist eine geschlossene Kabelstrecke in der die Netzwerk-Knoten im Kreis angeordnet sind. Das bedeutet, dass an jedem Knoten ein Kabel ankommt und ein Kabel abgeht. Der Ring enthält keinerlei aktive Netzwerk-Komponenten. Die Steuerung und den Zugriff auf das Übertragungsmedium regelt ein Protokoll, an das sich alle Knoten halten. Wird die Kabelverbindung an einer Stelle unterbrochen um z. B. neue Knoten zu integrieren oder einen defekten Teilnehmer auszutauschen, fällt das Netzwerk komplett aus.

Die **Bus-Topologie** besteht aus mehreren Knoten, die hintereinander oder nebeneinander in Reihe angeordnet sind, wobei die Knoten über eine gemeinsame Leitung miteinander verbunden sind. Typische Vertreter dieser Topologievariante sind die heute auf den Markt verbreiteten Feldbusse wie z. B. CAN, Profibus, Interbus usw. sowie die aus der Office-Welt bekannten Standard-Ethernet Techniken 10Base5 und 10Base2 (Cheapernet). Alle Knoten haben einen gleichberechtigten Zugriff auf das Übertragungsmedium und die Daten, die darüber übertragen werden. Durch ein spezielles Zugriffsverfahren wie z. B. CSMA/CD kann jeder Knoten als Datenquelle auf dem Bus agieren.
Ein Hinzufügen oder Abstecken oder Austauschen von Busteilnehmern im laufenden Betrieb ist meist mit einem Teilausfall des Netzes verbunden, da diese Aktionen eine Auftrennung des Kabels erfordern.

Die **Stern-Topologie** enthält eine zentrale Instanz, die eine Verbindung zu allen im Netzwerk beteiligten Knoten unterhält. Jeder Knoten ist über eine eigene physikalische Leitung an die Zentrale angebunden. Der Zentralknoten (Hub oder Switch) übernimmt dabei die Verteilung der Datenpakete, da alle Netzverbindungen über Ihn laufen. Die Stern-Topologie hat sich in den letzten Jahren als Industriestandard (Ethernet 100BaseTx) in der Bürokommunikation durchgesetzt.
Während eine defekte Zentral-Komponente zum Ausfall des Netzwerkes führt, ist andererseits die sternförmige Struktur eine ideale Voraussetzung für ein problemloses Hinzufügen und Entfernen von Teilnehmern, ohne dass der Rest des Netzwerkes davon beeinflusst wird.

Als man vor Jahrzehnten in der Automatisierungstechnik die Feldbusse eingeführt hatte, war eines der Argumente die Elimination der sternförmigen Verdrahtung der Maschinen und Anlagen mit einer busförmigen Topologie. Konträr zu dieser Tatsache steht die sternförmige Ethernetverdrahtung aus der Bürowelt. Um hier nicht einen Rückschritt einzugehen, müssen für industrielles Ethernet neue Topologien eingeführt werden.

Die **baumförmige Netzwerktopologie** ist ein guter Kompromiss aus Stern- und Busstruktur, da durch ein Hinzufügen, Abstecken oder Austauschen von Busteilnehmern im laufenden Betrieb meist nur der Teilnehmer selbst oder nur einen kleiner Teil des Netzes beeinflusst wird. Durch den Verzicht auf eine zentrale Netzwerk-Komponente müssen alle Netzwerkknoten mit einer entsprechenden Funktionalität ausgestattet werden – man erkauft sich also die einfachere Verdrahtung durch eine komplexere Busanschaltung. ETHERNET Powerlink, EtherCAT und Profinet IRT, um die wichtigsten Vorschläge zu nennen, sind Vertreter der baumförmigen Netzwerktopologie.

2.1.3.1 Busanschaltung

Einen wesentlichen Anteil an der „Plug-And-Play"-Fähigkeit eines Netzwerkes hat die Busanschaltung der einzelnen Geräte. Sie beeinflusst in großem Maße welche „Plug-And-Play"-Varianten realisierbar sind. Da sich fast alle Anschaltungen auf eine Hub- oder Switch -Technik zurückführen lassen, wird deren Funktionsweise kurz erklärt.

Hubs arbeiten auf der Bitübertragungsschicht (Schicht 1) des OSI-Modells. Sie haben eine reine Verteilfunktion und dienen lediglich der Verstärkung der elektrischen Signale. Alle angeschlossen Knoten teilen sich also die gesamte Bandbreite die durch den Hub zu Verfügung steht (z. B. 100 MBit/s). Ein Hub empfängt ein Datenpaket und sendet es an alle anderen Ports. Diese Technik ist bezüglich Latenzzeiten besonders effektiv, birgt aber den Nachteil, dass beschädigte Datenpakete, wie sie z. b. durch „Plug-And-Play"-Vorgänge entstehen können, auf alle Netzteilnehmer weitergeleitet werden und somit das gesamte Netzwerk stören können. Um diesen Nachteil zu eliminieren empfiehlt es sich Busanschaltungen (Switch) einzusetzen, die auf der Sicherungsschicht (Schicht 2) des OSI-Modells arbeiten.

Ein Switch schaltet direkte Verbindungen zwischen den angeschlossenen Geräten und stellt außerdem die gesamte Bandbreite des Netzwerkes auf dem gesamten Kommunikationsweg zur Verfügung. Beschädigte Datenpakete, wie sie durch „Plug-And-Play"-Aktivitäten entstehen können, werden erkannt und nicht an andere Netzwerksegmente weitergeleitet. Eine Störung des gesamten Netzwerkes ist daher ausgeschlossen.

2.1.3.2 Vergleich der 4 Grundtopologien

Topologie / Struktur	Vorteile	Nachteile
Ringstruktur	• verteilte Steuerung • große Netzausdehnung	• aufwendige Fehlersuche • bei Störungen Netzausfall • hoher Verkabelungsaufwand • nur Cold „Plug-And-Play" sinnvoll
Busstruktur	• einfach installierbar • einfach erweiterbar • kurze Leitungen	• Netzausdehnung begrenzt • bei Kabelbruch fällt Netz aus • aufwändige Zugriffsmethoden • nur Cold „Plug-And-Play" sinnvoll

Sternstruktur	• einfache Vernetzung • einfache Erweiterung • hohe Ausfallsicherheit	• hoher Verkabelungsaufwand • Netzausfall bei Ausfall der Zentralkomponente • Hot „Plug-And-Play" sowie Coordinated „Plug-And-Play" möglich • abhängig von Busanschaltung
Baumstruktur	• dezentrale Steuerung • „unendliche" Netzausdehnung • hohe Ausfallsicherheit	• aufwendige Administration • teuere Vernetzung • Linienstruktur möglich • Hot „Plug-And-Play" sowie Coordinated „Plug-And-Play" möglich • abhängig von Busanschaltung

Ethernet (IEEE 802.3) hat sich aus mehreren Gründen als zukünftiges Übertragungsmedium in der Automatisierungstechnik etabliert. Ausgereifte Technologie mit hohen Datenraten und kostengünstige Installation sowie vertikale Durchgängigkeit von der Planungs- bis zur Sensorschicht sind hier die Schlagworte. Allerdings bereitet die durchgängige Verwendung bis zum letzten Sensor Probleme. Hier sind oftmalig harte Echtzeitanforderungen zu erfüllen. Nicht selten werden zwischen den Netzwerkknoten Synchronisiergenauigkeiten von 1 µs und Updatezeiten von nur wenigen hundert Mikrosekunden gefordert. Trotz der immens hohen Datenraten stößt Ethernet hier an seine Grenzen. Der Grund: das Kanalzugriffsverfahren das Kollisionen von Datenframes zulässt und diese nach einem stochastischen Verfahren auflöst. Eine Zustellung von Daten innerhalb des geforderten Zeitrahmens kann so nicht garantiert werden.

Neue Ansätze wie Ethernet Powerlink, EtherCAT, SERCOSIII und Profinet IRT, um die Wichtigsten zu nennen, beheben dieses Defizit. Die Protokolle setzen alle auf Standard Ethernet (100BasTx) auf und erfüllen harte Echtzeitanforderungen. Je nach Netzwerktopologie und Busanschaltungstyp (Hub <-> Switch) unterstützen sie verschiedene „Plug-And-Play"-Varianten auf Kommunikationsebene.

Gleichzeitig zum Echtzeit-Datenverkehr bleibt reservierte Bandbreite für spontane Kommunikation, z.B. TCP/IP. Somit ist es möglich geschützte Netzbereiche zu schaffen innerhalb derer einerseits deterministischer Datenaustausch (zyklisch) andererseits aber auch durchgängige Kommunikation (azyklisch) mittels TCP/IP möglich ist.

2.2 Busunabhängiges „Plug-And-Play"

Alle derzeit kommerziell verfügbaren und echtzeitfähigen seriellen Bussysteme sind mit jeweils unterschiedlicher Ausprägung für das Konzept des „Plug-And-Plays" geeignet. Sie unterscheiden sich hinsichtlich unterschiedlicher Übertragungstechniken (Sercos II ↔ Ethernet) aber auch durch unterschiedliche Protokolle (vgl. EthernetPowerLink ↔ EtherCAT). Diese Umstände machen es einem potentiellen Anwender nicht einfach Geräte, die über verschiedene Bussysteme angesprochen werden, zu einer Applikation zu kombinieren. Ziel von PAPAS war es daher dem Anwender eine einheitliche Schnittstelle zur Verfügung zu stellen, um die gesamte Prozessperipherie erreichen zu können. Idealerweise sollen dabei die notwendigen Komponenten wie z.B. Geräte und Bus Controller automatisch per „Plug-And-Play" in die Steuerung integriert werden können.

Der nachfolgende Abschnitt beschreibt ein Softwarekonzept, welches die geforderte „Plug-And-Play" Funktionalität vom Bussystem abstrahiert und so einen transparenten, deterministischen Zugriff auf die Busteilnehmer bereitstellt.

Die Kommunikation in PAPAS setzt wegen der notwendigen festen Kopplung der einzelnen Komponenten eine Master-Slave Architektur voraus. Das einfache Herstellen einer Verbindung zwischen dem Master-Node z. B. einer Steuerung und einem oder mehreren Slave-Nodes (z. B. Antriebe), bewirkt im PAPAS-System eine Vielzahl an Interaktionen zwischen

unterschiedlichen Soft- und Hardwareschichten. Das PAPAS-Schichtenmodell beschreibt die wichtigsten Komponenten und deren Datenfluss:

- Slave – Node: Ein Gerät das die gewünschte Endbenutzerfunktion bereitstellt;
- Funktion: Beschreibung der Funktionalität, die das Gerät den jeweiligen Anwendungen zur Verfügung stellt (Z.B. Messergebnis eines Sensors)
- Logisches Gerät: Die Funktionalitäten, die ein Gerät zur Verfügung stellen kann, können sehr stark variieren. Das Logische Gerät stellt dem Master unabhängig davon eine einheitliche Basisschnittstelle zur Verfügung. Damit wird der Master in die Lage versetzt, die PAPAS-relevanten Aspekte auch unterschiedlicher Geräte auf eine identische Art und Weise zu verwalten.
- Client Software: Zum Gerät gehörende Treibersoftware, welche die Anwendung unterstützt;
- PAPAS System Software: Software die das jeweilige Betriebssystem als PAPAS–Unterstützung bereitstellt. Sie ist sowohl von der Client-Software als auch den Geräten unabhängig.
- Bus Interface: Bus Interface Hardware (z. B. EtherCAT- oder EthernetPowerLink – Controller etc.).

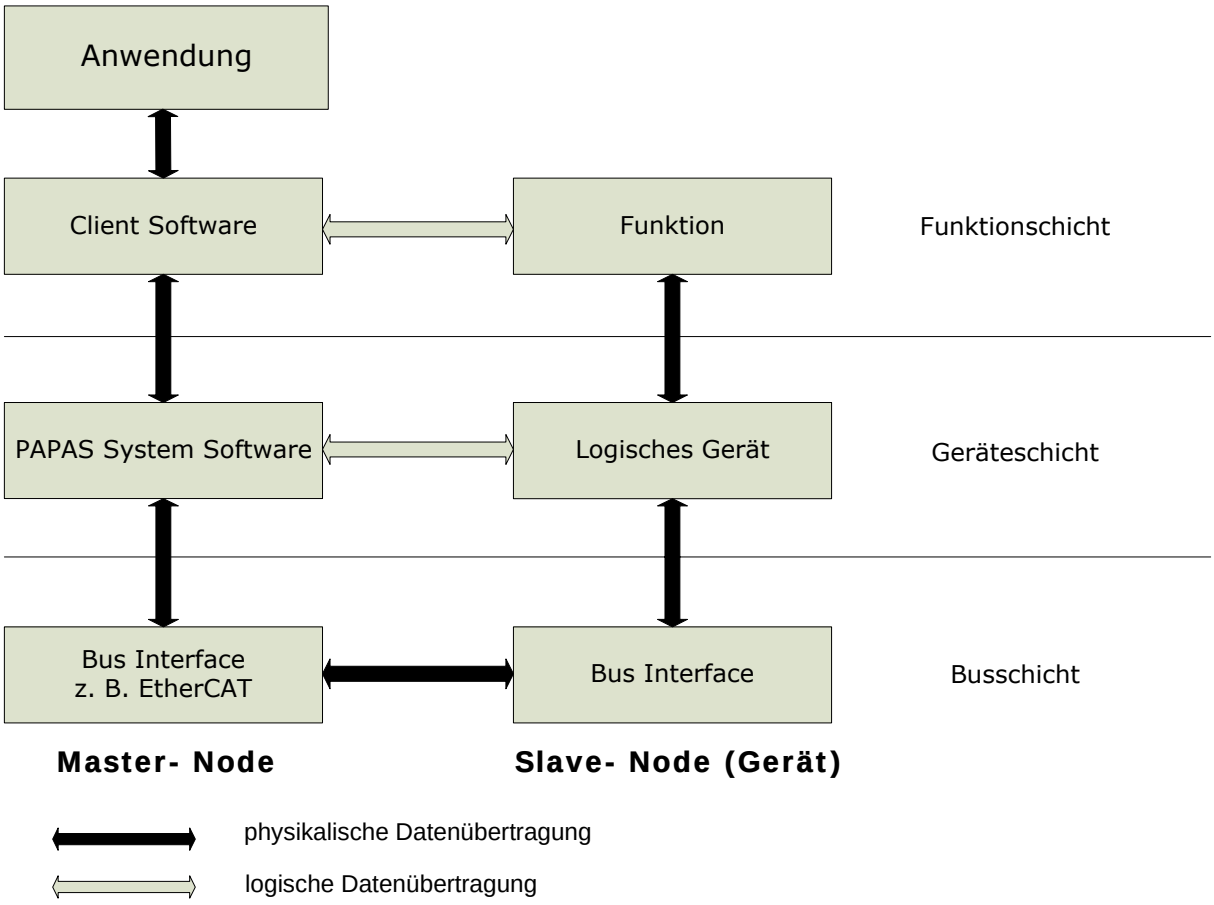


Abb. 2-2 PAPAS - Schichtenmodell

Master und Slave setzen sich jeweils aus drei Schichten zusammen. Die dunklen Pfeile beschreiben den tatsächlichen, physikalischen Datenfluss. Der logische Datenfluss (helle Pfeile) beinhaltet das Protokoll zum Austausch schichtenrelevanter Informationen. Die Busschicht stellt die physikalische, signaltechnische und paketorientierte Verbindung bereit. Auf Geräteschichtebene werden z. B. Systeminformationen, die der Gerätekonfiguration dienen, ausgetauscht. Die Funktionsschicht beschreibt den logischen Zusammenhang zwischen Master und Slave. Alle Anforderungen der Client-Software, die im Zusammenspiel mit der Client-Funktion erforderlich sind, werden über die unterlagerten Schichten und deren Schnittstellen übertragen.

2.2.1 Das PAPAS-Protokoll

PAPAS arbeitet nach dem Master/Slave - Prinzip. Der Master setzt einen Request auf den Bus ab. Stehen die erforderlichen Ressourcen zur Verfügung sendet entweder der Master oder der selektierte Slave seine Daten. Daran schließt sich die Acknowledge-Phase an, die die erfolgreiche Übertragung der Daten signalisiert. Das PAPAS-Protokoll definiert vier Transfer-Modi für Daten (*Control, Interrupt, Bulk und Isochron*), die vom eingesetzten Bussystem bereitgestellt werden. Alle Slave–Nodes stellen einen gemeinsamen Satz an Device Requests und Deskriptoren zur Verfügung. Das PAPAS-Protokoll benutzt die Transfer-Modi auch als Transportcontainer für die Device-Requests. Die Deskriptoren beschreiben die Slave–Node -Eigenschaften.

2.2.1.1 Transfer-Modi

- *Control Transfer*: Dieser Transfertyp wird von der PAPAS- Software zur Gerätesteuerung, bzw. zur Konfiguration nach Anschluss eines neuen Gerätes verwendet. Alle Geräte unterstützen diesen Transfertyp. Je nach Gerätetyp steht zusätzlich mindestens ein weiterer Transfer-Modus zur Verfügung. Das Protokoll für den Control Transfer gewährleistet eine doppelte Fehlersicherung.
- *Interrupt Transfer*: Der Interrupt-Transfer ist gedacht für die asynchrone Übertragung kleiner Datenmengen. Dabei garantiert das Protokoll, dass die Daten mindestens einmal in einem gerätespezifisch definierten Service-Intervall ausgetauscht werden. Bei auftretenden Busfehlern wird die Übertragung im nächsten Intervall wiederholt. Typischerweise arbeiten z.B. I/O-Klemmen im Interrupt-Modus.
- *Bulk Transfer*: Der Bulk-Transfer wird vor allem von Geräten genutzt, welche große, jedoch zeit-unkritische Datenmengen zu übertragen haben. In diesem Modus garantiert das Protokoll zwar den Transport der Daten, sobald entsprechende Bandbreiten vorhanden sind, aber es gibt weder ein maximales Service-Intervall, noch feste Datenraten. Ist momentan keine ausreichende Bandbreite verfügbar, wird die Anfrage zu einem späterem Zeitpunkt wiederholt. Bei Busfehlern muss ebenso wie im Interrupt-Modus eine erneute Übertragung versucht werden.
- *Isochroner Transfer*: Der Isochron-Transfer (periodische Übertragung) erlaubt im Gegensatz zu den anderen Modi nicht nur die Garantie fester Übertragungsraten, sondern auch einer maximalen Latenzzeit. Bereits bei Geräteanschluss prüft der Master, ob die vom Gerät geforderte Bandbreite zugeteilt werden kann. Sollte dies nicht der Fall sein, werden jedoch keine bestehenden Verbindungen beendet. Die Anfrage muss wiederholt werden. Eine Wiederholung gibt es korrekt übertragenen Daten nicht. Eine solche nachträgliche Fehlerkorrektur wäre bei den typischen Anwendungen des Isochron-Modus, d.h. Realzeit-Übertragungen z.B. von Antriebsdaten, wenig sinnvoll.

2.2.1.2 Device Requests

Die gesamte Kommunikation zwischen Master- und Slave-Node wird mit Hilfe von *Device Requests* abgewickelt. Die hier verfolgte Strategie lehnt sich sehr stark an das Konzept vom USB an. In Ergänzung zu den *Standard Device Requests* können weitere geräteabhängige Requests definiert sein. Hierbei unterscheidet man einerseits profilspezifische Requests und zum anderen herstellerspezifische Requests.

In der folgenden Tabelle sind die *Standard Device Requests* definiert, die jedes Gerät unterstützen muss.

Request	**wichtige Parameter**	**Bemerkung**
GET_STATUS	Endpunkt	Abfrage des Zustands eines Endpunkts
CLEAR_FEATURE	Feature, Endpunkt	Deaktiviert ein Feature

SET_FEATURE	Feature, Endpunkt	Aktiviert ein Feature
SET_ADDRESS	Geräteadresse	Zuweisen der Geräte-Adresse
GET_DESCRIPTOR	Deskriptortyp	Ermitteln eines Deskriptors
SET_DESCRIPTOR	Deskriptortyp, Deskriptor	Updaten oder Hinzufügen eines Deskriptors
GET_CONFIGURATION	-	Ermitteln der Konfiguration
SET_CONFIGURATION	Konfiguration	Setzen der Konfiguration
GET_INTERFACE	Interface	Ermitteln der alternativen Einstellungen des momentanen Interface
SET_INTERFACE	Einstellungen, Interface	Setzen der alternativen Einstellungen des momentanen Interface

2.2.1.3 Deskriptoren

Deskriptoren werden von den Geräten benutzt, um ihre Attribute dem Master mitzuteilen. Es sind verschiedene Standard Deskriptortypen definiert.

Deskriptor	**Bemerkung**
DEVICE	Enthält allgemeine Informationen über das PAPAS- Gerät
CONFIGURATION	Enthält Informationen über eine spezielle Geräte-Konfiguration. Jedes Gerät hat eine oder mehr Konfigurationen. Jede Konfiguration hat ein oder mehr Interfaces. Jedes Interface verfügt über einen oder mehr Endpunkte.
STRING	Optional
INTERFACE	Beschreibt ein bestimmtes Interface.
ENDPUNKT	Enthält Informationen über einen Endpunkt. Für den Default-Endpunkt existiert kein Deskriptor.

Auch hier gilt: In Ergänzung zu den Standard Deskriptoren können weitere geräteprofilabhängige Deskriptoren definiert sein.

2.2.1.4 Geräteklassen - Geräteprofile

Alle Geräte müssen die Standard Requests und Standard Deskriptoren bereitstellen. Es ist möglich, zusätzliche gerätespezifische Requests und Deskriptoren zu definieren. Um eine Geräteklasse zu definieren, muss das Erscheinungsbild und das Verhalten dieser Geräteklasse beschrieben werden. Es sind zusätzlich in der Klassendefinition die von dem Gerät benutzten Interfaces und Endpunkte zu beschreiben.
Für die in PAPAS zum Einsatz kommenden, unterlagerten Bussysteme sind verschiedene Geräteprofile(Geräteklassen) definiert. Geräteprofile haben den Vorteil gemeinsamer Merkmale, die es erlauben, generische Treiber(Client Software) für die verschiedenen Geräteklassen zu erstellen.

2.2.2 PAPAS – Kommunikationsmodell

In Abb. 2-3 ist die Aufgabenverteilung innerhalb des PAPAS Kommunikationsmodells dargestellt. Es werden folgende Begriffe und Schnittstellen eingeführt, um eine klare Trennung unterschiedlicher Softwaremodule zu ermöglichen:

- *Endpunkt:* Als Endpunkt wird das „Ende" eines Datenflusses zwischen Master- Node und einem Gerät bezeichnet.
- Jedes logische Gerät besteht aus einer Anzahl von unabhängigen Endpunkten.
- *Interface:* Ein Interface besteht aus einer Anzahl (Gruppierung) von Endpunkten

- *Konfiguration*: Jedes Gerät hat eine oder mehr Konfigurationen. Jede Konfiguration hat ein oder mehr Interfaces. Jedes Interface verfügt über einen oder mehr Endpunkte.
- *Pipe:* Bei PAPAS erfolgt die Kommunikation zwischen Master - und Slave – Nodes (Geräten) über sogenannte Pipes. Eine Pipe ist eine abstrakte Punkt- zu - Punkt-Verbindung - letztlich einfach eine Art Ressourcen-Reservierung. Solange eine Anwendung ein Gerät benutzt, wird die Pipe aufrecht erhalten. Existierende Pipes werden grundsätzlich nicht beendet - auch nicht, wenn die gesamte verfügbare Bandbreite vergeben ist, und neue Geräte an den Bus angeschlossen werden. Es werden zwei Arten von Pipes unterschieden. Während Message Pipes eine definierte PAPAS Datenstruktur enthalten und somit überprüfbar sind, wird der Inhalt von Stream Pipes keiner Prüfung unterzogen - dies ist Aufgabe der Client- bzw. Anwendersoftware.
- *Transaktion:* Eine Transaktion ist ein auf dem Bussystem stattfindender, paketorientierter Datenaustausch.
- *Transfer:* Ein Transfer kann aus einer oder mehreren Bustransaktionen bestehen.
- *IRP* (I/O-Request Packet): Ein IRP kann aus einem oder mehreren "Transfers" (vgl. PAPAS -Protokoll) bestehen. Sowohl die Client- als auch die „Bussoftware" kommunizieren über diese Schnittstelle mit der PAPAS - Systemsoftware.

Grundsätzlich besteht ein PAPAS - System aus einem Master- und einem oder mehreren Slave – Nodes (Geräten).

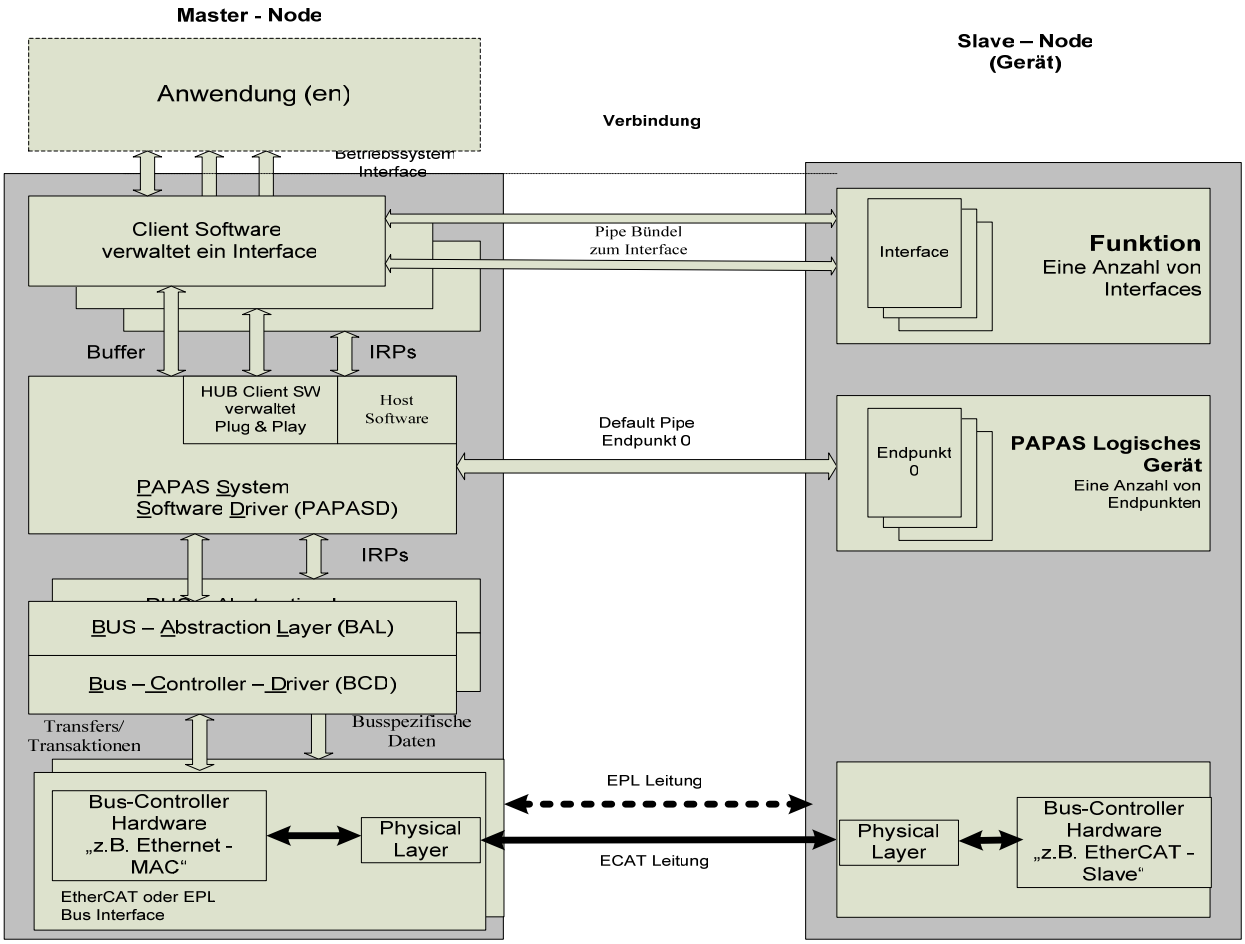


Abb. 2-3 PAPAS - Kommunikationsmodell

2.2.2.1 Master Node: Hardware und Software

Der Master Node besteht im Wesentlichen aus den vier Komponenten.

- Bus Interface Hardware
- Bus System Software
- PAPAS System Software
- Client Software

Bus Interface Hardware

Ein ähnliches Interface findet sich auch im Slave Node wieder. Als busunabhängiges Kommunikationssystem unterstützt PAPAS deterministische und „Plug-And-Play"-taugliche Bustechnologien. Mindestanforderung ist jedoch ein zyklischer (isochroner) und azyklischer Datenkanal, mit dessen Hilfe Produktivdaten und Parameterdaten ausgetauscht werden können. Grundsätzlich hat das Bus Interface die Aufgabe die Daten - entsprechend den physikalischen Anforderungen - in Bitströme auf der Datenleitung umzuwandeln. Die Bus Interface Hardware besteht aus den Funktionseinheiten MAC (Medium Access Controller) und dem PHY (Physical Layer Transceiver).

PAPAS System Software

Beim PAPAS-System-Driver (PAPASD) handelt es sich um die Kernsoftware, die vom Betriebssystem als PAPAS-Unterstützung zur Verfügung gestellt wird. Das PAPAS-System wird in mehrere Softwareschichten unterteilt:

- PAPAS Driver (PAPASD)
- HUB Client Software („Plug-And-Play")
- Bus Abstraction Layer (BAL)
- Bus Controller Driver (BCD)
- Host Software (optional)

Der PAPASD stellt dem Client das PAPASDI (I wie Interface) zur Verfügung. Der PAPASD bietet Mechanismen zur Datenübertragung in Form von I/O-Request Packets (IRPs). Weiterhin ermöglicht der PAPASD dem Client die Parametrierung des funktionell zugehörigen Gerätes. Der PAPASD verwaltet dazu die Default-Pipe (Endpunkt 0). IRPs werden vom PAPASD an den Bus Abstraction Layer Driver (BAL) weitergereicht. Diese Schicht bearbeitet die von oben kommende Information und wandelt sie in für den Bus Controller Driver (BCD) verständliche Form um. Die Aufgabe des BCDs besteht seinerseits darin, den Kommunikationsverlauf auf dem entsprechenden Bussystem zu steuern.

Der Hub Client ist ein wichtiger Bestandteil der PAPAS System Software. Er spielt die zentrale Rolle beim automatischen Einbinden und Entfernen der Geräte. Dieser Teil der Software initiiert alle Konfigurationsaktivitäten, die zur funktionsgerechten Integration eines Gerätes notwendig sind. Der Hub Client benutzt dazu das gleiche Interface wie die Client Software. In einigen Betriebssystemen existiert zusätzliche System Software zum Laden und Konfigurieren des Gerätetreibers (Host-Software). In solchen Fällen sollte dieses Interface -anstelle des direkten Zugriffs auf das PAPASDI- benutzt werden.

Client

Die Client-Schicht umfasst sämtliche Softwareteile, die für die Interaktion mit dem zugehörigen Gerät verantwortlich sind. Die Client-Schicht ist nicht in der Lage, direkt auf die Hardware eines Gerätes zuzugreifen. Die Client-Software ist der eigentliche Geräte-Treiber, der vom Geräte-Hersteller mitgeliefert wird. Eine Ausnahme hiervon bilden die Klassentreiber, die die in dem jeweiligen Geräteprofil definierten notwendigen Eigenschaften unterstützen.

Insgesamt umfassen die Schichten die folgenden Funktionen:

- Erkennen, dass ein Gerät hinzugekommen ist bzw. entfernt wurde („Plug-And-Play" Unterstützung);

- Durchführen der Standard-Kontroll-Aktivitäten (Endpunkt 0);
- Den Datenaustausch zwischen Master und Geräten durchführen;
- Erstellen von Statistiken über Zustand und Aktivitäten;
- Überwachung der Kommunikationsabläufe;

2.2.2.2 Slave Node: Hardware und Software

In diesem Absatz werden die wesentlichen Bestandteile beschrieben, die ein „Plug-And-Play"- fähiges Gerät unterstützen muss, um als Slave-Node arbeiten zu können. Slave–Nodes sind in erster Linie Geräte wie Antriebe, Sensoren oder I/O-Einheiten. Auch ein Slave Node wird in mehreren Schichten dargestellt (siehe Abb. 2-3).

Die unterste Schicht - das Bus Interface - sendet und empfängt Datenpakete. Die mittlere Schicht ist für das Routing zuständig. Das heißt die empfangenen Daten werden den verschiedenen Endpunkten zugeordnet. Die oberste Schicht repräsentiert die gerätespezifische Funktionalität des Slave Nodes.

Bevor ein Slave(Gerät) benutzt werden kann, ist eine Konfiguration nötig, die vom Master durchgeführt wird. Er bezieht dazu aus dem Gerät Informationen(Deskriptoren), um Eigenschaften und Fähigkeiten des Gerätes zu ermitteln.

Während der Konfiguration durchläuft der Slave Node eine Reihe von Zuständen. Unter Zuhilfenahme der masterseitigen Bus Abtraction Layer (BAL) Software ist es möglich, die vom real eingesetzten Kommunikationssystem angebotenen „Zustände" in PAPAS -konforme, zu transformieren.

Die nachfolgende Tabelle erläutert diese Zusammenhänge am Beispiel von EtherCAT.

Zustand - PAPAS	Zustand - EtherCAT	Bemerkung
Attached	Init	Gerät mit Bussystem verbunden und eingeschaltet
Address	PreOperational	Adressvergabe und Parametrierung für den Bus
Configured	SaveOperational	Funktions-Konfiguration
Operational	Operational	Gerät betriebsbereit für Anwendung

Die Gerätezustände sind auch von besonderer Bedeutung für die unterschiedlichen „Plug-And-Play" Arten und können sowohl vom Hub-Client als auch vom zugeordneten Slave-Client beeinflusst werden.

2.2.2.3 PAPAS- Hubs

Die PAPAS-Bus-Topologie entspricht einem schichtförmigen Sternsystem(vgl. Baumstruktur). An der Spitze der Hierarchie befindet sich der so genannte "Master Hub". Im Zentrum jeder Schicht darunter ist ein weiterer Hub angeordnet, an welchen die Geräte und eventuell einer oder mehrere Hubs angeschlossen sind, die alle zusammen die nächste Schicht bilden. Dies kann sich für beliebig viele Ebenen wiederholen.

Je nach Bussystem sind Hubs physikalisch vorhandene - (z. B. Ethernet PowerLink), oder virtuelle Komponenten (z. B. EtherCat). Sie erfüllen im Rahmen des „Plug-And-Play" Szenarios wichtige Aufgaben:

- Erkennen, dass Geräte hinzugekommen bzw. entfernt wurden;
- Erkennung von Busfehlern und entsprechende Reorganisation;

Physikalisch betrachtet ist ein Hub eine Art „Verteiler" mit mehreren Anschlüssen (Ports). Die Ports eines Hubs können sich unter anderem in folgenden Zuständen befinden:

- Disabled: Der Port leitet in keiner Richtung irgendwelche Signale weiter. Ein Port wird dann disabled, wenn das Gerät entfernt worden ist, wenn der Master den Hub dazu aufgefordert hat oder falls eine Fehlersituation an diesem Port registriert worden ist (Resetzustand).
- Attached: Es werden keine Daten in Geräte-Richtung weitergereicht. Ein Port wechselt in diesen Zustand nach Aufforderung durch den Master oder wenn ein Gerät mit ihm verbunden wird.
- Enabled: In diesem Zustand ist der Port bereit, Daten weiter zu leiten. Ein Port betritt diesen Zustand nach erfolgreicher Initialisierung.

2.2.2.4 Software-Mechanismus

Auf der Seite des Master-Nodes ist die Software in Schichten aufgebaut: Auf oberster Ebene befindet sich die Client Driver Software. Die Zugriffe auf die einzelnen Slave-Nodes werden durch den PAPASD (PAPAS System Driver) zusammengefasst und an die unterlagerte Schicht weitervermittelt. Für den Datenaustausch zwischen Client und Funktion ist im PAPAS- System ein Pufferspeicher-Management enthalten.

Auf unterster Ebene arbeitet der BCD (Bus Controller Driver), der den Zugriff auf die Hardware realisiert. Ein vorgeschalteter BAL (Bus Abtraction Layer) abstrahiert die unterschiedlichen BCDs (z.B. EtherCat Master Treiber), sodass eine einheitliche Schnittstelle zum PAPASD entsteht. Da ein System mehrere Bus-Controller haben kann, sind hier mehrere Treiber möglich.

Der PAPASD arbeitet als Multiplexer zwischen den einzelnen Client- Drivern und den Bus Abtraction Layern. Er regelt die Zuteilung der Bandbreite.
Der Client- Driver sieht nur sein Gerät, für das er zuständig ist. Dabei kann es vorkommen, dass das Gerät zwar angesprochen wird, erhöhte Bandbreiten- Anforderungen aber nicht bedient werden können. Dies gilt nur für azyklische Vorgänge –für isochrone Transfers wäre es eine fatale Verletzung deterministischer Datenverbindungen.

2.2.2.5 Konfiguration

Einer der großen Vorteile des PAPAS- Systems ist es, Geräte während des Betriebs hinzufügen und entfernen zu können. Wenn ein Gerät angeschlossen wird, weist der PAPASD diesem Gerät eine freie Geräte- ID zu.

Nachdem das Gerät erkannt ist, sind drei Konfigurations-Schritte für jedes Gerät notwendig:

1. Geräte- Konfiguration:
   Einstellen aller Bus-Parameter für das Gerät. Bereitstellen von Ressourcen, die für das Gerät sichtbar sind.

2. PAPAS- Konfiguration:
   Einstellen von Parametern, die für das Gerät nicht sichtbar sind. Diese Informationen betreffen die Pipe und beschreiben, wie die Pipe durch den Client benutzt werden kann. Festgelegt wird dabei, wie viele Daten maximal je IRP übertragen werden können und das entsprechende Service Intervall. Das Service Intervall beschreibt, in welchen zeitlichen Abständen Daten über eine isochrone Pipe übertragen werden. Es beschreibt das Polling-Intervall für Interrupt- Pipes. Mit diesem Schritt ist die Konfiguration bis zur Schicht PAPAS- System abgeschlossen, d.h. bustechnische Belange sind erfüllt.

3. Funktions- Konfiguration:
   Nun, da die Pipe benutzt werden kann, können gerätespezifische Einstellungen vorgenommen werden. Diese Konfigurationen betreffen nur noch die Client- und Funktions- Schicht.

Wie nicht anders zu erwarten war, bietet auch die PAPAS-Spezifikation keine konkrete Festlegung der Schnittstellen zum „Anwenderprogramm", dem PAPASDI. Hier wird auf das jeweilige Betriebssystem verwiesen. Einen Sinn macht das deshalb, da die Nutzung von Peri-

pheriegeräten nicht von deren Hardware-Interface abhängig sein soll. Für eine Antriebssteuerung sollte es egal sein, ob der Antrieb z.B. am Parallel-Port oder an einem Bussystem angeschlossen ist.

2.2.2.6 Ressource Management

Wenn eine Pipe zu einem Endpunkt erzeugt wird, ist es die Aufgabe des PAPAS- Systems festzustellen, ob diese Pipe unterstützt wird. Dabei wird u. a. überprüft, ob die geforderte Bandbreite vom Endpunkt bereitgestellt werden kann. Konkret wird diese Aufgabe vom Bus Controller Driver (BCD) übernommen, da er während der Konfiguration die entsprechenden Busteilnehmerbelange erfassen kann. Bei der Zuteilung von Bandbreiten ist es wichtig, anzumerken, dass auch sehr "anspruchsvolle" isochron arbeitende Geräte (Antriebe) grundsätzlich nicht die volle Bandbreite belegen dürfen. Um zu verhindern, dass der Bus von solchen Geräten blockiert wird, reserviert der BCD in jedem Fall ca. 10% der freien Bandbreite für asynchrone Geräte und Protokoll-Aufgaben.

2.2.2.7 Datentransfer

Ein Interface(Gerätefunktion) ist ein Bündel von Pipes, die mit einem bestimmten PAPAS-Gerät verbunden sind. Ein solches Interface wird von genau einem Client verwaltet. Aus Sicht des Client stellen sich die Daten als serieller Datenstrom dar. Um Daten zu senden, setzt der Client ein IRP an den PAPASD ab. In Abhängigkeit von der Richtung der Übertragung wird dabei ein leerer oder ein gefüllter Puffer übergeben. Ist der Request abgeschlossen, kehrt der IRP mit einem Statuswert an den Client zurück. Ein IRP besteht aus einem oder mehreren Bus-Transaktionen.

2.2.2.8 Bus Abstraction Layer

Der BAL ist die Abstraktion der Bus Controller Hard- und Software. Er stellt ein Software-Interface zu Verfügung, welches durch die PAPAS- Spezifikation festgelegt ist. Die Aufgabe des BALs besteht nun seinerseits darin, den Kommunikationsverlauf des unterlagerten Bussystems zu steuern. Hierzu werden die vom PAPASD ankommenden Transfers in busspezifische Transaktion umgewandelt, in eine prioritätsbehaftete Liste eingetragen und dementsprechend verarbeitet. Eine adäquate Diagnose und Fehlerbehandlung des verwalteten Busses ist ebenfalls implementiert.

2.2.2.9 Bus Controller Driver

Der Bus Controller Driver (BCD) ist die Schnittstelle zur ausgewählten Bus Controller Hardware. Er ist die unterste Ebene im PAPAS Stack. Dieses Interface ist durch die jeweilige Busspezifikation festgelegt.

Grundsätzlich ist es mit PAPAS gelungen die Anwendungsentwicklung erheblich zu vereinfachen und zu beschleunigen. Dies konnte einerseits durch die busunabhängige Implementierung und andererseits durch die „Plug-And-Play" Eigenschaften des Systems erreicht werden. Traditionell liefern Hersteller gerätespezifische, hardwareabhängige Koppelkarten mit betriebssystemabhängiger Treibersoftware für ihre Geräte. Es ist dann die Aufgabe des Anwenders die Peripherie mit großem Aufwand in seine Applikation zu integrieren.
Feldbussysteme mit genormten Hard- und Softwareschnittstellen verringern zwar den Aufwand beim Gerätewechsel, solange das Bussystem beibehalten wird.

PAPAS geht noch einen Schritt weiter und beseitigt durch die Einführung einer Busabstraktionsschicht die Abhängigkeit vom unterlagerten Bussystem. Dem Anwender ist es dadurch möglich, ohne jeglichen Zusatzaufwand, Geräte mit gleicher Funktion aber unterschiedlichen Busschnittstellen in einer Applikation zu verwenden.

„Plug-And-Play" ist allerdings das bedeutende Merkmal von PAPAS. Peripherie, die einem PAPAS- Geräteprofil entspricht und eine entsprechende Client Software bereitstellt, kann nach erfolgter, automatischer Konfiguration unmittelbar in einer Applikation eingesetzt werden.

2.3 Bewertung von Bussystemen

Von Anfang an hatten die Projektpartner nicht die Absicht einen neuen, leistungsfähigeren Feldbus zu entwickeln. Ziel war es vielmehr, bereits vorhandene, hoch leistungsfähige Bussysteme auf deren Eignung für eine „Plug-And-Play" Kommunikation zu untersuchen und dann eine prototypenhafte Implementierung durchzuführen.

2.3.1 Anforderungsanalyse

Das in PAPAS zu realisierende Kommunikationssystem muss viele Anforderungen aus den unterschiedlichsten Anwendungsbereichen erfüllen. Auf der Protokollebene müssen z.B. Dienste aus dem Bereich der Feldbusse, aber auch administrative bzw. wartungstechnische Dienste realisiert werden.

Das PAPAS-Protokoll muss eine Synchronisation zwischen der zyklisch arbeitenden Steuerung und den ebenfalls zyklisch arbeitenden Antrieben, Sensoren und mechatronischen Systemen bieten. Von einem Interpolator werden in der Steuerung zyklisch – in gleich bleibenden, kurzen Zeitabständen – Sollwerte für jeden Antrieb errechnet. Jeder Antrieb folgt den zyklisch vom Interpolator gelieferten Sollwerten mittels einer eigenen Regelung. Auf diese Weise erfolgt sowohl die präzise Steuerung von Einzelantrieben als auch die exakte Interpolation mit beliebig vielen Antrieben. Die Zeitpunkte, zu denen die Istwerte erfasst werden, und die Zeitpunkte, zu denen die Sollwerte im Antrieb wirksam werden, sind für die präzise Koordination der Antriebe ebenso bedeutend wie die Genauigkeit der interpolierten Sollwerte und die Messgenauigkeit. Eine Maßpräzision von 1 Mikrometer entspricht bei einer Geschwindigkeit von 1 m/sec. z.B. einer Zeitpräzision von 1 Mikrosekunde. Die Anpassung der Antriebe, Sensoren und sonstigen Aktuatoren an unterschiedliche Anwendungen und Steuerungen erfolgt typischerweise durch Parametrierung. Neben den im Betrieb auszutauschenden zyklischen Echtzeitdaten ist für den Austausch von Parametern und Diagnoseinformationen ein azyklisches aber sicheres Protokoll ausreichend. Mit Hilfe von Anwendungsszenarien wurden gemeinsam mit allen Projektpartnern unter Einbeziehung der Erkenntnisse aus bereits existierenden Industrie- und Anwenderarbeitskreisen die Anforderungen an das zu entwickelnde Kommunikationssystem spezifiziert.

2.3.2 Bewertungskategorien

Zur Auswahl eines Kommunikationssystems wurden die Bewertungskategorien Technik Echtzeit, Technik allgemein, Offenheit/Standards, Kommerziell/ Verfügbarkeit und Werkzeuge für Engineering/Diagnose aufgestellt. Diese Hauptbewertungskategorien zur Auswahl des Kommunikationssystems wurden in verschiedene Teilaspekte untergliedert. Die Bewertung und Gewichtung der einzelnen Kategorien und Teilaspekte für die Auswahl wurden gemeinschaftlich vorgenommen. Die Bewertung in den Kategorien wurde ergänzt um eine Bewertung von Anwendungsfällen (Use Cases) aus dem Bereich der Robotik, anhand derer die Eignung der Kommunikationssysteme konkreter gefasst wurde. Im Folgenden sollen die einzelnen Bewertungskategorien kurz erläutert werden.

Technik Echtzeit

Die Echtzeitfähigkeit eines Kommunikationssystems ist die wesentliche Grundlage für dessen Eignung in einem Automatisierungsumfeld. Die Echtzeitfähigkeit wird durch Kriterien wie Zykluszeit, Anzahl der Teilnehmer, Integration von E/A oder Kameras, Synchronisierung, Querverkehr, Datendurchsatz usw. bestimmt.

Technik allgemein

Die übrigen technischen Aspekte eines Kommunikationssystems wie Einfachheit der Verdrahtung, mögliche Leitungslänge, Elektromagnetische Verträglichkeit, „Plug-And-Play" Eigenschaften (Adressierung, Gerätewechsel), Diagnosefähigkeiten usw. stellen ebenfalls wichtige Auswahlkriterien dar.

Offenheit / Standards

Die Offenheit und Standardisierung sowie die Verfügbarkeit dieser Standards sind wichtig für den Einsatz eines Kommunikationssystems im Rahmen von PAPAS. Darüber hinaus sollte

das System international verbreitet sein oder die Verbreitung absehbar sein. Ebenso sollte das System für verschiedene Komponenten wie E/As, Antriebe, Steuerungen usw. standardisiert sein.

Kommerziell / Verfügbarkeit

In dieser Kategorie werden die Verfügbarkeit der erforderlichen Komponenten sowohl heute als auch zukünftig und die kommerziellen Aspekte betrachtet. Dies beinhaltet die Anschaltungen für Geräte und Steuerungen sowie erforderliche Infrastrukturkomponenten.

Werkzeuge für Engineering / Diagnose

Ohne geeignete Werkzeuge für Engineering und Diagnose ist der Einsatz eines Kommunikationssystems nicht möglich. Berücksichtigt wird dabei die Geräte- und die Netzwerksicht.

2.3.3 Use Cases

Neben den objektivierbaren technischen bzw. ökonomischen Kriterien, wurden auch Anforderungen formuliert, die aus den bisherigen bzw. geplanten Anwendungen der Partner abgeleitet sind. Allen *Use Cases* ist gemeinsam, dass sie aus Themenfeld Robotik stammen.

Im ***Use Case 1*** wird ein Szenario zugrunde gelegt, das höchste Anforderungen an die Geschwindigkeit der Übertragung bzw. den Determinismus eines Bussystems stellt. Es handelt sich dabei um einen Roboter, in dem alle Komponenten an einem einzigen Bussystem angeschlossen sind. Kennzeichnend für die benötigte Kommunikation ist, dass die einzelnen Komponenten in ganz unterschiedlichen Takten arbeiten.

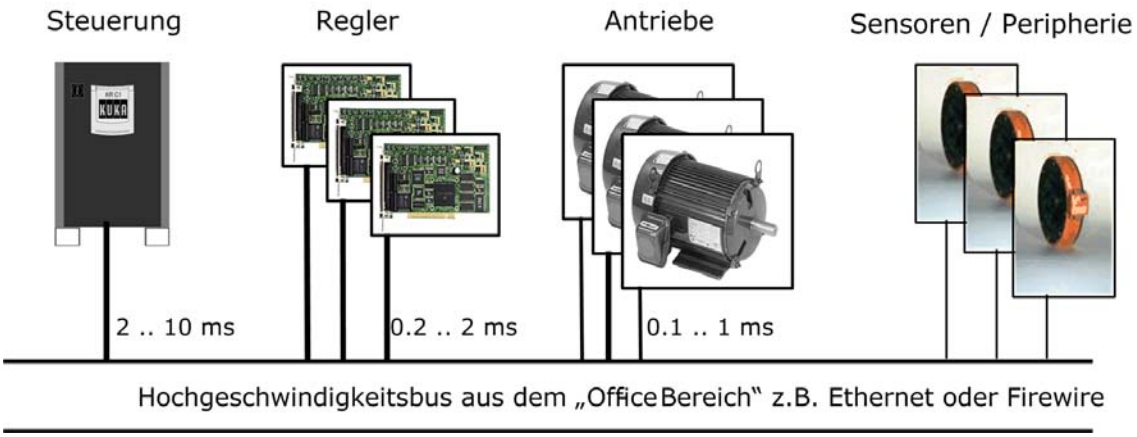


Abb. 2-4 Use Case 1

Es wird davon ausgegangen, dass es sich um ein Single-Master System handelt, das in einem minimalen Zyklus von 125µs arbeitet. An den Bus sind u.a. folgende Teilnehmer angeschlossen:

- PC
- Regelungsprozessor
- 6-12 „dumme Umrichter" (ohne Regler)
- 1 -3 Positionserfassungssysteme
- Powermodul (zeitunkritisch)
- Zusatzkomponenten
    - 1-3 Sensoren
    - 1-3 Aktoren
    - Kamera (1 MByte)

Dieses Szenario deckt die Vision von einem einzigen Kommunikationssystem in einem Roboter ab.

Beim ***Use Case 2*** handelt es sich ebenfalls um ein Industrieroboter-Szenario. Im Unterschied zu dem Vorigen wird hier jedoch davon ausgegangen, dass sich in der Kommunikationskette auch intelligente Technologiemoduln wie z.B. die programmierbare Fokussieroptik von Trumpf-Laser, bzw. die intelligente Umrichter (mit Regler), befinden, die eine eigene

interne Kommunikation besitzen. Die Anforderungen an die Kommunikationsgeschwindigkeit sind hier geringer. Zusätzlich zu den im *Use Case 1* aufgeführten Komponenten kommen noch folgende hinzu:

- intelligente Steuerungssysteme wie z.B. PC, SPS, Controller, DriveSPS, …
- bis zu 10 Zusatzteilnehmer in Form von Technologiemoduln

Im ***Use Case 3*** werden die Anforderungen an die Kommunikation betrachtet, die dadurch entstehen, wenn rein intelligente Technologiemoduln, oder ganze voneinander unabhängige Systeme intelligenter Technologiemoduln miteinander kommunizieren müssen. Ein modular aufgebauter Roboter kann man sich z.B. als eine Kombination von Technologiemoduln vorstellen. Genauso kann man einen vollständigen Roboter als **ein** Technologiemodul betrachten. Das hier ins Auge gefasste Szenario ist z.B. eine Roboterarbeitszelle mit mehreren Robotern, mehreren Zuführungseinrichtungen, mehreren PFO's, …. Die Kommunikationsteilnehmer sind:

- intelligente Steuerungssysteme (3 – 20)
- 50-100 Teilnehmer in Form von Technologiemoduln

Sie können dabei räumlich sehr weit auseinander liegen (100 – 200m).

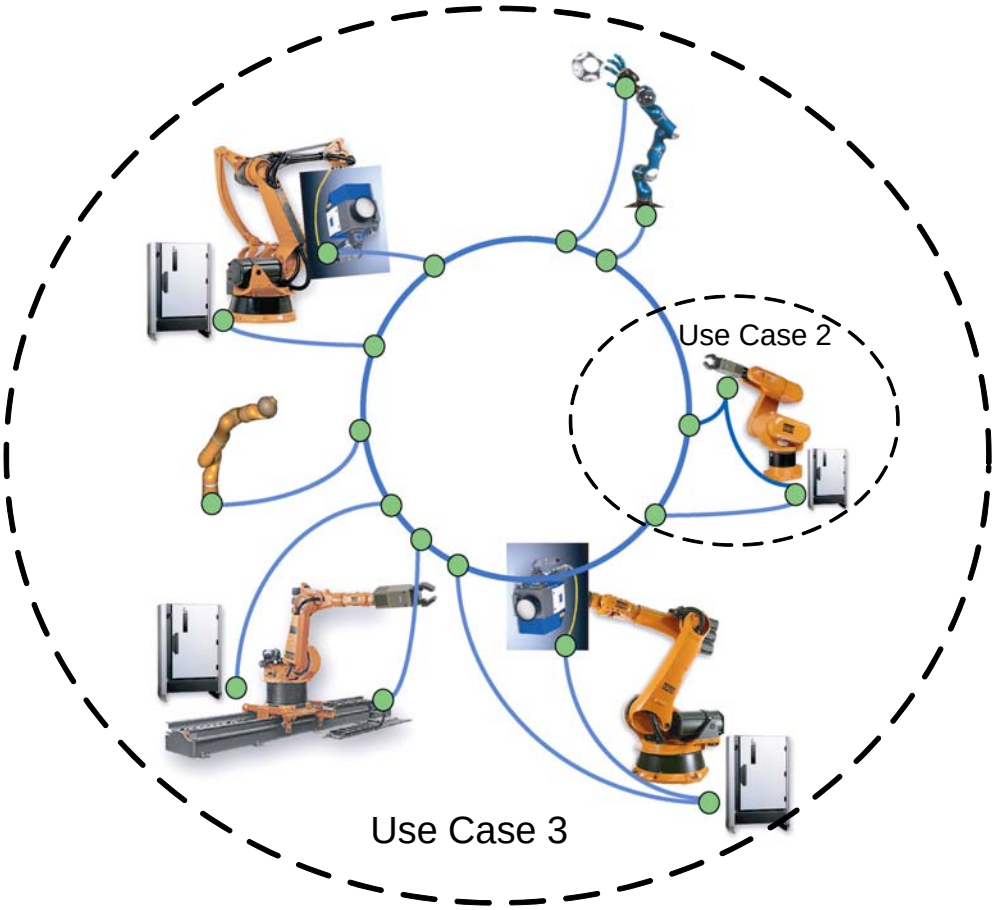


Abb. 2-5 Use Cases 2 & 3

Der ***Use Case 4*** unterscheidet sich von den vorangegangenen im Wesentlichen dadurch, dass kein klassisches Industrieroboterszenario adressiert wird. Es handelt sich um einen mobilen Serviceroboter, wie er in der Zukunft in unterschiedlichen Ausprägungen in Industriehallen, aber auch in Büros, Krankenhäusern und Haushalten zu finden sein wird.

Abb. 2-6 Service Roboter

In dem Single-Master System finden sich folgende Teilnehmer:

- 1-3 PC's
- Regelungsprozessor
- 7-20 Servoachsen, z.B. ein 7-achsiger Roboterarm und eine künstliche Hand mit 13 weiteren Achsen
- ein zeitunkritisches Stromversorgungsmodul
- Zusatzkomponenten
    - Kraft-/Momentsensoren in jeder Achse
    - Kameras
    - Optische Abstandsensoren

2.3.4 Ausgewählte Kommunikationssysteme

Gemäß den Bewertungen in der Entscheidungsmatrix fällt die Entscheidung gleichermaßen zugunsten Powerlink und EtherCat aus.

Im Rahmen des Projektes ist es jedoch nicht sinnvoll das PAPAS Protokoll auf zwei Kommunikationssysteme aufzubauen.

Sowohl Powerlink als auch EtherCat gestalten nach Aussage der jeweiligen Nutzergruppe ihre Kommunikationsprofile gemäß CANopen. Das CANopen Kommunikationsprofil wurde von der Nutzergruppe CAN in Automation (CiA) erarbeitet.
Aufgrund dieser Vereinbarung wird das PAPAS Protokoll zum einen als Erweiterung des CANopen Kommunikationsprofils und zum anderen in Form von Geräteprofilen auf Basis des Kommunikationsprofils gestaltet.

Die Erweiterungen und Geräteprofile beinhalten die folgenden Teile:

- Plug and Play Erweiterungen
- Nutzbarkeit der CANopen Geräte- und Anwendungsprofile DS4xx
- Application Identification & Configuration
- Eindeutige Zuordnung der Geräte zu Anwendungen und Beschreibungsdaten
- ADR Automatic Device Replacement
- Backup- und Restoremechanismen
- Roboter und Peripherie Definition von Gerätefunktionen und deren Parametern
- Elektronische Beschreibung für weitere Geräte- und Systemeigenschaften
- Mechanik

- Simulationsmodelle
- Konfigurationsmodelle
- Functional mapping
- Verbinden von funktionalen Einheiten

2.4 „Plug-And-Play" Implementierung von PAPAS

Leitgedanke bei der Implementierung eines „Plug-And-Play"-Konzeptes war die Entwicklung und Bereitstellung einer einheitlichen Schnittstelle zum verwendeten Echtzeitbus zur Verfügung zu stellen, die dem standardisierten I/O Modell von UNIX folgt. Auf der Basis eines geeigneten Gerätetreibers können Anwendungen und Betriebssystem transparent mit der Hardware kommunizieren.

Im Sinne des „Plug-And-Play" Leitgedanken steht eine automatische Erkennung und einfache Konfiguration von Busteilnehmern im Vordergrund, wobei der verwendete Implementierungsansatz mit einem standardisierten Gerätetreibermodell die Abstraktion vom verwendeten Echtzeitbus erlaubt. Erreicht wir dies durch die strikte Trennung von Gerätetreiber, Buskontroller und BAL, dir Vermittlungsebene zwischen dem Gerätetreiber und dem Buskontroller.

Für das „Plug-And-Play" wurden Funktionen zur Busüberwachung und zum dynamischen Laden von Klassentreibern entwickelt. Der Bus wird im Hintergrund beispielsweise zyklisch auf Änderungen überwacht. Neue Teilnehmer werden durch BAL mit einem Gerätetreiber verknüpft, die ins Dateisystem eingehängt sind. Für Geräte die vom Bus entfernt wurden, wird die existierende Verknüpfung zum Gerätetreiber entfernt. Die Nutzung einer Vermittlungsebene (BAL), die für den Benutzer eine transparente Verknüpfung zum Gerät herstellt, bietet dem Benutzer ein vereinheitlichtes, einfaches Interface zu den Busteilnehmern.

Mit Ende der Projektlaufzeit wurde das „Plug-And-Play"-Konzept von PAPAS anhand einer prototypischen Umgebung demonstriert. Die gezeigte Anwendung basiert im Kern auf einen Leichtbauroboter als robotisches Assistenzsystem (vgl. Abb. 2-7). Als prototypische Aufgabe für einen industriellen Fertigungsprozess wurde eine Montageaufgabe ausgewählt, die heutzutage weitestgehend manuell ausgeführt wird.

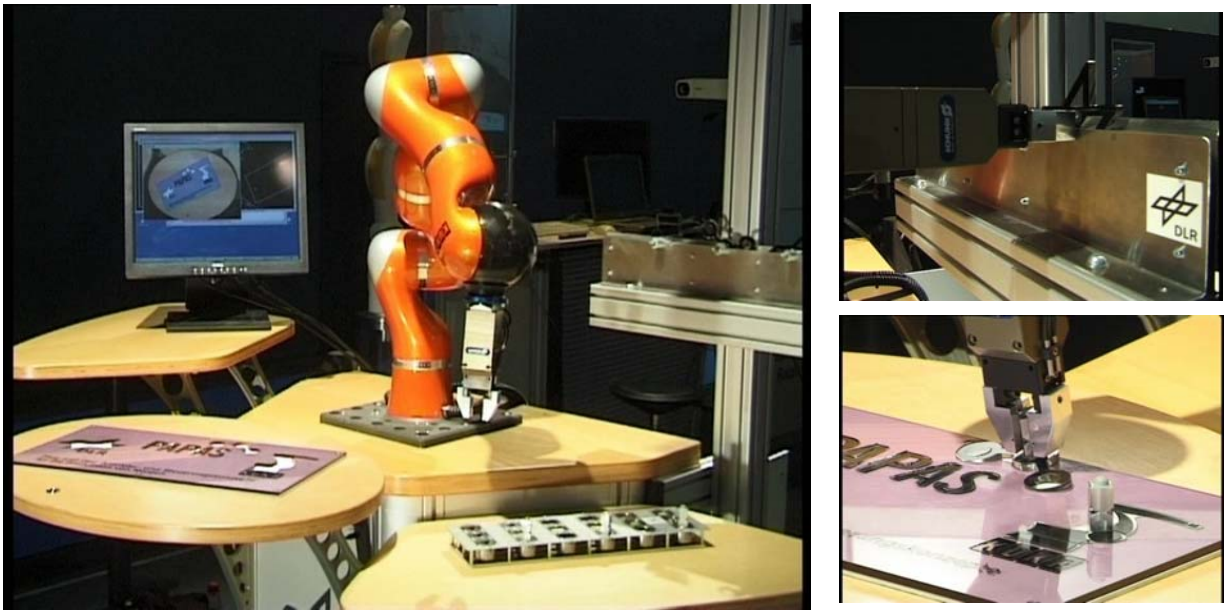


Abb. 2-7 PAPAS – „Plug-And-Play" Demonstrator Szenario

Zur Demonstration und Verifikation der „Plug-And-Play"-Fähigkeiten wurden prototypische Gerätetreiber für eine programmierbare Fokussieroptik (PFO), einen nachgiebigen Kraft-Momenten-Sensor (FTS) und einen Zwei-Backengreifer als typische Vertreter von Industriewerkzeugen entwickelt:

- FTS vom DLR: Das Einsatzfeld eines nachgiebigen Kraft-Momenten-Sensors (FTS), ein Werkzeug zur Messung von Kräften und Momenten in allen sechs Freiheitsgraden, in der Automatisierungstechnik ist vielseitig. Beispielsweise findet der FTS eine breite Anwendung, um Positionsungenauigkeiten des Werkzeugs/-stücks auszugleichen oder an bewegten Werkstücken zu arbeiten. Mit der Berücksichtigt von auftretenden Kräften und Momenten können Werkzeuge, Werkstücke und der Roboter vor Überlastungen geschützt werden.
- PFO der Firma Lenze: In Anwendungen der industriellen Automation kann ein Roboter die programmierbare Fokussieroptik als Werkzeug tragen, um großräumige Bewegungen über ein Werkstück zu ermöglichen, während die Laseroptik für die exakte Positionierung des Laserstrahls zuständig ist. Eine Kombination von PFO und Roboter erlaubt eine freie Bewegung und Orientierung der Laseroptik im dreidimensionalen Arbeitsraum.
- 2-Finger-Parallelgreifer der Firma Schunk

Zwei-Backengreifer

Kraft-Momenten-Sensor

Programmierbare Fokussieroptik

Abb. 2-8 Typische Werkzeuge und Geräteklassen

Für den virtuellen BAL wurden geeignete Gerätetreiber entwickelt, um die genannten prototypischen Industriewerkzeuge über eine einheitliche „Plug-And-Play"-fähige Treiberschnittstelle bereitzustellen. Konzipiert als EtherCAT-Prototypen wurden für diese Werkzeuge prototypische … entwickelt.

Um dem Anwender ein einfaches Mittel zur Evaluierung und Kommandierung von angeschlossenen Geräten anzubieten, wurde im Rahmen von PAPAS ein entsprechendes Werkzeug für eine erste einfache Systemdiagnose entwickelt. Das Werkzeug verwendet die einheitliche Gerätetreiberschnittstelle, über die alle Geräte in das System eingebunden sind, um mit angemeldeten Geräte zu kommunizieren. Die „Plug-And-Play"-Fähigkeit des BAL erlaubt die einfache Erkennung neuer Geräte durch das Diagnosewerkzeug. Jedes neue Gerät wird nach erfolgreicher Systemanmeldung durch das Werkzeug erkannt und als kommandierbares Gerät in einer Geräteliste aufgenommen. Für jedes Gerät muss in der aktuellen Implementierung eine spezifische Eingabemaske hinterlegt und implementiert werden, die mit der Selektion eines Gerätes (z.B. PFO) in der Geräteliste aktiviert wird. Der Anwender kann unter Einsatz dieser Eingabemaske mit dem Gerät kommunizieren. Neben dem Abfragen und Auswerten des Gerätestatus können gerätespezifische Kommandos generiert werden, um das angeschlossene Gerät beispielsweise in Betrieb zu nehmen oder in einen anderen Zustand zu überführen. Eine Busspezifische Diagnose bzw. Kommandierung ist in der aktuellen Version noch nicht realisiert, ist aber in zukünftige Implementierungen vorgesehen.

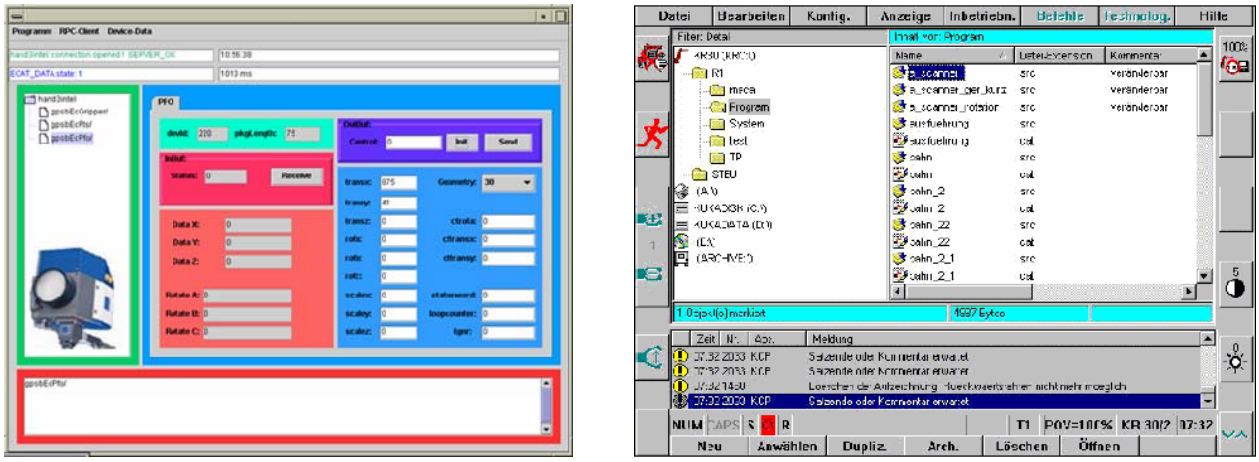


Abb. 2-9 Bedien- und Programmieroberfläche

Die einfache Bedienung und Programmierung eines KUKA Roboters erfolgt über das KUKA-Control-Panel (KCP) mit einer Windows Bedienoberfläche (vgl. **Abb. 2-9 Bedien- und Programmieroberfläche**). Das KCP ist üblicherweise mit dem der Steuereinheit verbunden. Analog zum Diagnosewerkzeug verwendet das KCP die entstandene einheitliche Geräteschnittstelle, um mit angemeldeten Geräten zu kommunizieren. Beispielsweise können Geräteinformationen im Status-Fenster des KCP angezeigt werden. Im Statusfenster wird der Bediener über Änderungen wie die An-/Abmeldung eines Gerätes informiert. Wird beispielsweise ein Greifer neu an den Bus angeschlossen erfolgt eine entsprechende Nachricht im Statusfeld; anschließend kann die Interaktion mit dem Gerät durch die KCP-Steuerung erfolgen.

3 Entwurfstechnologien für „Plug-And-Play“

Die „Plug-And-Play“-Antriebs- und Steuerungstechnik eröffnet für die Konzeption von Handhabungssystemen eine neue Größenordnung an Entwurfsfreiheiten und Variantenreichtum, weil sich neue Systemarchitekturen unter Verwendung vorhandener Antriebs- und Steuerungskomponenten leichter realisieren lassen. Hierzu sind insbesondere modellbasierte Steuerungs- und Regelungsverfahren unabdingbar um z.B. die Maschinendynamik zu verbessern, d.h. schneller, genauer und mit weniger Vibrationen eine Aufgabe durchzuführen.

Die flexible Konfigurierbarkeit erfordert, dass die verwendeten modellbasierten Steuer- und Regelungsverfahren einfach an die jeweilige Situation angepasst werden können. Im PAPAS Projekt werden deswegen Systemmodelle von Maschinen *komponentenorientiert* aufgebaut, da dann im Modell leicht Komponenten wie Getriebe, Motoren, Power-Module ausgetauscht werden können. Diese Modelle werden in Form eines „Modellservers“ zur Verfügung gestellt, der eine leichte Konfigurierbarkeit der für Auslegung, Regelung und Steuerung benötigten Modelle des gewünschten Handhabungssystems erlaubt. Weiterhin werden optimierungsbasierte Entwurfsverfahren zur Verfügung gestellt um das Mehr an Entwurfsfreiheit der „Plug-And-Play“-Technik vollständig ausschöpfen und systemdynamisch beherrschen zu können.

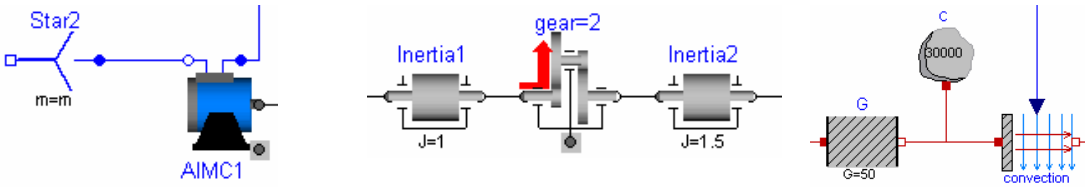
Eine Grobauslegung wird offline mit Optimierungsverfahren basierend auf nichtlinearen Simulationen der vorhandenen Modelle erfolgen. Mit einem Feintuning werden dann Steuerungs- und Reglerparameter online am realen System mit Optimierungsverfahren adaptiert (Hardware-in-the-loop Optimierung). Hierbei werden sowohl die Echtzeitalgorithmen für Steuerung und Regelung, wie auch die einstellbaren Parameter automatisch vom Entwurfsrechner auf die entsprechenden PAPAS Teilnehmer herunter geladen.

3.1.1 Gerätebasierter Modellserver für den Entwurf

Die Funktionalitäten der „Entwurfstechnologien“ basieren sehr stark auf mathematischen Modellen von Komponenten und gesamten Systemen. Die Beschreibung erfolgt „gerätenah“ in komponentenorientierter Form. Systemkomponenten innerhalb des PAPAS-Projektes sind z.B. Antriebe, Sensoren, Tools, Roboterarme, Regler, etc. und sie werden durch ihre physikalischen Parameter beschrieben, wie z.B. Masse, Wirkungsgrad, sowie durch mathematische Gleichungen, die insbesondere ihr Verhalten im Zusammenspiel mit dem Gesamtsystem definieren. Die verwendeten Komponenten, sowie die Maschinen-Systemmodelle (Komponenten plus Verschaltung der Komponenten) werden in einem Modell-Server gespeichert. Dieser besteht aus einer Bibliothek der Modellierungssprache Modelica, in der die Gleichungen aller Komponenten, sowie die Daten-Parameterierung zusammengefasst sind.

3.1.1.1 Objektorientierte Modellierungssprache Modelica

Die frei verfügbare Modellierungssprache Modelica[1] (www.Modelica.org) wird verwendet um komplexe, multi-disziplinäre Systemmodelle mit Differentialgleichungen, algebraischen und diskreten Gleichungen zu beschreiben. Die Anwendersicht sind dabei Objektdiagramme, wie sie beispielhaft im folgenden Bild zu sehen sind:



elektrische Maschinen Antriebsstränge Wärmeleitung

Abb. 3-1 Modelica Objektdiagramme

Freie und kommerzielle Modelica Bibliotheken sind z.B. verfügbar für 1-dim./3-dim. mechanische, elektrische, thermische, hydraulische, pneumatische, thermo-fluid und Regelungs-

[1] Modelica® ist ein eingetragenes Warenzeichen der gemeinnützigen Modelica Associatoin.

technische Komponenten, siehe auch www.Modelica.org/libraries. Modelica Simulationsumgebungen von unterschiedlichen Herstellern sind verfügbar und/oder in Entwicklung. Für eine aktuelle Liste siehe www.Modelica.org/tools. Die zurzeit mächtigste Modelica Simulationsumgebung ist Dymola (www.dynasim.se). Diese wurde im PAPAS Projekt verwendet. Dymola kann Modelica-Modelle nach Simulink[2] in Form von S-Function C-Mex Files exportieren.

Die Auswahl von Modelica als Basis für den Modellserver hat am Ende des PAPAS Projekts eine wichtige Bestätigung gefunden: Im Juni 2006 hat Dassault Systèmes, eines der weltweit führenden Unternehmen bei CAD und PLM (Product Lifecycle Mangement), seine neue Produktlinie CATIA[3] Systems angekündigt, siehe www.3ds.com/cn/news-events/press-room/release/1220/1/. Zentraler Baustein wird hierbei die Verhaltensmodellierung mit Modelica und Dymola sein.

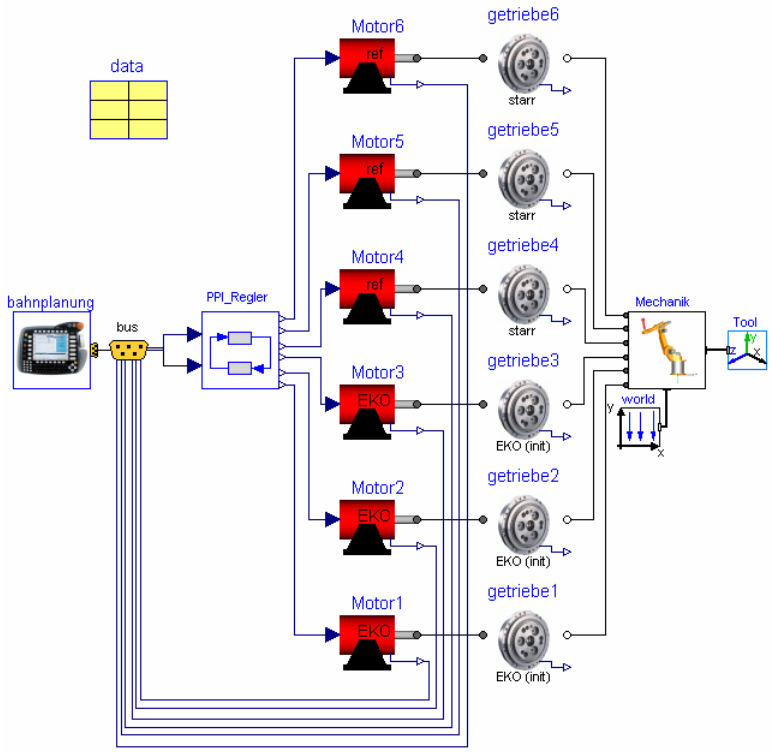


Abb. 3-2 Modelica-Modell eines Roboters im PAPAS Modellserver

3.1.1.2 Modellserver (Modelica Bibliothek)

Modelle von Robotern und anderen Handhabungssystemen werden für unterschiedliche Zwecke eingesetzt, beispielsweise zur Entwicklung von Vorsteuerungsalgorithmen oder zum Entwurf modellbasierten Regler. Aus diesem Grunde ist eine objektorientierte, physikalisch basierte Modellierung notwendig, die mit Modelica einfach implementiert werden kann. Die einzelne Modellkomponenten (siehe Abb. 3-2)– Bahnplanung, Maschinendaten, Regler, Motor, Getriebe und Mechanik – können je nach gewünschtem Detaillierungsgrad mit unterschiedlichen Modellvorstellungen über ein pull-down Menü ausgewählt werden.
Mit einem aus Maschinendaten generierten „data" Block wird der Robotertyp ausgewählt, Die Bahnplanung basiert auf dem C++ Code der Robotersteuerung. Sämtliche Soll- und Ist-Signale werden über das hierarchische Bussystem dem Regler zur Verfügung gestellt. Die mechanische Struktur und die Antriebsstränge bestehen aus Gelenken, Körpern, Motor und Getriebe. Die Subssysteme können sowohl starr als auch elastisch modelliert werden. In Abb. 3-3 ist beispielhaft ein elastisches Mechanikmodell zu sehen. Die Simulationsergebnisse können sowohl in Diagrammen als auch in Form einer automatisch generierten 3-D Animation betrachtet werden, die automatisiert aus den Roboterdaten generiert wird.

[2] Simulink® ist ein eingetragenes Warenzeichen von „The MathWorks".
[3] CATIA® ist ein eingetragenes Warenzeichen von Dassault Systèmes

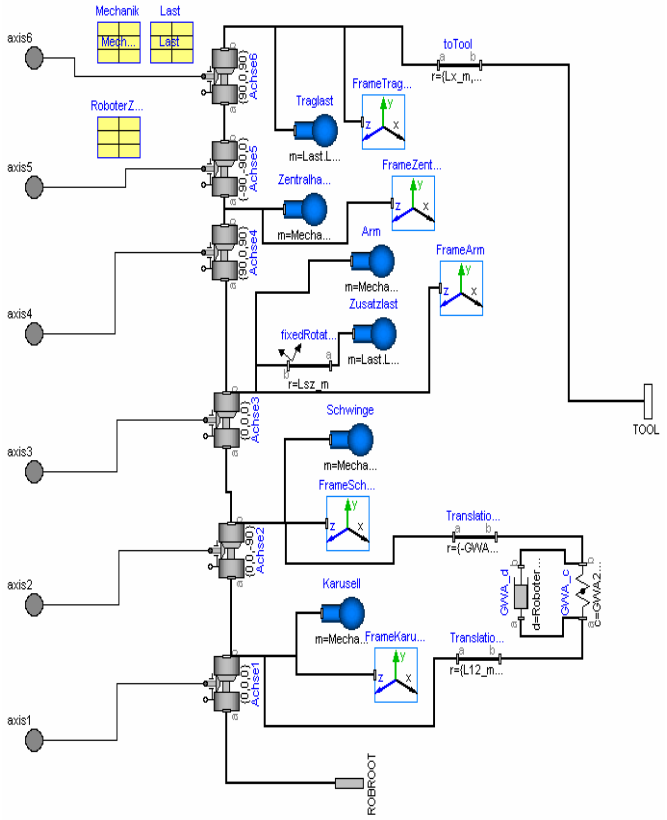
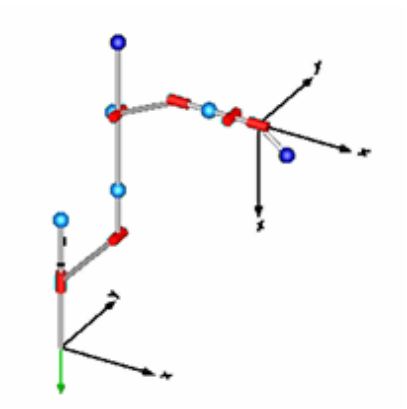


Abb. 3-3 Strukturelastisches Mechanikmodell



Aus Roboterdaten automatisch erzeugte Basisanimation

3.1.1.3 Schnittstelle zu Maschinendaten (KUMAC)

Um das aktuelle Maschinendatenformat von KUKA Robotern im PAPAS Modellserver verwenden zu können, wurde ein flexibler Konverter KUMaC (KUKA Maschine Data Compiler) implementiert, der die Transformation in ein benutzerdefinierbares Format, z.B. Modelica, XML, oder auch C-Code, realisiert. Im Einzelnen wurden folgende Teiloperationen realisiert:

- Erfassung von Datentypen und Variablen aus KUKA-spezifischen Dateien.
- Speicherung der eingelesenen Werte in einem internen XML-Format.
- Beschreibung der Ausgabedatei mittels einer Vorlage.
- Erzeugung einer Zieldatei aus Maschinendaten und Vorlage.

Der Anwender kann entweder über ein graphisches User Interface direkt mit dem Programm interagieren, oder in einem Batch – Modus die gewünschten Operationen auf eine große Anzahl von Dateien anwenden, z.B. um die Maschinendaten der ca. 400 aktuell bei KUKA verwendeten Robotervarianten automatisiert in das Modelica-Format des PAPAS-Modellservers umzuwandeln. Dieser Kompiliervorgang wird dann in einer Batch – Datei beschrieben und automatisch ausgeführt.

Die Anwendungen von KUMAC bestehen aktuell in der

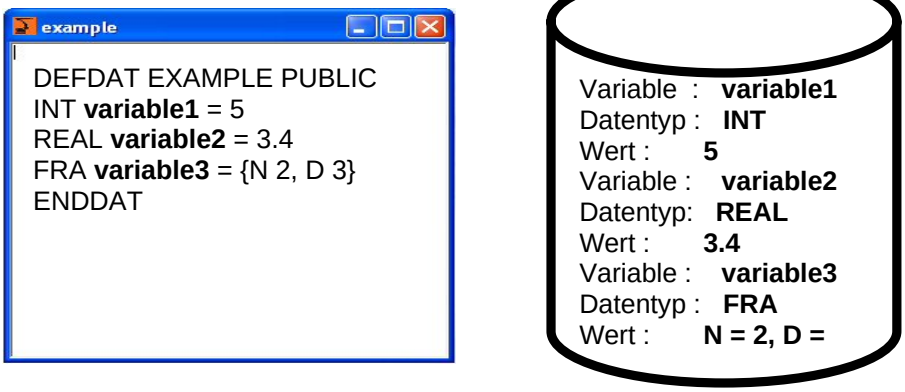
- Definition von Antriebsstrangparametern und kompletten Robotermodellen für Modelica zur Verwendung im Modellserver.
- Erzeugung von C–Header Dateien für xPC (siehe Kapitel 3.2.1): Einige fundamentale Kenngrößen des Roboters, die für Überwachungen, Umrechnung von Einheiten in „Antriebsstrangeinheiten" wie Geberinkremente, Strominkremente in SI-Einheiten verwendet werden, werden auf der xPC-Entwurfsumgebung schon zur Hochlaufzeit

benötigt. Die entsprechenden Parameter für die verschiedenen Robotertypen werden mittels KUMAC in C-Code für ein Header-File übersetzt.

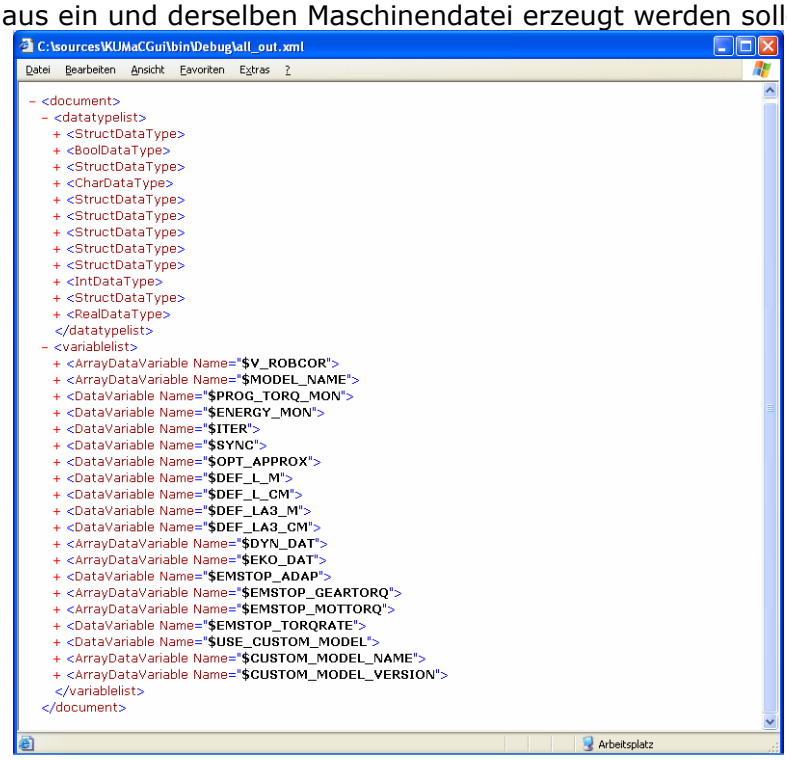
- „Punktabstraktion" aus Roboterprogrammen: KRL-Programme enthalten in der Regel Technologieanweisungen für Kleben, Schweißen usw. Für Auslegungszwecke sind aber nur die Bewegungen und Bewegungsparameter relevant. Aus KRL-Programmen werden daher reine Geometrieinformationen extrahiert und in einem abstrakten Zwischenformat abgelegt.

Im Folgenden werden die Teilfunktionalitäten des KUMAC beschrieben. Zunächst wird eine KUKA-interne Datei eingelesen, die die in KRL standardmäßig vorhandenen Datentypen beschreibt. Diese Typen stehen dann bei der weiteren Analyse von Maschinendaten oder anderern KRL-Dateien zur Verfügung.

Danach werden weitere selektierbare Dateien, in der Regel Maschinendaten-Dateien eingelesen. Diese enthalten Variablendeklarationen, Variablenzuweisungen und ggf. weitere Typdefinitionen. Die Variablentypen und -werte und zunächst im Speicher abgelegt:



Optional kann dieses interne Speicherformat in ein XML – Zwischenformat zur Speicherung der Daten umgewandelt werden. Diese XML - Datei kann entweder beim Kompilieren zur Weiteren Verwendung erzeugt werden, oder als Eingabe für die folgenden Schritte verwendet werden (und erspart dann das Parsen der Maschinendateien, wenn unterschiedliche Ausgabeformate aus ein und derselben Maschinendatei erzeugt werden sollen):

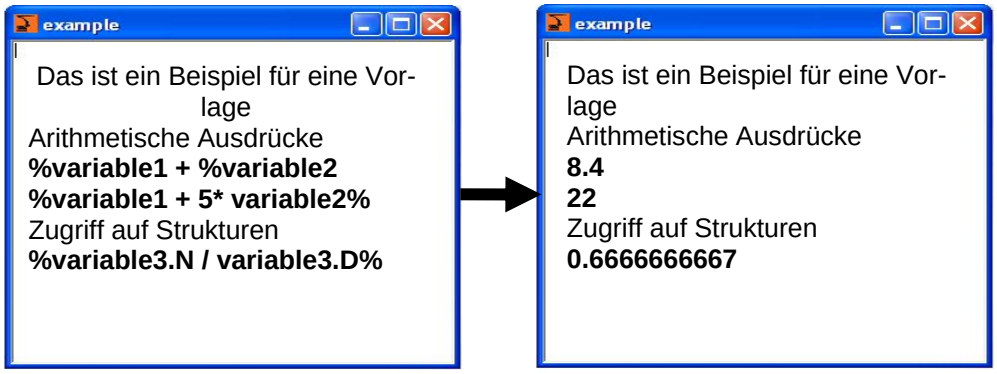


Danach wird in einer vom Anwender erzeugten Vorlagen-Datei („template") das gewünschte Zielformat beschrieben. Die template-Datei enthält

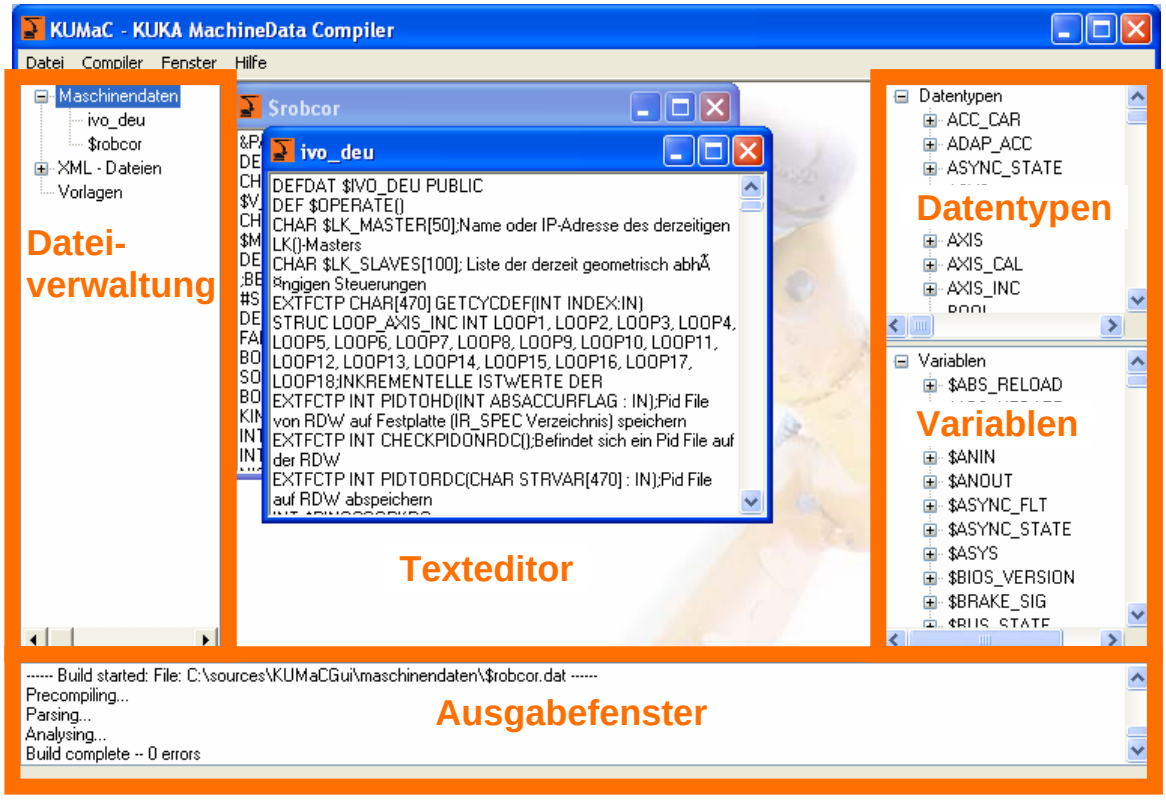
- normalen Text, der 1:1 in die Zieldatei übertragen wird,
- Platzhalter wie
  - Verweise auf Maschinendaten
  - arithmetische Ausdrücke
  - KRL-Variablen
  - Konstanten

Mittels der arithmetischen Operationen sind Umrechnungen z.B. von Einheiten, benutzerdefiniert möglich.

Der Template Compiler sucht dann alle Platzhalter in der Vorlage, ersetzt KRL – Variablen durch ihren Wert, berechnet arithmetische Ausdrücke und erzeugt die Ergebnisdatei. Im untenstehenden Beispiel erzeugt die Vorlagendatei links die Ausgabendatei rechts



Alle Operationen und Zwischenergebnisse sind in einer graphischen Oberfläche zu Diagnosezwecken sichtbar:



3.2 Echtzeitumgebung mit Hardware-in-the-Loop Optimierung

3.2.1 xPC-Target Echtzeitumgebung

Um mit dem PAPAS Modelserver entwickelte modellbasierte Steuerungs- und Regelungsalgorithmen an einem Kuka-Roboter testen zu können, wurde eine Hardware-In-The-Loop Umgebung basierend auf The MathWorks xPC Target realisiert [Kurz2006]. xPC Target ist eine Host-Target-Prototyping Umgebung der Firma The MathWorks, die es dem Benutzer ermöglicht, MATLAB Simulink Modelle mit Hardware zu verbinden und sie in Echtzeit auf PC-kompatibler Hardware auszuführen [Math2006]. In dieser Umgebung wird ein Rechner als Host PC verwendet, auf dem in Matlab Simulink Regelungsalgorithmen entwickelt werden. Mit Hilfe des Simulink-Export Moduls von Dymola können in Modelica/Dymola erstellte Modelle sehr komfortabel in Simulink als C-Code eingebunden werden (C-mex S-Function). Zum Beispiel ist es möglich, die Komponente des Reglers aus dem Gesamtsimulationsmodell entsprechend Abb. 3-4 in Simulink zu integrieren.

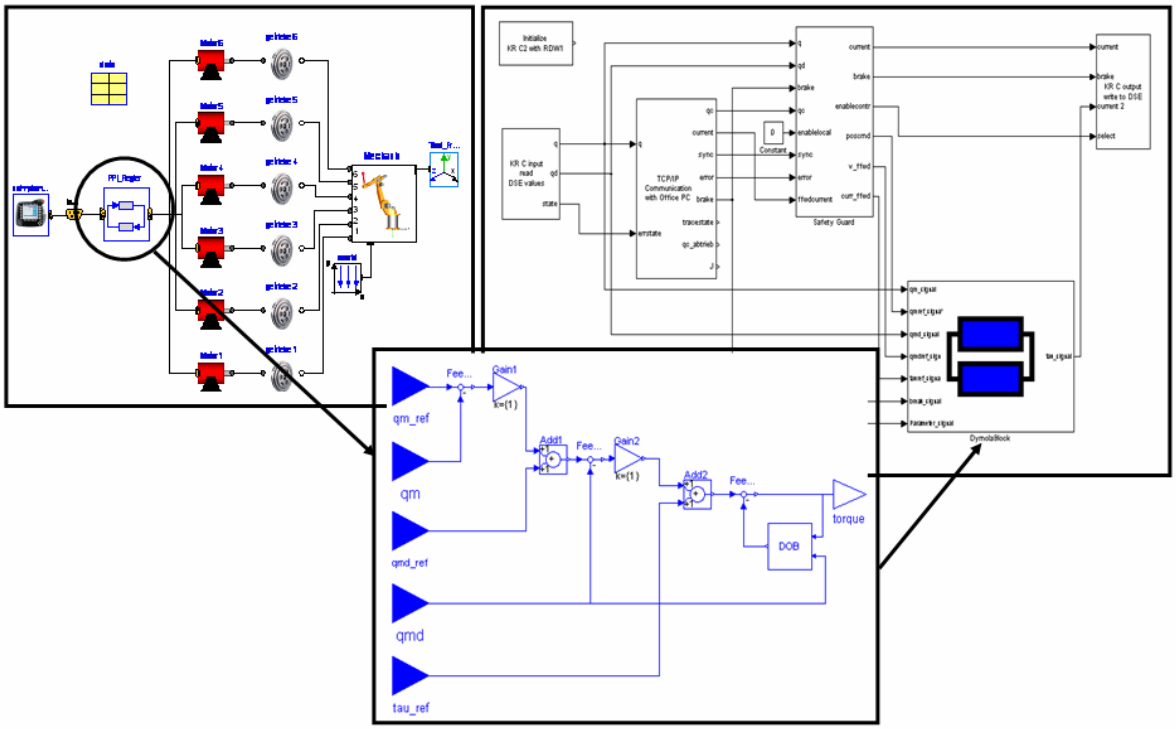


Abb. 3-4 Integration eines Modelica-Modells in Simulink

Nach der Integration des Modelica-Modells in Simulink kann mit Hilfe des Real-Time Workshop der Firma The MathWorks automatisch echtzeitfähiger Code generiert werden. Der echtzeitfähige Code wird anschließend auf den Target-PC geladen, auf dem das xPC Target-Echtzeit-Kernel läuft. In diesem Fall wird der Rechner im Steuerschrank des KUKA-Roboters als Target-PC verwendet. Die Steuerung und Regelung wurde innerhalb der entwickelten Echtzeitumgebung weiterhin so aufgeteilt, dass Bahnplanung und Benutzeroberfläche auf einem weiteren Rechner, dem sogenannten Office-PC, ausgeführt werden. Der Office-PC ist ein Standardprodukt der Firma KUKA für Offline-Simulation und Programmierung. Die Software entspricht der standardmäßig zur Robotersteuerung eingesetzten Software mit Ausnahme der Schnittstelle zu den elektrischen Antrieben. Auf diese Weise steht die gesamte Rechenleistung auf dem Target-PC im Steuerschrank zur Verfügung zur Berechnung von modellbasierten Steuerungs- und Regelungsalgorithmen. Die Kommunikation zwischen Host, Target und Office-PC läuft über Ethernet. Die beschriebene Echtzeitumgebung vereint die bekannte benutzerfreundliche Bedienoberfläche der Standard-KUKA-Steuerung mit den modernen Softwarewerkzeugen zum Reglerentwurf der Firmen Dymola und Matlab. Die Struktur der Echtzeitumgebung ist in
Abb. 3-5 dargestellt.

Weiterhin ist es möglich, zusätzliche Sensoren, beispielsweise zur Messung von Drehraten, in die Echtzeitumgebung zu integrieren. Standardmäßig sind Industrieroboter nur mit Sensoren zur Messung der Motorposition ausgestattet. xPC-Target unterstützt I/O-Boards führender Hardwarehersteller. So können analoge Sensorsignale mit Hilfe eines PCI I/O-Boards eingelesen und in digitale Signale umgewandelt werden. Durch Hinzufügen von zugehörigen I/O Schnittstellenblöcken aus der xPC Target Simulinkbibliothek können die gemessenen Signale beispielsweise zur Zustandsbeobachtung eingesetzt werden. xPC-Target bietet zudem die Möglichkeit, zeitliche Signalverläufe online oder offline nach Beendigung einer Echtzeitanwendung darzustellen. Dies erweist sich insbesondere beim Testen und Implementieren neuer Regelungsstrukturen als sehr hilfreich. Für die im folgenden Abschnitt beschriebene Hardware-In-The-Loop Optimierung ist es notwendig, dass Modellparameter während einer Echtzeitanwendung geändert werden können. Dies ist mit xPC Target auch möglich.

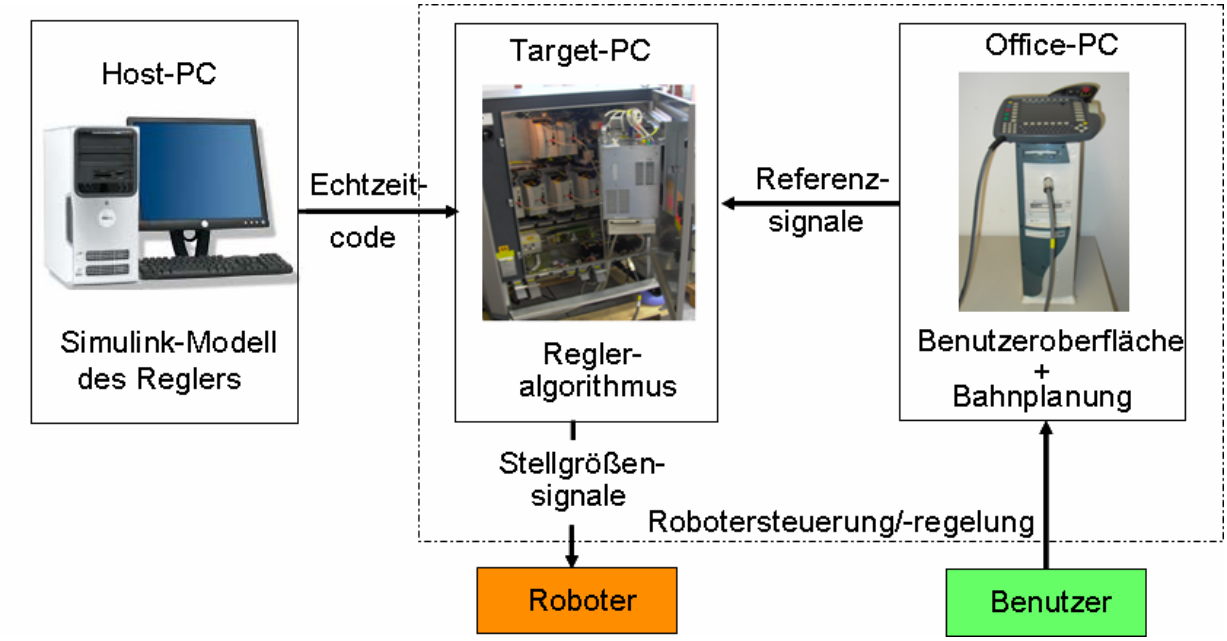


Abb. 3-5 Struktur der xPC-Target Echtzeitumgebung

3.2.2 Hardware-in-the-Loop Optimierung

Ein vollständiger Reglerentwurf basierend auf Simulationen mit nichtlinearen Modellen vom Roboter führt aufgrund von Modellabweichungen hinsichtlich Modellstruktur und Modellparametern nicht zu optimalen Ergebnissen bezüglich der Reglergüte. Daher wird im Rahmen einer sogenannten Hardware-in-the-Loop Optimierung eine Fein-Einstellung der Reglerparameter durchgeführt [Auer1997]. Zur Optimierung wird hierbei das DLR-Tool MOPS eingesetzt [Joos2005]. Der Ablauf einer solchen Hardware-in-the-Loop Optimierung ist in Abb. 3-6 skizziert.

Auf dem Entwurfs-PC (Host-PC) läuft ein Simulink-Modell des zu optimierenden Reglers. Mit Hilfe des Real-Time Workshop von der Firma The MathWorks wird automatisch echtzeitfähiger Code des Reglermodells erstellt. Dieser Code wird anschließend auf den Target-PC im Steuerschrank des Roboters geladen. Der Roboter fährt nun vorgegebene Referenzbahnen ab, wobei die Position und Orientierung des Tools mit Hilfe eines optischen Koordinatenmessgeräts vermessen werden. Die Messdaten werden eingelesen und ausgewertet. Basierend auf dem Vergleich zwischen gemessenen Bahnen und Referenzbahnen werden definierte Kriterien berechnet, die die Reglergüte wiedergeben. Mit Hilfe von in MOPS verfügbaren Optimierungsmethoden werden basierend auf den berechneten Kriterien neue Reglerparameter berechnet und das Experiment wiederholt. Auf diese Weise werden durch den Optimierer gesteuert iterativ die Reglerparameter verbessert. Im Allgemeinen werden zur Optimierung gradientenfreie Verfahren eingesetzt, da diese Verfahren unempfindlicher gegenüber Kriterien sind, die aufgrund des Messrauschens bei gleichen Fahrten zu unterschiedlichen Ergebnissen führen.

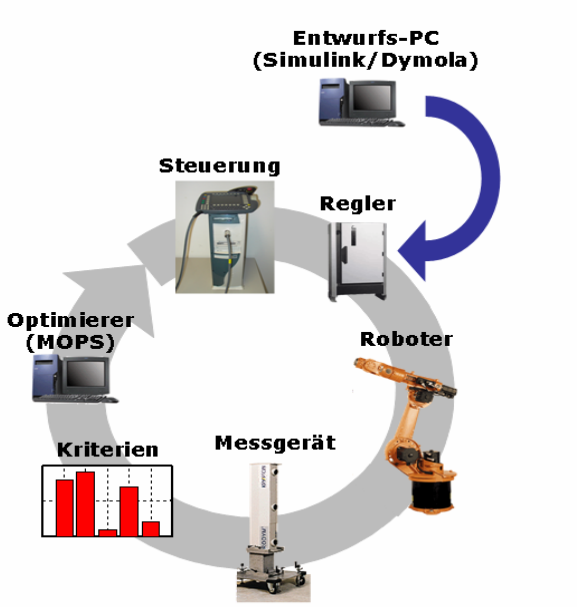


Abb. 3-6 Ablauf der Hardware-in-the-Loop-Optimierung

3.3 Entwurf modellbasierter Regelungen

In diesem Abschnitt wird exemplarisch an Hand einer nicht-linearen Vorsteuerung und einer Reglerkomponente (Disturbance Observer) gezeigt, wie die im PAPAS Projekt entwickelten Entwurfstechnologien für neuartige Roboterregelungen eingesetzt werden können.

3.3.1 Nichtlineare Vorsteuerung mit inversen Streckenmodell

Es gibt unterschiedlichste Verfahren zur modellbasierten Regelung. Eine wichtige Klasse von Verfahren baut auf einer Struktur mit 2 Freiheitsgraden auf [Krei1999]. Die zwei Freiheitsgrade bestehen hierbei aus einer Vorsteuerung und einem Regler. Für die Vorsteuerung bietet es sich an, ein inverses Streckenmodell zu verwenden. Im Idealfall, muss der Regler dann lediglich Störungen, die auf die Strecke wirken, sowie Modellfehler ausgleichen. Im linearen Fall ist die Invertierung einer Strecke meist unproblematisch solange die Strecke minimalphasig ist, also keine instabilen Nullstellen enthält. Es müssen dann lediglich Pole und Nullstellen vertauscht werden und die Strecke muss gegebenenfalls um eine Wunschübertragungsfunktion erweitert werden, damit das inverse System kausal bleibt.

Im nichtlinearen Fall gestaltet sich das Invertieren wesentlich schwieriger. Eine Möglichkeit zum Invertieren von nichtlinearen Systemen ist, das System als differential-algebraischen Gleichungssystemen (DAE) darzustellen und die Struktur (Zustände, unbekannte Variablen) des Systems so zu ändern, dass diese auf ein inverses System führt. Die Gleichungen werden dabei wie für ein Vorwärtsmodell aufgestellt, allerdings wird die Bedeutung der Variablen geändert. Eingänge und Ausgänge werden vertauscht, was zu einem inversen Modell führt. Grundlage hierzu ist der Algorithmus von Pantelides [Pant1988], mit dem bestimmt werden kann welche Gleichungen wie oft zu differenzieren sind, um das DAE System lösen zu können. Um inverse Modelle generiern zu können sind hierfür meist höhere Ableitungen der neuen Eingänge nötig, weshalb hier (ähnlich zum linearen Fall) die Strecke mit Filtern (Wunschübertragungsfunktion im Sinne von Kreisselmeier [Krei1999]) erweitert werden muss. Über den Algorithmus von Mattson und Söderlind [Matt1993] können die neuen Zustände für das inverse System gefunden werden („dummy derivate method").

Beide Algorithmen sind Voraussetzung für eine sinnvolle Modelica Simulationsumgebung, insbesondere auch von Dymola [Dyna2005]. Damit können in Modelica beschriebene Modelle (im Prinzip) leicht implementiert werden (im Detail gibt es Probleme). Abb. 3-7 zeigt schematisch die Invertierung eines einfachen Streckenmodells in Modelica. Die „twoInputs" bzw. „twoOutputs" Blöcke dienen zum Vertauschen von Ein- und Ausgängen.

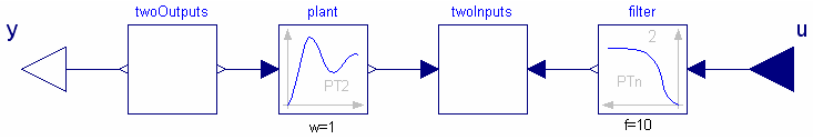


Abb. 3-7 Vertauschen von Ein- und Ausgangssignalen in Modelica.

Es gibt allerdings nicht für jedes Modell ein stabiles inverses System, bzw. eine eindeutige Lösung, weshalb in den Modellen teilweise bestimmte Approximationen vorgenommen werden müssen, um zu einem stabilen inversen System zu gelangen. Für ein allgemeines DAE-System existiert kein Stabilitätsbeweis, allerdings kann die Stabilität für einige Klassen von Systemen gezeigt werden, hierunter fallen auch Roboter mit elastischen Antrieben. Auch bestimmte Funktionen sind nicht invertierbar, müssen also vor der Invertierung entsprechend approximiert werden. Aktuator Begrenzungen können bei der Invertierung nicht berücksichtig werden, diese müssen aus den Modellen vor dem Invertieren entfernt werden.

Details zu dem skizzierten Vorgehen der Invertierung nichtlinearer Systeme sind in [Hoep2004, Thue2005, Looy2005] zu finden. In Abb. 3-8 ist der Einsatz von inversen, nichtlinearen Streckenmodellen in der Vorsteuerung zu sehen, die im PAPAS Projekt auf dem in Kapitel 3.1 skizzierten Modellserver basieren.

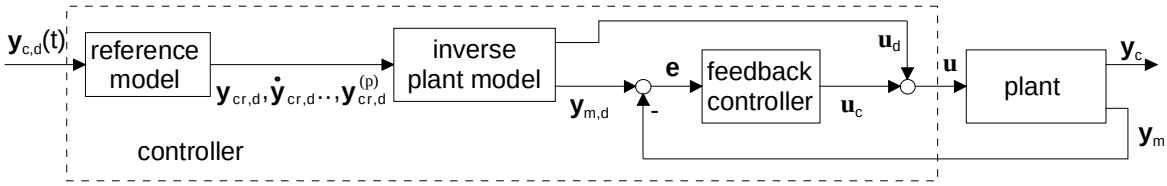


Abb. 3-8 Regler mit einem inversen, nichtlinearen Streckenmodell als Vorsteuerung

Das Konzept des Verfahrens kann am Beispiel eines einfachen Roboter-Antriebsstrangs demonstriert werden:

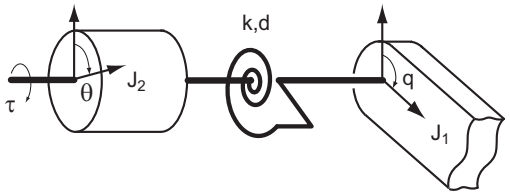


Abbildung 8: Struktur des Antriebstrangs (ohne Reibung)

Gleichung (1) beschreibt den Antriebstrang (vereinfacht ohne Reibung) wobei mit q der Abtriebsseitige Achswinkel und mit Θ der Motorwinkel beschreiben wird.

$$0 = J_1\ddot{q} + k(q-\theta) + d\cdot(\dot{q}-\dot{\theta}) + g(q)$$
$$\tau = J_2\ddot{\theta} - k(q-\theta) - d\cdot(\dot{q}-\dot{\theta}) \qquad (1)$$

Die Dämpfungskonstante wird im Beispiel mit d, die Federsteifigkeit mit k bezeichnet (hierbei kann k auch eine nichtlineare Funktion sein, um nichtlineare Getriebeeffekte simulieren zu können). Die Gravitation, in Abhängigkeit der Stellung, wird durch den Term g(q) beschrieben, J1 und J2 sind die Trägheiten des Abtriebs und des Antriebs.

Ziel ist es, die inverse Dynamik des Systems zu berechnen, daher das Motormoment τ in Abhängigkeit von q darzustellen. Dieses Gleichungssystem kann durch Differentiation und die Wahl neuer Zustände in die Form einer algebraischen Gleichung überführt werden (2).

$$\ddot{\theta} = \frac{1}{d}\left(J_1 q^{(3)} + \frac{\partial k}{\partial(q-\theta)}\cdot(\dot{q}-\dot{\theta}) + \frac{\partial g}{\partial q}\dot{q} \right) + \ddot{q} \qquad (2)$$

Hiermit kann nun Gleichung (1) bezüglich τ gelöst werden.

Für Industrieroboter mit elastischen Antrieben kann dieses Verfahren ebenfalls angewendet werden. Grundlage ist hierzu ein Modell vom PAPAS Modellserver, das sich objektorientiert

aus verschiedenen Komponenten zusammensetzt (Abb. 3-9). Die Roboter-Kinematik ist hierbei mit Starrkörpern modelliert, die Antriebsstränge bestehen jeweils aus einem nichtlinearer Feder-Dämpfer Element sowie an- und abtriebsseitiger Reibung. Der Motor ist als Trägheit modelliert, da seine Dynamik im Vergleich zur Mechanik vernachlässigt werden kann. Die gyroskopischen Kopplungsterme, welche durch die rotierenden Antriebsstränge auf die Roboter Kinematik wirken, werden ebenfalls vernachlässigt.

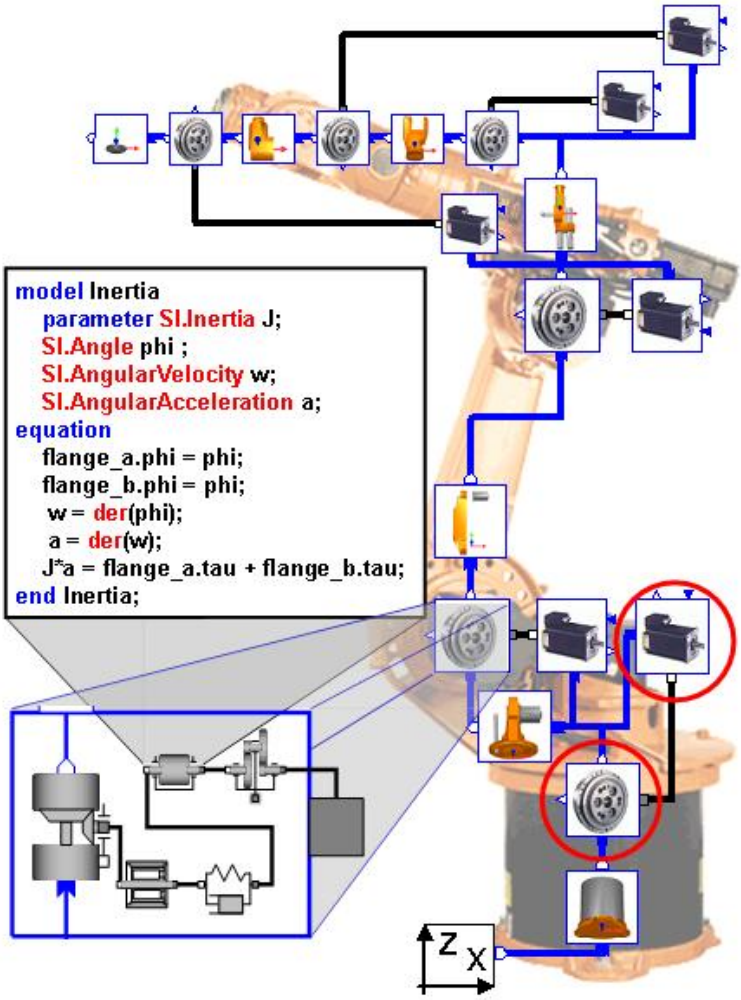


Abb. 3-9 Modelica Roboter Modell vom PAPAS Modellserver (schematisch)

Um dieses Modell invertieren zu können, müssen die Bewegungsgleichungen zweimal differenziert werden. Die Ableitungen der neuen Eingangsgrößen (abtriebsseitige Achswinkel qa) werden über einen Tiefpassfilter erzeugt.

Das hierdurch gewonnene inverse Modell kann zur Vorsteuerung verwendet werden, um die Motormomente (τ) in Abhängigkeit der Abtriebsseitigen Winkel (qa) zu berechnen. In Abb. 3-10 sind Messungen von Experimenten zu sehen (aus [Thue2005]) bei der die Vibrationen der Roboterhand dargestellt werden. Dieselbe Sollbahn wird hierbei einmal mit einem einfachen PD Regler und das zweite Mal mit einem PD Regler mit nicht-linearer Vorsteuerung durchfahren.

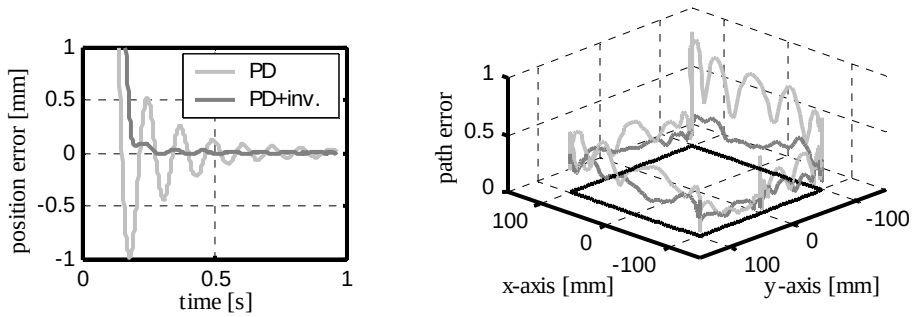
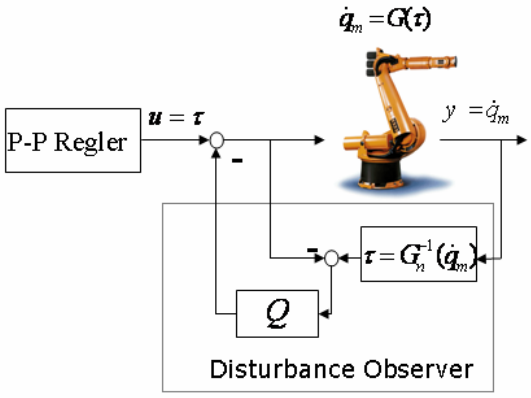


Abb. 3-10 Vergleich von Reglerfehlern.
Helle Linien: PD-Regler ohne Vorsteuerung. Dunkle Linien: PD-Regler mit nichtlinearer Vorsteuerung. Linkes Bild: Positionierung einer PTP-Bahn. Rechtes Bild: Bahnfahren

Es ist dabei deutlich zu sehen, dass das inverse Robotermodell die Vibrationen beim Positionieren stark reduziert. Im rechten Teil von Abb. 3-10 ist ein Bahnfahrt-Experiment zu sehen. Die gewünschte Trajektorie ist hierbei ein Quadrat das durchfahren werden soll. Wiederum wird die Bahn mit und ohne nichtlinearer Vorsteuerung durchfahren, wobei die Vibrationen der Roboterhand mit einem optischen Koordinatenmessgerät vermessen werden. Im Bild ist die normalisierte absolute Abweichung von der Sollbahn senkrecht zur Bahnebene aufgetragen. Auch hier ist zu sehen, dass das nicht-lineare inverse Robotermodell in der Vorsteuerung die Bahnfehler stark reduziert.

3.3.2 Disturbance Observer

Der Disturbance Observer wurde für lineare Systeme entwickelt, um die Robustheit gegenüber Störungen und Modellierungsfehlern zu erhöhen [Umen1991]. Die Struktur kann durch die Verwendung eines nichtlinearen inversen Referenzmodells auch bei nichtlinearen Systemen angewendet werden [Looy2005]. In Abb. 3-11 ist die Struktur einer Gelenkregelung basierend auf dem Prinzip des Disturbance Observers dargestellt.



G_n : Wunschübertragungsverhalten
Q : Tiefpassfilter 2. Ordnung

Abb. 3-11 Struktur des Disturbance Observers

Der Disturbance Observer schätzt die auf das System einwirkende Störung unter Verwendung eines inversen Modells der nominellen Regelstrecke. Eingang des inversen Modells ist in diesem Fall die gemessene Gelenkdrehzahl. Aus dem Vergleich des Reglermoments mit dem Ausgang des inversen Modells ergibt sich das geschätzte Störmoment. Die Genauigkeit des geschätzten Störmoments nimmt dementsprechend mit zunehmender Übereinstimmung von Regelstrecke und gewähltem nominellem Modell zu. Daher wird bei nichtlinearem Übertragungsverhalten ein nichtlineares nominelles Modell eingesetzt.
Der P-P Regler stellt einen in der Antriebstechnik weit verbreiteten Kaskadenregler zur Positionsregelung dar. Er beinhaltet eine innere Regelschleife zur Drehzahlregelung und eine äußere Schleife zur Positionsregelung. Entsprechend dem Prinzip einer Kaskade wird der

Ausgang der Positionsregelschleife auf die Drehzahlregelungsschleife aufgeschaltet. Die Auslegung dieses Reglers erfolgt auf der Basis des nominellen Systems, wobei davon ausgegangen wird, dass keine Störungen auf das System einwirken.
Q ist ein Tiefpassfilter. Er wird verwendet, um das inverse System realisieren zu können, da das System selbst kausal ist. Weiterhin wird durch den Einsatz des Filters der Einfluss des Messrauschens reduziert.

Die geschätzte Störung wird entsprechend Abb. 3-11 vom Reglermoment subtrahiert um den Einfluss der Störung zu unterdrücken. Innerhalb der Bandbreite des Q-Filters verhält sich das System mit integriertem Disturbance Observer wie das nominelle System ohne Einfluss von Störungen. Die realisierbare Bandbreite des Q-Filters ist aufgrund des Messrauschens begrenzt.

3.4 Literatur

[Auer1997] Auer, E.; Bals, J.; Joos, H.-D.: **Automatische Reglereinstellung bei Robotern in einem ISO-Inbetriebnahmeprozess mit mehrzieliger Optimierung.** In: 2. VDI/VDE-GMA-Aussprachetag Rechnergestützter Entwurf von Regelunssystemen, 1997.

[DeLu1996] De Luca, A. and Tomei, P.: **Elastic joints**, in *Theory of robot control*, de Wit, C. C., Siciliano, B. and Bastin, G., ed., Springer Verlag, Berlin, Heidelberg, New York, Tokyo, pp. 179-217, 1996.

[Dyna2005] Dynasim: Dymola – Users Manual; http://www.dynasim.com, 2005

[Hoep2004] Höpler, R. and Thümmel, M.: **Symbolic Computation of the Inverse Dynamics of Elastic Joint Robots**, *Proc. of the 2004 IEEE Intern. Conference on Robotics and Automation* pp 4314-4319, 2004.

[Joos2005] Joos, H.-D.: **Mops- Multi-Objective Parameter Synthesis.** User's Guide V5.0. Oberpfaffenhofen, 82234 Wessling: German Aerospace Center, Institute for Robotics and Mechatronics, 2005.

[Krei1999] Kreisselmeier, G.: **Struktur mit zwei Freiheitsgraden**. *Automatisierungstechnik* **6**:266-269, 1999.

[Kurz2006] Kurze, M., Weiss, M., Otter, M.: **Methods and Tools to design and test robot control systems**. In: Proceedings of the Joint Conference on Robotics: ISR 2006 and Robotik 2006, 2006.

[Looy2005] Looye G., Thümmel M., Kurze M., Otter M., Bals J.: **Nonlinear Inverse Models for Control.** In: Proceedings of the 4th International Modelica Conerence, 2005, pp. 267-279. Download from: http://www.modelica.org/events/Conference2005/online_proceedings/Session3/Session3c3.pdf

[Math2006] The MathWorks Deutschland: **Online Documentation for Matlab and the MathWorks products** www.mathworks.de/access/helpdesk/help/helpdesk.html.

[Matt1993] Mattsson, S.E. and Söderlind, G., 1993: **Index reduction in differential-algebraic equations using dummy derivatives**, *SIAM Journal of Scientific and Statistical Computing* **14**:677-692.

[Mode2005] Modelica Association: **Modelica® - A Unified Object-Oriented Language for Physical Systems Modeling**. Tutorial,Version 1.4; http://www.modelica.org.

[Pant1988] Pantelides, C.C.: **The consistent Initialization of differential-algebraic Systems**, *SIAM Journal of Scientific and Statistical Computing* **9**:213-231, 1988.

[Thue2005] Thümmel, M., Otter, M., Bals, J.: **Vibration Control of Elastic Joint Robots by Inverse Dynamic Models**, IUTAM Symposium on Vibration Control of Nonlinear Mechanics and Structures. Eds. H. Ulbrich and W. Günther, pp. 343 – 353, Springer Verlag, 18. – 22. Juli 2005.

[Umen1991] Umeno T., Hori Y.: **Robust speed control of dc servomotors using modern two degrees-of-freedom controller design**. *IEEE Trans. Ind. Electron*, 38-5, pp. 363-368, 1991.

4 Nachgiebiger Leichtbauroboter als Produktionsassistent

Der dritte Schwerpunkt innerhalb des PAPAS-Projektes war die Weiterentwicklung eines nachgiebigen und einfach zu programmierenden Leichtbauroboters als Produktionsassistent. Als Basis hierfür diente der am DLR entwickelte LBR III. Dieser verfügte zu Beginn des Projektes über eine entsprechende Basissteuerung. Diese ermöglicht die Regelung des LBRs in diversen Modi. Neben der klassischen Positionsregelung ist u.a. auch eine Impedanzregelung implementiert. Eine besonders für den Anwendungsbereich „Produktionsassistent" interessante Betriebsart ist die sogenannte „Gravitations-Kompensation". Hierbei werden die Momente der einzelnen Achsen so geregelt, dass der Roboter sich gerade noch selbst halten kann. Jede extern aufgebrachte Kraft führt nun dazu, dass der Roboter „ausweicht". Dadurch lässt sich der Roboter sehr intuitiv mit der Hand führen.

Wesentlicher Nachteil des damals existenten Systems war es, dass es keine übergelagerte Ablauf- und Bahnsteuerung gab, die im Stile einer Industrierobotersteuerung die Programmierung komplexer Anwendungen ermöglicht. Daher sollte im Rahmen des PAPAS-Projektes die DLR-Basissteuerung an die KUKA Robotersteuerung (KRC) angebunden werden, um die Vorteile beider Welten zu kombinieren und so einen entscheidenden Schritt in Richtung „Produktionsassistent" zu tun.

Im Nachfolgenden werden zuerst die Anforderungen bzw. die Vision bzgl. eines Produktionsassistenten dargestellt. Anschließend wird auf die Realisierung der Steuerungskopplung sowie auf die Produktion einer ersten Prototypenreihe im Hause KUKA eingegangen. Zum Abschluss werden beispielhafte Demonstratoren, wie sie im Rahmen von PAPAS bzw. aufbauend auf den PAPAS Ergebnissen realisiert wurden.

4.1 Zukunftstrend: Produktionsassistent

In den letzten Jahren gab es im Bereich Industrieroboter einen klaren Trend weg vom klassischen Roboter, der im Wesentlichen in der Automobilbranche und dort meist im Rohkarossenbau eingesetzt wird, hin zum Roboter für die General Industrie. Aber neben diesen neuen Branchen und Anwendungsfeldern werden Roboter inzwischen auch in nichtproduzierendem Gewerbe, wie zum Beispiel Medizin, Entertainment, Lebensmittel oder Logistik eingesetzt.
Als langfristiges Ziel der Robotik wird weltweit an persönlichen Servicerobotern im Home-Bereich geforscht und gearbeitet. Doch zur Erreichung dieses Ziels, insbesondere hinsichtlich vermarktbarer Produkte, sind noch viele Probleme zu lösen. Mittelfristig zeichnet sich auf den Weg dorthin ein Zwischenziel ab, welches, aufgrund seines hohen wirtschaftlichen Potentials, ein entscheidender Durchbruch im Robotikbereich sein dürfte: Der Produktionsassistent als Vielzweck-Werkzeug in produzierenden kleinen und mittleren Unternehmen (KMU) (siehe Abb. 4-1).

Wesentliches Element dürfte hierbei der vermehrte Einsatz von Sensorik zur Erfassung der Umwelt (Bauteile, Werkzeuge, Personen, …) sein. Der DLR LBR III bietet sich aufgrund der integrierten Momentensensorik in jeder Achse als Grundbaustein eines solchen Produktionsassistenten an. Mit der damit zu erreichenden Nachgiebigkeit lassen sich neue Anwendungsfälle für Roboter erschließen. Aber auch seine Leichtbauweise und die damit einhergehende Verringerung des Risikos bei Roboter-Mensch-Kontakten machen ihn zu einem herausragenden Kandidaten. Hinzukommt, dass aufgrund der 7-Achskinematik der Bewegungsspielraum des LBR im Vergleich zu 6-Achskinematiken erhöht ist.

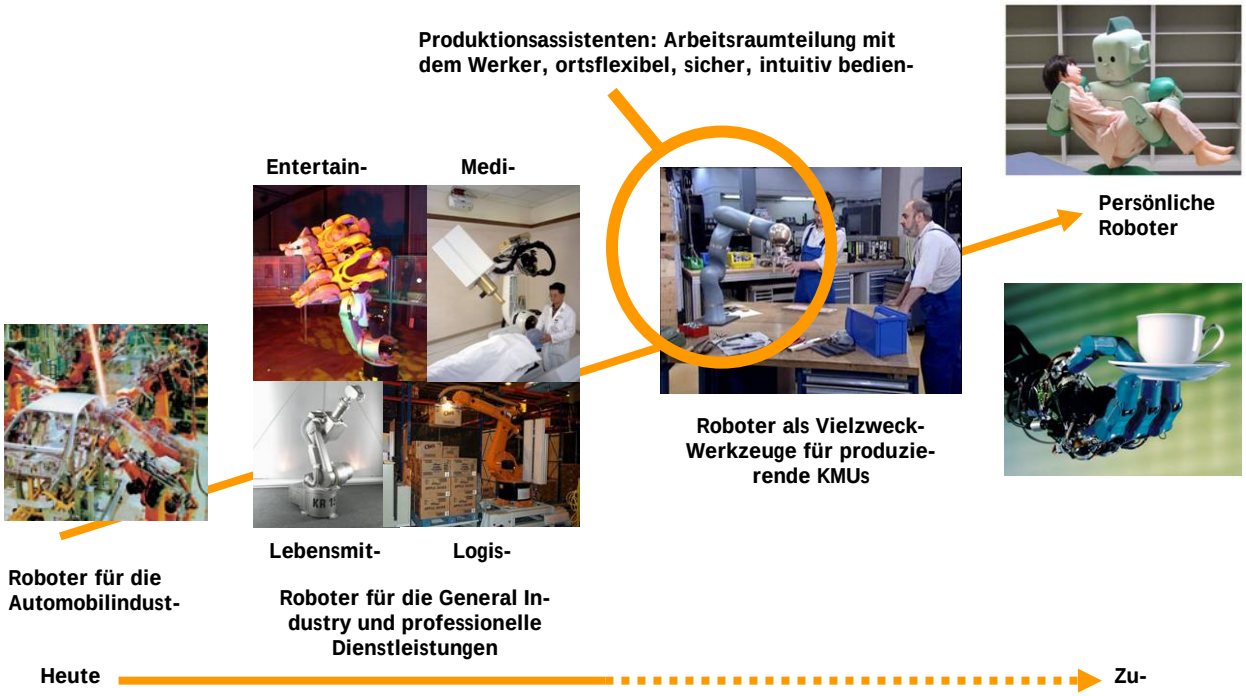


Abb. 4-1 Next Step: Produktionsassistent

Neben diesen grundlegenden Eigenschaften des LBRs gibt es eine Reihe von Eigenschaften, die einen zukünftigen Produktionsassistenten von herkömmlichen Industrierobotern, wie sie heute eingesetzt werden, unterscheidet:

„Klassischer" Industrieroboter	**zukünftiger Produktionsassistent**
fest installiert	ortsflexibel einsetzbar
zyklisch wiederkehrende Aufgabe, seltene Änderungen	Aufgaben selten wiederkehrend, häufige Änderungen
online / offline programmiert durch Roboterspezialisten	online instruiert durch Prozessexperten unterstützt durch Offline-Methoden
kaum Interaktion mit Werker, wenn dann Programmierung	häufige Interaktion mit Werker, auch Kraft-/Präzisionsunterstützung
trennende Schutzeinrichtung	Teilung des Arbeitsraums mit dem Werker
rentabel nur bei mittleren bis großen Losgrößen	rentabel selbst bei kleinen Losgrößen

Tabelle 4-1 Features eines Produktionsassistenten

Man kann also deutlich erkennen, dass eine Kombination der DLR Basissteuerung und LBR Technologie mit der KUKA Ablauf-/Programmiersteuerung und Bedienung nur ein erster, wenn auch wesentlicher Schritt in Richtung „Produktionsassistent" ist. Die neuesten Robotertechnologien (Leichtbau, Nachgiebigkeit, Redundanz) verbunden mit dem „Look and Feel" einer modernen Industrierobotersteuerung (Ablaufsteuerung, Roboterprogrammierung mit KRL, Bedienung mit Programmierhandgerät) ist grundlegende Voraussetzung zur Schaffung der oben genannten Eigenschaften. Dennoch sind weitere Arbeiten, vor allem im den Bereichen „intuitive Programmierung" und „sichere Mensch-Roboter-Interaktion" auch nach erfolgreichen Abschluss des PAPAS-Projektes von Nöten.

Auf der AUTOMATICA 2004 wurde ein erster Prototyp des PAPAS-Demonstrators vorgeführt. Hierbei wurde ein DLR LBR verbunden mit der KUKA KRC und dessen Potential als Fügeroboter wurde vorgeführt (siehe Abb. 4-2).

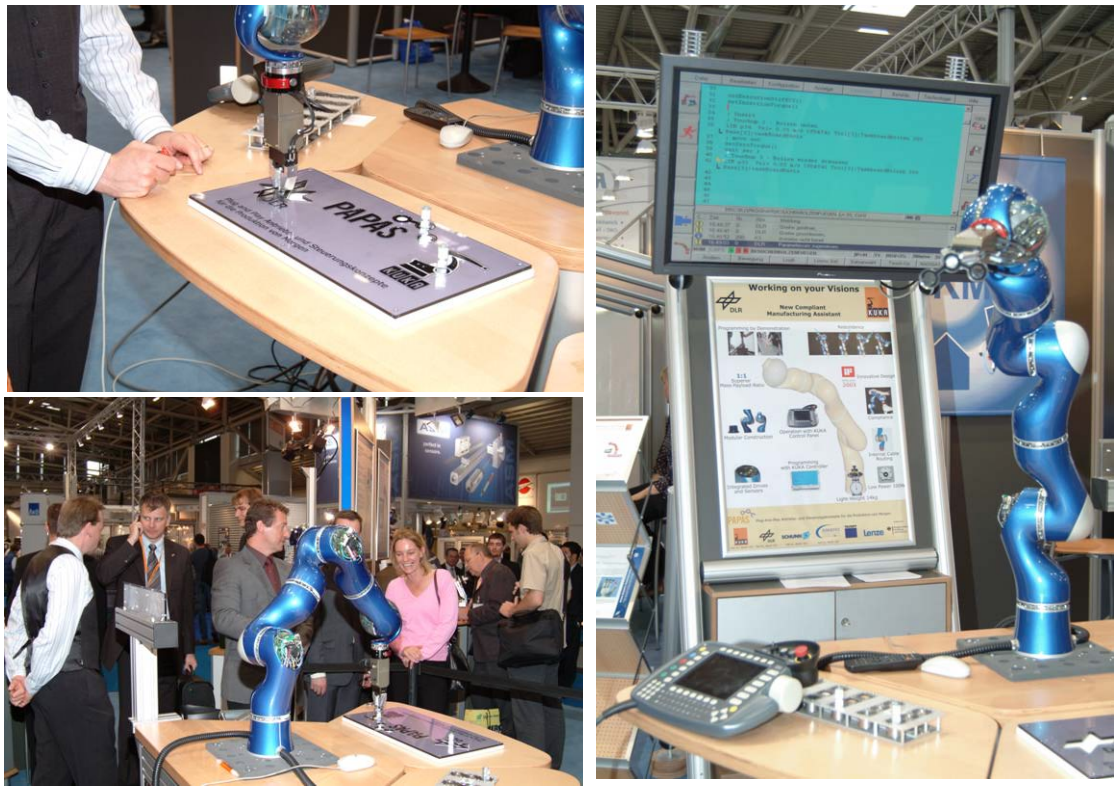


Abb. 4-2 Automatica 2004

Insgesamt war das Feedback der Besucher sehr gut. Dies zeigte sich auch anhand der auf der AUTOMATICA 2004 durchgeführten Befragung. Hierfür wurde ein Fragebogen verteilt, um herauszufinden, ob ein Bedarf an Produktionsassistenten vorhanden ist, welche Features ein Produktionsassistent benötigt und welche Anwendungsfelder für einen Produktionsassistenten geeignet sind. Aber auch Fragen bzgl. Plug and Produce wurden gestellt.

Weit über 300 Interessenten haben sich an den vier Ausstellungstagen über den neuen DLR LBR III informiert. Jedoch war der Rücklauf ausgefüllter Fragebögen mit einer Anzahl von 44 doch recht gering. Dies lag wohl maßgeblich daran, dass der Fragebogen zu umfangreich war. Dennoch lieferte die Auswertung einige interessante Anhaltspunkte:

- Das „Wichtigste" von den vorhandenen Features war aus Sicht der Befragten, die Möglichkeit den Roboter durch manuelles Führen zu programmieren (Durchschnittsbewertung: 1,33)[4] gefolgt von der Nachgiebigkeit (1,40). Auf den nächsten beiden Plätzen folgten die Eigenschaften bzgl. der hohen Integration, wie interne Kabelführung (1,58) und integrierte Sensorik (1,63). Weiterhin war die Leichtbauweise ein wichtiges Feature (1,71)

- Als weniger wichtig, aber mit immer noch hohem Durchschnittswert, wurden folgende Punkte bewertet. Redundanz (2,35), Design (2,37) und Programmierung/Bedienung mit KUKA Steuerung (2,48).

- Die „schlechte" Bewertung der Programmierbarkeit mit KUKA KRL lässt sich darauf zurückführen, dass dieses Feature im Rahmen des Demonstrators nicht vordergründig gezeigt

[4] Mögliche Wertung von 1 (important) bis 4 (unimportant)

wurde und eigentlich nur von entsprechenden Fachleuten bewertet werden kann. Die Erläuterung der Programmiermöglichkeiten über die KUKA Steuerung wäre sehr aufwändig gewesen. Andererseits zeigt dies auch auf, dass eine Verbesserung der Programmierbarkeit eines Produktionsassistenten ein sehr zentrales Forschungsthema ist.

- Plug and Play wurde als ein sehr wichtiges Thema (1,65) eingestuft (siehe Abb. 4-3).

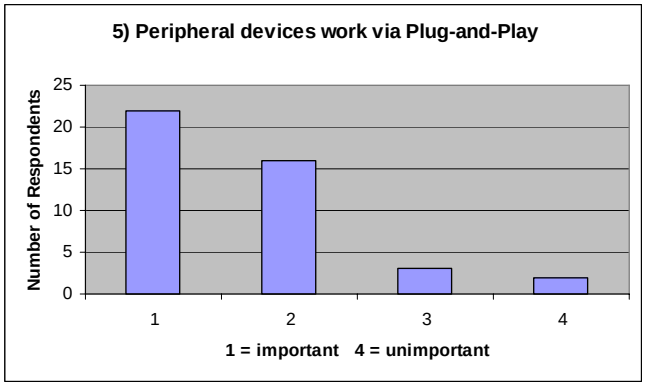


Abb. 4-3 Auswertung bzgl. Plug and Play

- Eine noch wichtigere Rolle spielt für die Anwender die direkte Mensch-Roboter Interaktion, welche im Schnitt mit 1,43 bewertet wurde.

- Bezüglich potentieller Anwendungsgebiete gab es keine „Überraschungen". Es wurden im Wesentlichen die schon im Vorfeld bekannten Applikationen genannt (siehe Abb. 4-4).

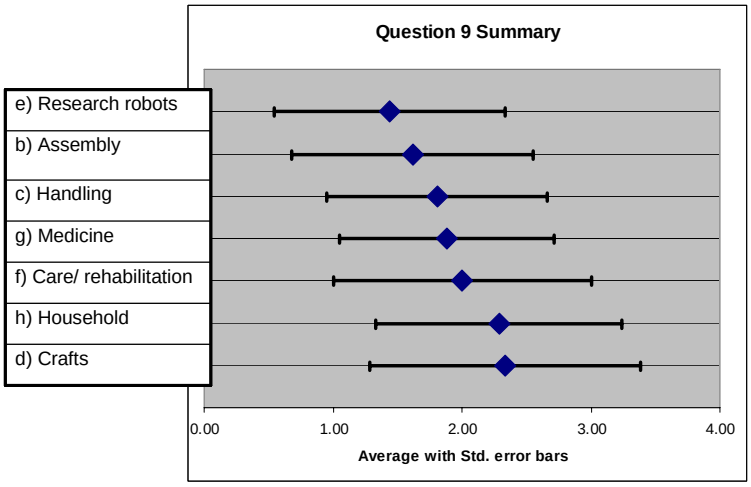


Abb. 4-4 Potentielle Anwendungsfelder

4.2 Anbindung DLR Basissteuerung und KUKA KRC

Um dem damaligen Forschungsroboter DLR LBR ein industrielles Look&Feel zu geben, wurden im Projekt PAPAS die KUKA KRC und der LBR zusammengebracht. Von der KRC Seite aus können die Eigenschaften des Roboters (Nachgiebigkeit, direkte Interaktion, kinematische Redundanz) genutzt werden. Dazu werden auf der LBR Seite Algorithmen eingesetzt, die in der DLR Basissteuerung integriert sind.

Zur Kopplung dieser DLR Basissteuerung mit der KUKA KRC wurde ein Protokoll vereinbart, mit dem Steuerinformationen und Messwerte über ein Medium – hier EtherNET/UDP – ausgetauscht werden.

Die Kopplung an sich ist relativ lose, zur Synchronisation werden die verschickten UDP Pakete genutzt. Im Empfänger sind jeweils FIFOs enthalten, deren Füllstand wird zur Synchronisation der Systeme genutzt.

Abb. 4-5 PAPAS Demonstrator

4.2.1 Die „PAPAS"-Schnittstelle

Die „PAPAS" Schnittstelle erlaubt den Austausch von Informationen zwischen KUKA KRC und DLR LBR Basissteuerung. Hierbei werden sowohl azyklische als auch zyklische Kommandos von der KRC gesendet. Zurück kommen Informationen bzgl. des Zustands und aktuelle Messwerte sowie bzgl. der Taktung (siehe Abb. 4-6).

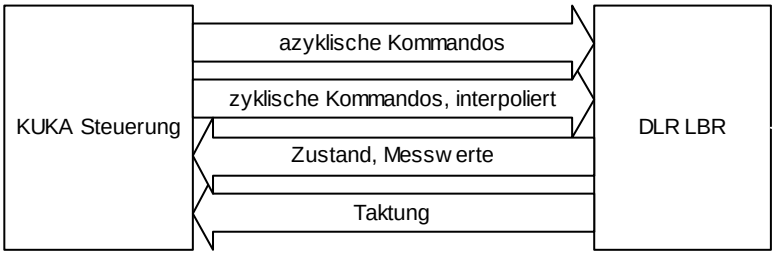


Abb. 4-6 Die PAPAS Schnittstelle

Der DLR LBR kann in den folgenden Betriebsarten kommandiert werden:

- Kartesisch Interpoliert, hierzu wird zyklisch pro Interpolationstakt ein neues kartesisches Kommando von der KUKA Steuerung erzeugt und an den DLR LBR weitergeleitet.
- Joint Level Interpoliert, hierzu werden pro Interpolationstakt zyklisch neue Gelenkpositionen von der KUKA Steuerung erzeugt und an den DLR LBR weitergegeben.
- Kartesisch PointToPoint (PTP), hierzu wird das kartesische Sollziel an den DLR LBR gegeben, dieser errechnet die inverse Kinematik und verfährt an den Zielpunkt.

In den jeweiligen Betriebsarten kann der LBR positions- oder steifigkeitsgeregelt verfahren. Daher müssen zu jedem Bewegungssatz (azyklisch) die folgenden Parameter definiert werden.

Qualitative Beschreibung der Information	**Bedeutung**	**(KRC) Systemvariable** `$stiffness` `STRUC STIFFNESS`
Art des Reglers	Auswahl von Positions-/ Steifigkeitsregelung. DLR-seitig werden Regler vorkonfiguriert, und von der KUKA Steuerung ausgewählt	`STRATEGIE`
Kartesische Steifigkeit	Der Einfachheit halber wird diese als Diagonalmatrix spezifiziert und mit einem zusätzlichen „TaskFrame" versehen	`CPSTIFFNESS`
Kartesische Sollkraft		`CPDESIREDFORCE`
Kartesische Dämpfung	Diagonalmatrix	`CPDAMPING`
Nullraumsteifigkeit		`AXISCPSTIFFNESS`
Nullraumdämpfung		`AXISDAMPING`
Gelenksteifigkeit/ Dämpfung	entsprechen der Nullraumsteifigkeit/ Dämpfung	`AXISSTIFFNESS`
Sollkonfiguration	Ziel – Gelenkstellung	`AXISCONFIG`
Auswahl Bitfeld für die Sollkonfiguration,	um nur einzelnen Gelenken eine Sollkonfiguration vorzugeben	`AXISDESIREDCONF`
Kartesisches Verhalten in der Nähe von Singularitäten	vor allem für Positionsgeregeltes Verfahren von kartesischen Trajektorien	`CPRESTRICTED`
Nebenbedingungen der Inverskinematik	diese werden DLR seitig vorkonfiguriert und von KUKA Steuerung ausgewählt	`CPNULLID`
Nullraumverhalten	Nullraumbewegung in Ruhe (kein kartesischer Bewegungsbefehl aktiv) Nullraumbewegung im Admittanzmodus, d.h. das Anfassen des LBR führt zu Nullraumbewegungen	`CPNULLMODE`

Tabelle 4-2 Parameter der Systemvariablen $stiffness

Zur Kopplung werden die folgenden Informationen zyklisch benötigt:

Sollpositionen Kartesisch	Frame
Sollkonfigurationen – Gelenkwinkel	7-Vektor – E6AXIS

Darüber hinaus muss die KUKA Steuerung in die Lage versetzt werden, die Antriebe des DLR LBR zu aktivieren und zu deaktivieren.

Antriebe an/aus	Bool

Der KUKA Steuerung können die folgenden Informationen zur Verfügung gestellt werden:

gemessene Gelenkpositionen	$POS_ACT	7-Vektor, E6AXIS
gemessene Gelenkmomente	$TORQUE_AXIS_ACT	7-Vektor, E6AXIS

gemessene Kraft am TCP	$TORQUE_TCP_ACT	Frame
Regleraktivität/ Zustände		
Eventuelle Fehlermeldungen – Hardwareausfall etc..		

Darüber hinaus werden azyklische Sonderbotschaften (USER_DEF) in beiden Richtungen im Rahmen des PAPAS Schnittstelle realisiert.

4.2.2 Logischer Datenfluss

In der folgenden Prinzipskizze sind die Datenströme, die zwischen Basissteuerung und KRC ausgetauscht werden dargestellt. Einzelne Betriebsarten werden im Folgenden diskutiert. In der Skizze wird jeweils die Basissteuerung betrachtet. Eingehende Pfeile werden mit Daten von der KRC versorgt, ausgehende Pfeile sind Messwerte und Zustandsinformation der Basissteuerung, die auf der KRC wiederum verarbeitet werden (siehe Abb. 4-7).

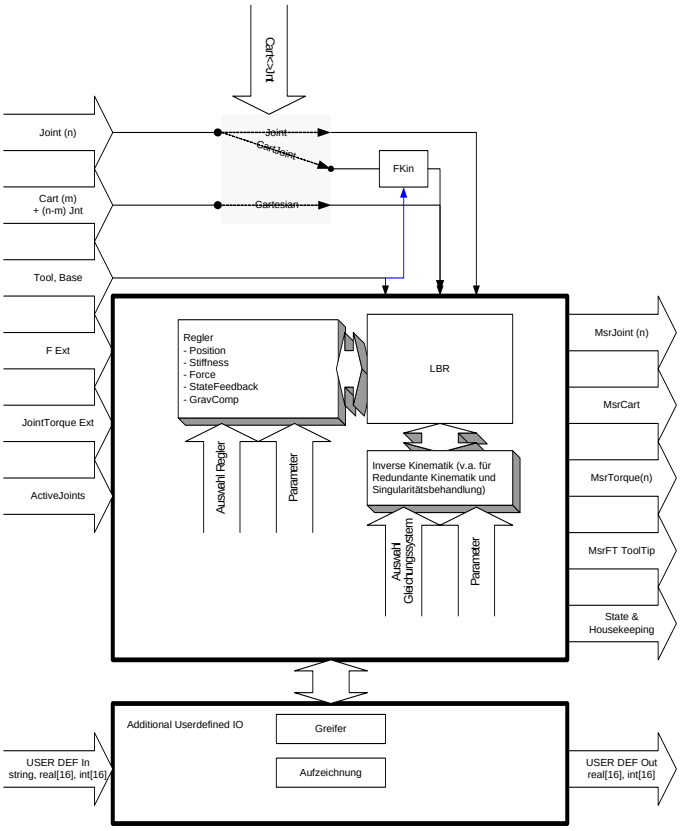


Abb. 4-7 genereller Datenfluss

Basisbetriebsart: Positionsregler im Gelenkmodus

Der Positionsregler im Gelenkmodus erwartet am Eingang einen 7-Komponenten-Vektor als Gelenkinformation (Sollpositionen – Vorschübe) und erwartet die Betriebsart „Joint". Als Rückmeldung von der Basissteuerung werden die Messwerte 7-Komponenten-Vektor Gelenkinformation und der Zustand (Housekeeping) des Roboters an die KRC übergeben.

Diese Betriebsart wird beim Systemstart ausgewählt. Die Reglerauswahl selbst wird durch eine Indexnummer realisiert. Bei nicht hinterlegten Nummern wird automatisch diese Betriebsart aktiviert.

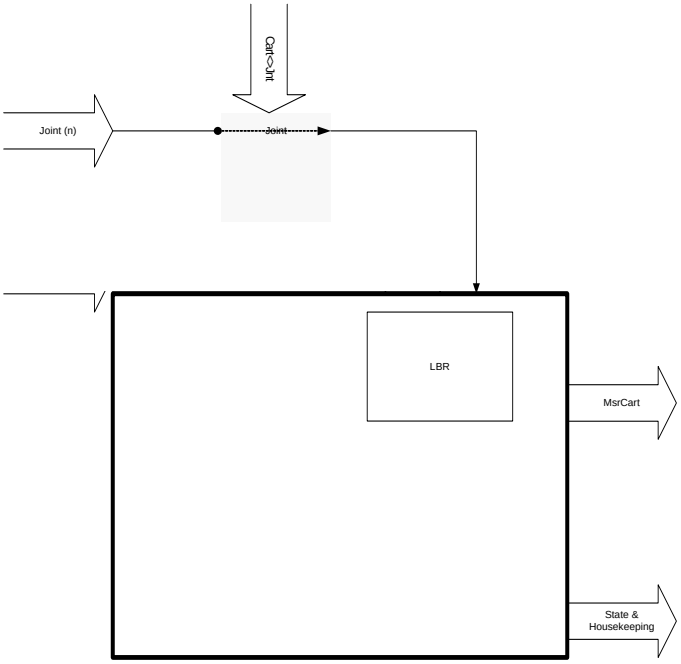


Abb. 4-8 Datenfluss Positionsregler im Gelenkmodus

Positionsregelung Kartesisch

In der kartesischen Positionsregelung werden zyklisch das kartesische Ziel und die Koordinatensysteme (Base&Tool) übertragen. Azyklisch wird zusätzlich eine Information übertragen, wie mit der kinematischen Redundanz verfahren werden soll.

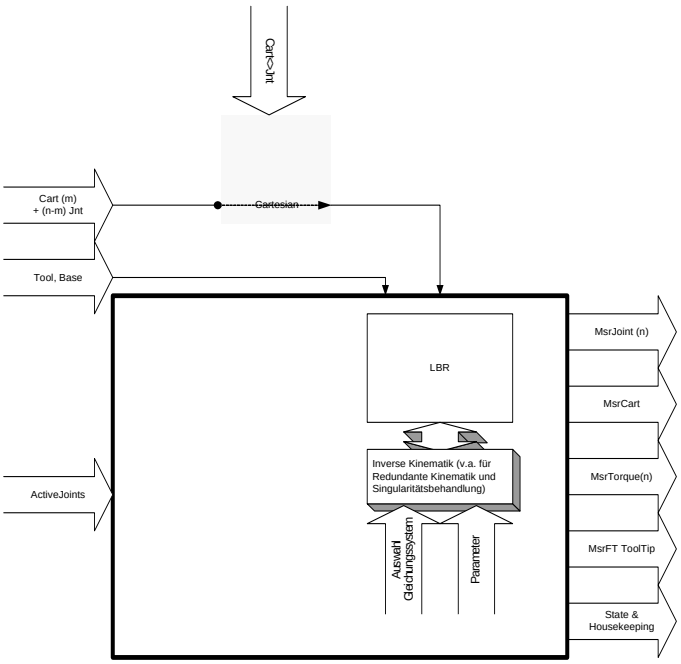


Abb. 4-9 Datenfluss Positionsregler kartesisch

Sonderfall Positionsregelung CartJoint
Dieser Fall verhält sich wie die Kartesische Positionsregelung mit dem Unterschied, dass das kartesische Ziel durch Gelenkkoordinaten beschrieben wird. Das bedeutet, dass hier die direkte Kinematik neu ausgewertet wird, um das kartesische Ziel zu beschreiben. In der PA-PAS Kopplung wird diese Betriebsart vorrangig eingesetzt.

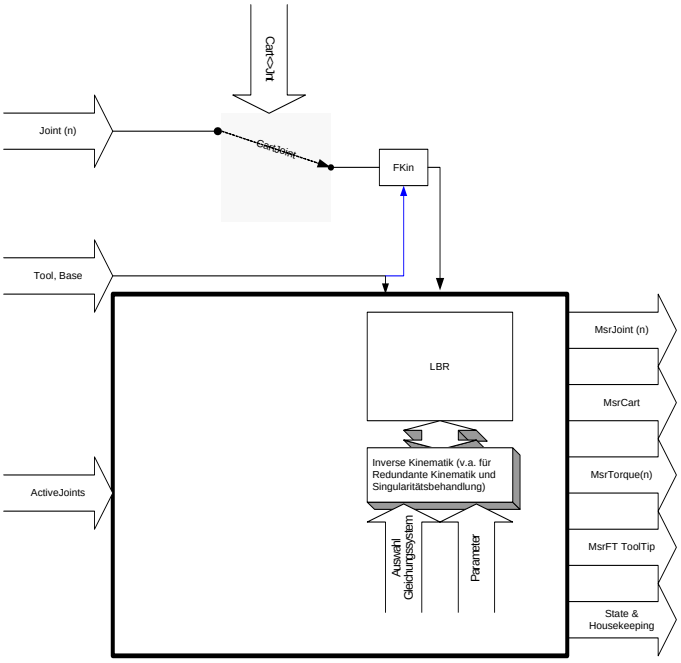


Abb. 4-10 Datenfluss Positionsregler CartJoint

Nachgiebigkeitsregelung im Gelenkraum
In dieser Betriebsart werden zusätzliche Steifigkeitsparameter (Feder – Dämpfer) pro Gelenk parametriert. Ansonsten entspricht sie der gelenkspezifischen Positionsregelung.

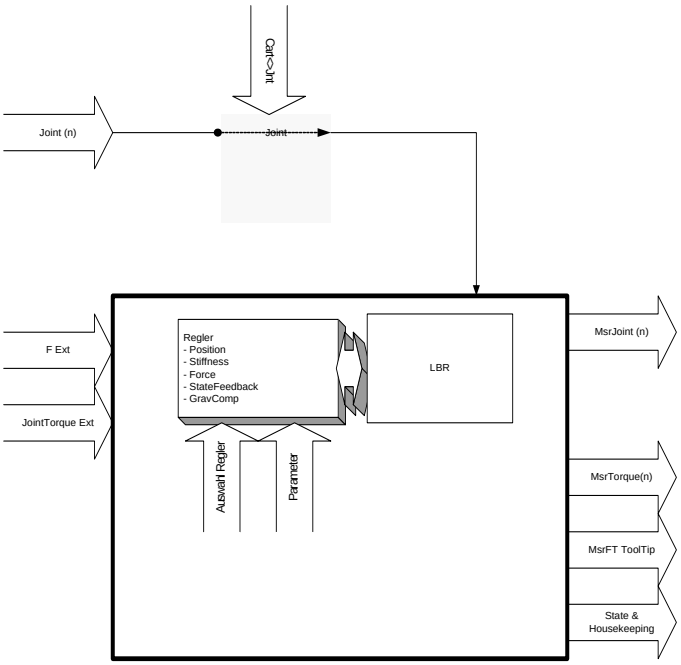


Abb. 4-11 Datenfluss Nachgiebigkeitsregler im Gelenkmodus

Nachgiebigkeitsregelung Kartesisch
In dieser Betriebsart werden zusätzlich kartesische Feder/Dämpfungs – Parameter übertragen. Da es sich um einen kinematisch redundanten Roboter handelt, muss der sogenannte Nullraum parametriert werden. D.h. für den Nullraum werden Gelenkfeder /Dämpfungsparameter benötigt.

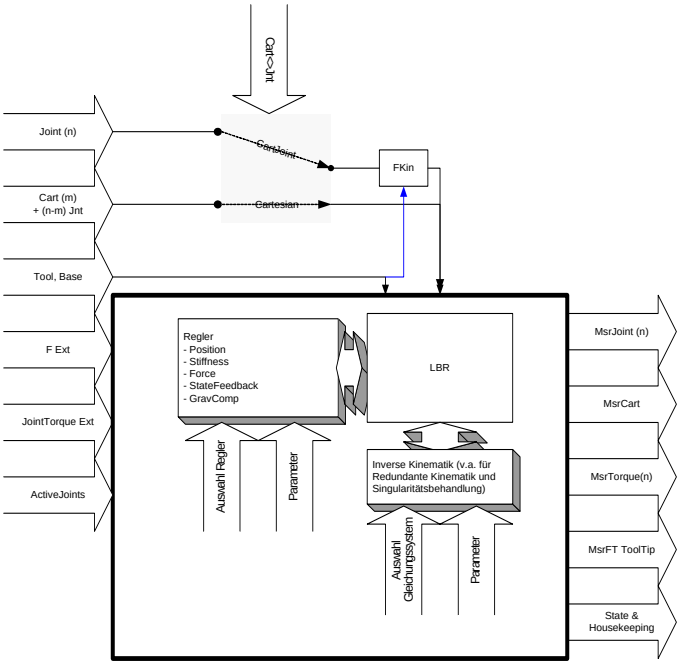


Abb. 4-12 Datenfluss Nachgiebigkeitsregler kartesisch

4.2.3 Umschaltung der Regler

Von der KRC aus können die Regelungsbetriebsarten des LBR umgeschalten werden, indem ein Schreibzugriff auf die Systemvariable $STIFFNESS ausgeführt wird. Auf KRC Seite wird ein Regler über eine Nummer ausgewählt, diese Nummer wird auf der Basissteuerung zu einem vorkonfigurierten Regler evaluiert.

4.2.4 Zusatzkommandos über die PAPAS-Schnittstelle

Zusätzlich gibt es die Möglichkeit, nutzerdefinierte Daten von der KRC zur Basissteuerung bzw. umgekehrt zu übertragen. Im Allgemeinen kann ein Text (16 Zeichen lang), 16 Float- und 16 Int- Werte übertragen.
In KRL wird das Interface wie im folgenden Beispielprogramm genutzt.

```
DEF grip( VAL:IN )
REAL VAL
int result
int md_int[16]    ;; 16 Intwerte können übertragen werden
real md_real[16]  ;; 16 Floatwerte können übertragen werden
md_real[1]=VAL    ;; Achtung Index in KRL ab 1 – in C ab 0
result=MD_CMD("PAPAS","GRIPPER",md_int[],md_real[])
END
```

4.2.5 Bedienoberfläche

Zur Bedienung des Leichtbauroboters (LBR 3) wurde die BOF (Bedienoberfläche) der KRC um einige Funktionalitäten erweitert. Diese lassen sich in folgende Teilbereiche unterscheiden:

- Plug-in : Reglerauswahl
- Systemvariable : $STIFFNESS
- 7-Achs Kinematik
- Inbetriebnahme / Konfiguration / Fehlerbehebung

Plug-In: Reglerauswahl
Um die unterschiedlichen Regelungsstrategien des LBR einzusetzen, können vorkonfigurierte Regler ausgewählt werden (siehe Abb. 4-13). Nach Anwahl des Reglers wird beim nächsten Lösen der Bremsen / Anziehen des Hauptschützes der ausgewählte Regler aktiviert.

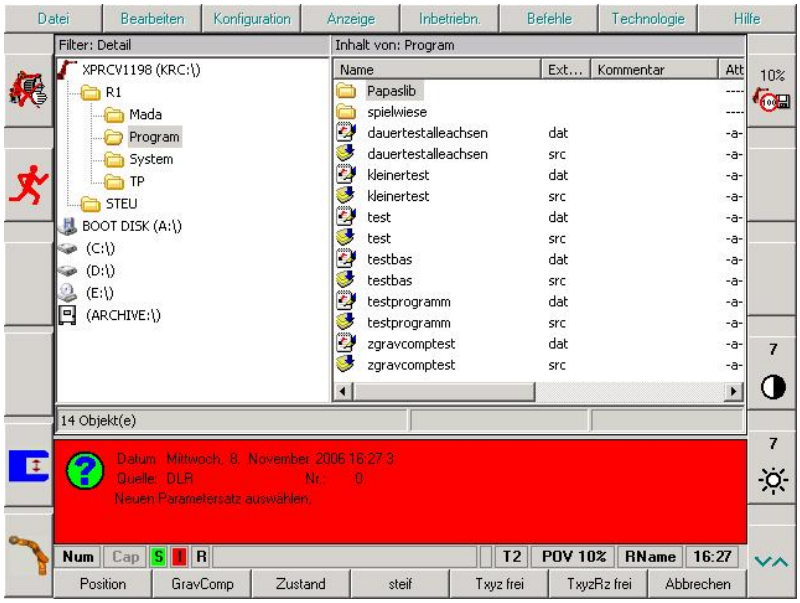


Abb. 4-13 Plug-In: Reglerauswahl

Folgende Regler können ausgewählt werden

Position:
Bei Verwendung dieses Reglers weist der Roboter keinerlei Nachgiebigkeit auf. Er verfährt rein positionsgesteuert, wie ein herkömmlicher Industrieroboter.

GravComp:
Bei Verwendung dieses Reglers ist die Traglast gravitationskompensiert, der Roboter lässt sich durch Handführung bewegen.
Anmerkung: Damit die Funktion Gravitationskompensation ordnungsgemäß funktioniert müssen die Parameter:
- *Lastdaten*
- *Ausrichtung des Roboters (Gravitationsvektor)*

korrekt eingegeben werden.
Ansonsten kann es zu unerwarteten Roboterbewegungen kommen

Zustandsregelung:
Bei Verwendung dieses Reglers weist der Roboter keinerlei Nachgiebigkeit auf. Er verfährt ähnlich positionsgesteuert wie bei dem Positionsregler.

Steifigkeitsregelung:
Bei Verwendung dieses Reglers befindet sich der Roboter in einer Nachgiebigkeitsregelung, d.h. er kann durch äußere Kräfte bewegt werden (Nachgiebigkeitsverhalten einer Feder)

Translationsbewegungen:
Bei Verwendung dieses Reglers ist die Traglast gravitationskompensiert. Der Roboter lässt sich aber nur translatorisch bewegen.

Rotationsbewegungen:
Bei Verwendung dieses Reglers ist die Traglast gravitationskompensiert. Der Roboter lässt sich aber nur rotatorisch bewegen.

Zudem lassen sich die Regler manuell konfigurieren (siehe Abb. 4-14)

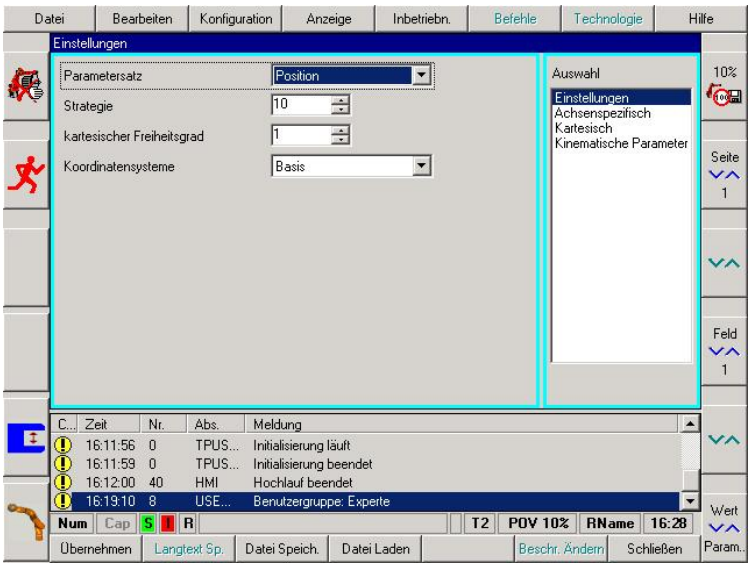


Abb. 4-14 Manuelle Reglerparametrierung

Systemvariable: $STIFFNESS
Die unterschiedlichen Regler können nicht nur interaktiv über die Bedienoberfläche konfiguriert werden, sondern lassen sich auch im Programmbetrieb in KRL nutzen. Hierzu dient die neue KRC Systemvariable *$*STIFFNESS. Die einzelnen Parameter dieser Struktur sind in Tabelle 4-2 aufgeführt.

Über den Parameter STRATEGIE lassen sich vorkonfigurierte KUKA-Regler auswählen. In der folgenden Tabelle werden die wichtigsten Regelstrategien dargestellt:

Regler Nummer (Strategie)	Interner Regler Name	Beschreibung
1	JntStiffnessT	Achsspezifischer Steifigkeitsregler
3	JntTorque	Gravitationskompensation
7	CartImpTrqNull	Kartesischer Steifigkeitsregler
9	JntSFeedback	Zustandsregler
10	JntPosition	Positionsregler

Tabelle 4-3 Vorkonfigurierte KUKA-Regler

7-Achs Kinematik
Ein besonderes Problem bestand darin, die 7-Achs-Kinematik des LBRs in die für 6 Achsen ausgelegte KUKA KRC einzubinden. Dies wurde dadurch gelöst, dass die 3. LBR Achse in der KRC als Zusatzachse E1 behandelt wird. Diese Achse lässt sich somit im Handbetrieb als auch im Programmbetrieb entsprechend ansteuern.

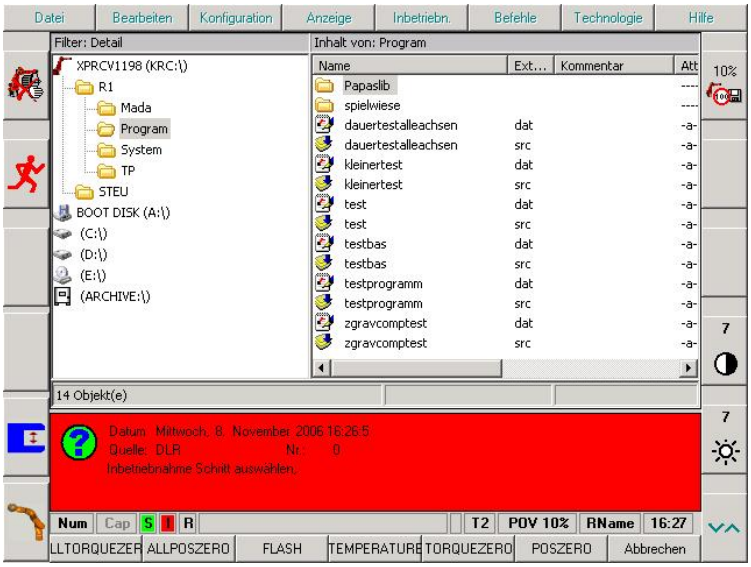


Abb. 4-15 Konfigurationsmodus

Im Konfigurationsmodus lassen sich folgende Soft-Keys anwählen:

AllTorquezero:
Zurücksetzen der Momentensensoren in allen Achsen. Wichtig für GravKomp Betrieb. Sollte nur in Schwerkraftneutraler Position (z. B. senkrecht nach oben) ausgeführt werden.

AllPosZero:
Justage aller Achsen.

Flash:
Laden der Parameter der einzelnen Achsen.

Temperatur:
Anzeige aller Temperaturdaten der sieben Achsen.

Torquezero:
Zurücksetzen der Momentensensoren für eine Achse. Wichtig für GravKomp Betrieb. Sollte nur in Schwerkraftneutraler Position (z. B. senkrecht nach oben) ausgeführt werden.

AllPosZero:
Justage einer Achse.

4.2.6 Prototyp KUKA-LBR III in Kleinserie

Aufgrund der erfolgreichen Anbindung des LBRs an die KUKA Steuerung und aufgrund des positiven Feedbacks auf der Automatica 2004, hat sich KUKA entschlossen, den LBR in einer von KUKA produzierten Variante in Kleinserie zu fertigen. Dieser Technologietransfer vom DLR zu KUKA ist ein entscheidender Schritt in Richtung Produktisierung. Dieser fand zwar zum Großteil außerhalb des PAPAS Projektes statt. Jedoch legte das PAPAS Projekt den entscheidenden Grundstein für diesen Schritt.

Dieser Roboter der neuesten Generation wurde auf der Automatica 2006 erstmals der Öffentlichkeit präsentiert.

Abb. 4-16 Prototyp der Kleinserie KUKA-LBR III

4.3 Automatica 2006 – Demonstratoren

Basierend auf den PAPAS-Entwicklungen und dem neuen KUKA-LBR III wurden für die Automatica 2006 drei Demonstratoren entwickelt, welche die neuartigen Features der Öffentlichkeit demonstrieren sollten. Alle Demonstratoren konnten dadurch realisiert werden, dass aufgrund der PAPAS-Kopplung zwischen KRC und LBR-Basissteuerung eine Programmierung der Anwendung auf KRL Ebene ermöglicht wurde. Die im Folgenden beschriebenen Anwendungen basieren somit maßgeblich auf den Ergebnissen des PAPAS-Projektes, auch wenn sie zum Teil außerhalb des PAPAS-Projektes entstanden sind.

4.3.1 Kurbler – Demonstration der Nachgiebigkeit

Zur Demonstration der kartesischen Nachgiebigkeit und deren Vorteile im Bereich Bearbeitung bzw. Montage wurde eine Zelle realisiert, in der gleich zwei Anwendungen gezeigt werden konnten:

- Buckelpiste

Der KUKA-LBR war so programmiert, dass er in der x-y-Ebene eine programmierte Trajektorie abfuhr. In z-Richtung war er kartesisch nachgiebig und folgte einer kurvigen „Buckelpiste". Um die Performance der Regelung darzustellen, konnte der Besucher interaktiv die Position der „Buckelpiste" verändern. Damit konnte gezeigt werden, wie sich der LBR in Echtzeit an neue Umgebungsbedingungen anpasst bzw. mit unscharfen Umgebungsbedingen umgehen kann (siehe Abb. 4-17).

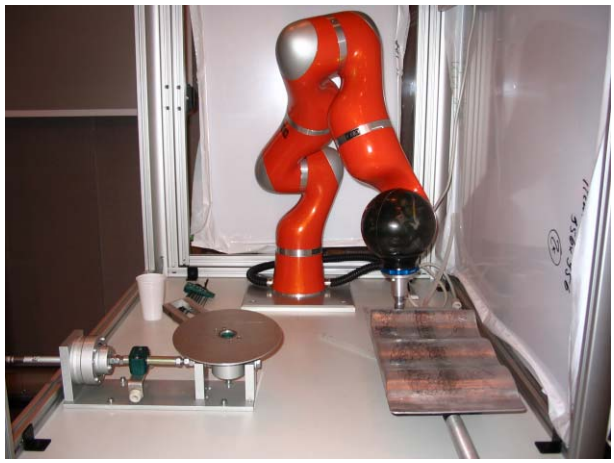


Abb. 4-17 Kurbler-LBR auf Buckelpiste

- Kurbel
In einer zweiten Anwendung hat der KUKA-LBR selbstständig das angebrachte Werkzeug (eine Art Bolzen) in ein Loch eingefügt (Peg-in-Hole). Anschließend konnte er mit Hilfe dieser Konstruktion eine Art Kurbelbewegung durchführen. Hierbei wurde er aufgrund der Zwangskopplung automatisch auf einer Kreisbahn gehalten. Um dem Besucher darzustellen, dass dies keine vorprogrammierte Kreisbahn war, konnte der Besucher wiederum interaktiv die Drehebene verändern (siehe Abb. 4-18).

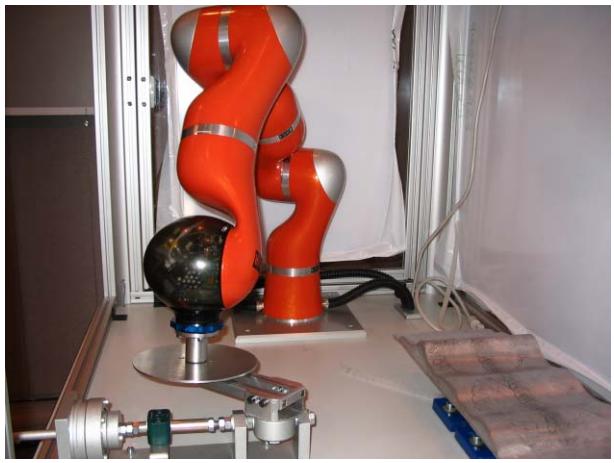


Abb. 4-18 Kurbler-LBR mit Kurbel

4.3.2 Kletterer – Demonstration der Leichtbauweise

Um eindringlich auf die neuartige Leichtbauweise und das einmalige Traglast/Eigengewicht-Verhältnis von 1:1 aufmerksam zu machen, wurde die sogenannte „Kletterer-Zelle" realisiert. Hierbei kletterte der KUKA-LBR eine Leiter hinauf und zog sich selber nach oben. Oben angelangt machte er einen „Kopfstand". Damit sollte dargestellt werden, dass der LBR in der Lage ist sein eigenes Gewicht zu tragen (siehe Abb. 4-19 und Abb. 4-20).

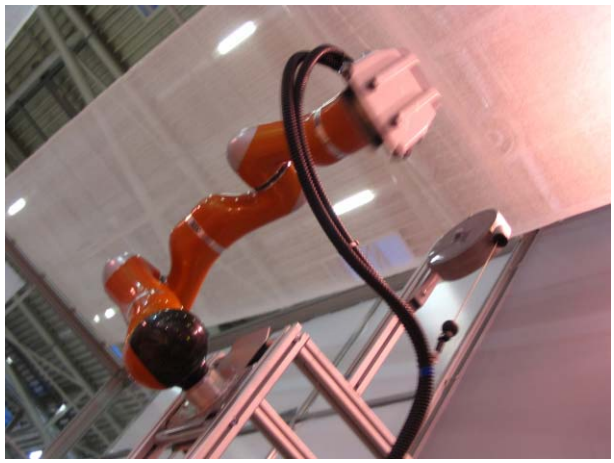


Abb. 4-19 LBR macht Kopfstand

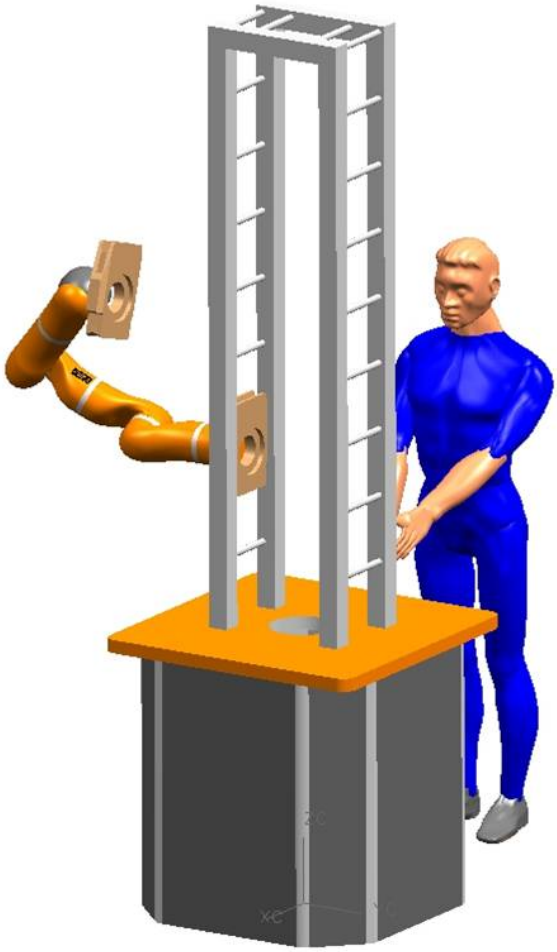


Abb. 4-20 CAD-Design: Kletterer

4.3.3 Lego-Zelle – Demonstration von „Programmieren durch Vormachen"

Eines der Highlights der Automatica 2006 war mit Sicherheit die KUKA-LBR Lego-Zelle. Hier wurde das Potential, das sich mit nachgiebigen Produktionsassistenten, die sich einfach programmieren lassen, auf hervorragende Art und Weise der Öffentlichkeit präsentiert.
Der Nutzer konnte den LBR intuitiv durch manuelles Führen bewegen. Dadurch konnte sehr schnell ein individuelles Programm (Losgröße 1) erstellt werden. Der Nutzer führte den LBR an eine von drei Zuführeinrichtungen (Rutschen) und konnte dort einen Legostein aufneh-

men. Diese Aufnahme erfolgte durch Drücken eines Bedienknopfes, welches dann den am LBR angebrachten Greifer schloss.

Anschließend konnte der Nutzer den gegriffenen Stein an einer beliebigen Stelle innerhalb eines begrenzten Gebiets ablegen. Dies erfolgte wiederum durch Drücken eines Bedienknopfes und entsprechendem Öffnen des Greifers. Durch mehrmaliges Widerholen ließen sich so ganze Lego-Konstrukte aufbauen.Hierbei „lernte" der Roboter die einzelnen Bauschritte und war in der Lage diese selbständig im Automatik-Modus zu wiederholen.

Neben dem kinderleichten „Programmieren durch Vormachen" wurde zudem gezeigt, wie eine Mensch-Roboter-Kooperation der Zukunft aussehen könnte.

Abb. 4-21 Ministerpräsident Stoiber an der Lego-Zelle

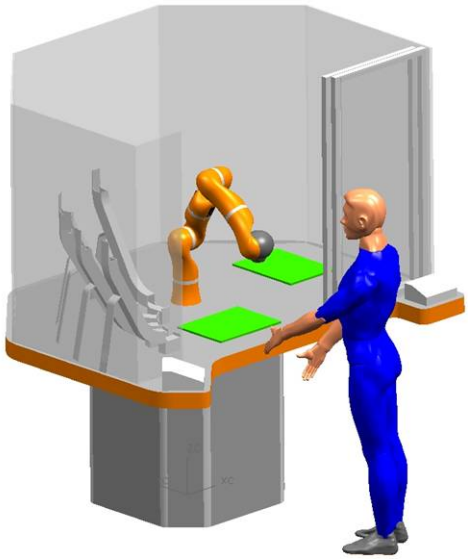


Abb. 4-22 CAD-Design: Lego-Zelle

5 Anhang: PAPAS Geräte- und Kommunikationsprofile

Anhang 1: Plug-And-Play Geräteprofil für Mehrachsensysteme (Schunk, Amatec)

Anhang 2: PAPAS-Profil für Programmierbare Fokussieroptik (Trumpf)

Anhang 3 „Plug-And-Play“ als Kommunikationsprofil/Anwendungsprofil (Lenze)

PAPAS Profil für Mehrachssensoren

(Schunk, Amatec)

Inhaltsverzeichnis

1 Motivation

Dieses Dokument beschreibt eine mögliche Schnittstelle zwischen Mehrachsensensoren, insbesondere Kraftmomentensensoren und einer übergeordneten Steuerung, z.B. Roboter auf Basis des im Förderprojekt PAPAS erarbeiteten Protokollstandards.

Beschrieben sind

- Parameterdaten
- Steuerungsdaten
- Zustandsmaschinen

Grundlage war das CANopen-Profil für „Drives and motion control (CiA DSP-402 V2.0)

Die Betrachtung wurde aus Sicht des Kraftmomentensensors der Fa. Schunk erstellt und enthält nur beschränkt Verallgemeinerungen. Weiterhin fehlen noch die Varianten zu Abfrage einzelner Achsen. Diese und andere Unzulänglichkeiten für eine offene Lösung bedürfen einer Anpassung in einem weiteren Schritt. Auch wurde nicht detailliert untersucht, inwieweit eine Anpassung auf das bestehende Profil Variablen und Objekte bereits ersetzen könnte.

2 Referenzen

Besprechungsprotokoll zum PAPAS Projekt 2004-07-08

CANopen-Profil für I/O-Module (CiA DSP-401 V1.4)

CANopen-Profil für „Drives and motion control (CiA DSP-402 V2.0)

CIA Draft Standard 301

Profilentwurf der Fa. Trumpf Laser GmbH+Co.Kg

3 Verwendete Abkürzungen

CAN	Controller Area Network. Spezifiziert in ISO11898
COB	Communication Object (CAN message). Transporteinheit im CAN Netzwerk. Daten müssen in einem COB durch das CAN Netzwerk geschickt werden.
COB-ID	COB-Identifier. Eindeutige Zuordnung des COB. Der Identifier bestimmt die Priorität des COB im Busverkehr.
CRC	Cyclic Redundancy Checksum, Prüfsumme zur Fehlererkennung
DLC	Data Length Code, Längeninformation einer CAN-Nachricht
EOF	End Of File, Kennzeichnung für das Ende der Datei (Nachricht)
FTC	Force Torque Compliance
FTI	Force Torque Interface
IDE	Identifier Extension Bit, Unterscheidung zwischen Standard- und erweitertem Format der COB-ID
OC	Object Code

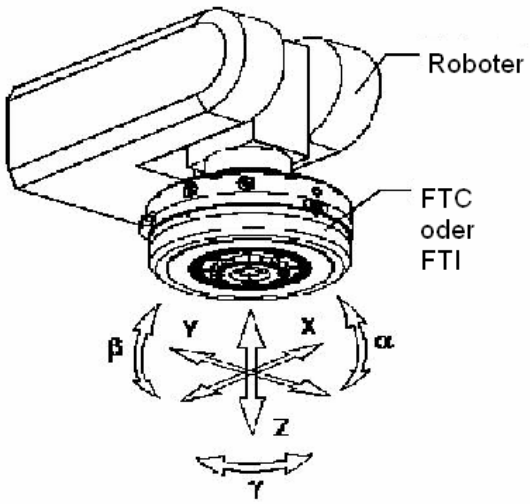
PAPAS	Plug And Play in der Antriebs- und Steuerungstechnik
PDO	Process Data Object, Objekt zum Datenaustausch zwischen verschiedenen Geräten
RF	Roboterflansch
SDO	Service Data Object. Ein Datenelement mit niedriger Priorität. Wird zur Konfigurierung des Busknotens benötigt.
TCP	Tool Center Point
WF	Werkzeugflansch

4 Prinzipielle Funktionsweise eines Mehrachsensensors

Immer höhere Anforderungen in allen Bereichen der Industrie förderten die Entwicklung immer höherentwickelter Automationssysteme. Hierbei wurden große Fortschritte in der Genauigkeit und Geschwindigkeit von Robotersystemen gemacht. Die Entwicklung von benutzerfreundlichen Sensoren wurde dabei bisher kaum berücksichtigt. Um höhere Flexibilität und Autonomie bei Robotern zu erreichen, sind

Sensoren aber unabdingbar. Bildverarbeitungssysteme werden schon heute häufig in der Qualitätskontrolle eingesetzt, Kraft-Momenten-Sensoren finden in den letzten Jahren immer häufiger ihren Einsatzbereich in der industriellen Automatisierung. Insbesondere sind solche Sensor geeignet zum

- Reagieren auf Beschleunigungskräfte.
- Ausgleichen von Positionsungenauigkeiten des Werkzeuges und/oder Werkstückes
- Arbeiten an bewegten Werkstücken
- Berücksichtigung von auftretenden Kräften und Momenten des Prozesses
- Schutz des Werkzeuges, Werkstückes und des Roboters vor Überlastung
- Fügeaufgaben
- Messung der TCP Verschiebung



Die Aufgabe des Sensors ist es also die Kräfte, Momente und Verschiebungen zwischen Werkzeugflansch und Roboterflansch zu ermitteln und bei Bedarf der Robotersteuerung zur Verfügung zu stellen. Je nach Ausführung kann es sich um

einen steifen oder einen nachgiebigen Sensor handeln. Weiterhin kann es sich auch um einen Sensor mit Beschleunigungssensoren handeln.

4.1 Kommunikationsprinzip

Die übliche Kommunikation zwischen Steuerung und Sensor erfolgt durch eine Abfrageanforderung der Steuerung an den Sensor und eine Antwort des Sensors mit den angeforderten Daten. Es kann sich hierbei um Prozessdaten oder um Parameterdaten handeln.

Weiterhin ist eine zyklische Anforderung möglich: Die Steuerung sendet einmalig eine Anforderung und der Sensor sendet als Antwort zyklisch die angeforderten Daten in den angeforderten Zeitabständen.

Die zyklische Übertragung wird durch einen neuen Steuerbefehl der Steuerung wieder beendet.

Als drittes gibt es Fehler- und Warnmeldungen, welche der Sensor ‚ungefragt' von sich aus im Bedarfsfall auf den Bus legt.

Die vierte Option ist die Möglichkeit der Steuerung Parameter des Sensors zu verändern.

Und zuletzt gibt es noch die Möglichkeit, den Sensor neu zu flashen. Diese Funktion steht dem Anwender üblicherweise nicht zur Verfügung.

Die Zuordnung der Daten zu den Achsen erfolgt wie in obiger Abbildung dargestellt. Es ist zu beachten, dass sich die Daten immer auf den Roboterflansch beziehen.

4.2 CAN Nachricht

Die generelle Kommunikation im CAN-Standard geschieht durch Frames. Ein Frame besteht aus 7 Kennfeldern:

- Start-Condition
- Message Identifier
- Steuerbits
- Daten (0-8 Bytes)
- Prüfbits
- Acknowledge-Bit
- Stop-Condition

Man unterscheidet außerdem die Frames nach der Länge des Identifiers:

- Standard Frame (11 Bit Identifier)
- Extended Frame (29 Bit Identifier)

Nach der Art des Frames unterscheidet man den

- Data Frame (Daten werden ohne spezielle Aufforderung gesendet)

- Remote Data Frame (Daten werden angefordert – Ein Empfänger, der den REMOTE identifiziert, sendet daraufhin seine Nachricht)

Den Aufbau des Frames nach Standard CAN 2.0A zeigt die folgende Tabelle:

Start	Identifier	RTR	IDE	r0	DLC	DATA	CRC	ACK	EOF+IFS
1 Bit	11 Bit	1 Bit	1 Bit	1 Bit	4 Bit	0…8 Byte	15 Bit	2 Bit	10 Bit

- Start: dominant, dient der Synchronisation,
- Identifier (COB-ID): Information für den Empfänger und Prioritätsinformation für die Busarbitrierung (kleine Werte zuerst),
- RTR: rezessiv, unterscheidet zwischen Daten- (dominant) und Datenanforderungstelegramm (rezessiv),
- IDE: Identifier Extension (Unterscheidung zwischen Standard- und erweitertem Format),
- r0: reserviert,
- DLC: enthält die Längeninformation der nachfolgenden Daten (in Byte),
- DATA: enthält die Daten des Telegramms,
- CRC: kennzeichnet den Fehlercode für die vorangegangenen Informationen. Die CRC Prüfsumme wird zur Fehlererkennung verwendet,
- ACK: enthält Rückmeldung von anderen Teilnehmern bei korrektem Empfang der Nachricht,
- EOF: kennzeichnet das Ende des Datentelegramms (7 rezessive Bits),
- IFS: kennzeichnet den Zeitraum für das Übertragen einer korrekt empfangenen Nachricht,
- SRR: ersetzt im Extended Frame das RTR Bit des Standard Frames,
- IDE: zeigt an, dass noch weitere 18 Bits folgen,
- r1, r0: reservierte Bits,
- DLC: Längeninformation der nachfolgenden Daten.

Die 11 Bit des COB-Identifiers sind wie folgt aufgebaut:

10	9	8	7	6	5	4	3	2	1	0
Funktionscode				Knotennummer						
X	X	X	X	X	X	X	X	X	X	X

X: frei wählbar

Der COB-Identifier beinhaltet die eindeutige Zuordnung des Nachrichtenobjekts. Er setzt sich zusammen aus dem Funktionscode, der die unterschiedlichen Nachrichtenarten berücksichtigt, und der Knotennummer, die jedem Sensor eindeutig zugeordnet ist. Die Knotennummer wird über eine Software eingestellt.

Die Knotennummer besteht aus sieben Bits.

Es stehen die Funktionscodes 0x2 (Binär 0010) für Lesen (receive), 0x3 (Binär 0011) für Schreiben (transmit) und 0x90 für SYNC zur Verfügung. Bei SYNC antoworten alle am Bus angeschlossenen Sensoren mit dem letzten für sie gültigen Kommando.

4.3 Transmit (Daten zum Sensor Senden)

Der Identifier setzt sich wie folgt zusammen: 0x3**XX**, wobei XX die hexadezimale Knotennummer des Sensors ist.

Beispiel:

Der Sensor mit der Knotennummer 7 soll ein Datenpaket senden (Testdaten):

Ident	DLC	Byte 0 (Kommandobyte)
0x307	1	0x46

4.4 Receive (Daten vom Sensor empfangen)

Prinzipiell können beliebig lange Datenpakete empfangen werden. Sind die angeforderten Daten länger als 8 Byte (max. CAN Nachrichtenlänge) werden diese auf mehrere Datenpakete verteilt, wobei das erste Datenbyte im jeweiligen Datenpaket die Paketnummer kennzeichnet. Eine Ausnahme stellt der erste Datensatz dar. Er enthält im ersten Datenbyte die Kommandobestätigung. Der Sensor antwortet immer auf dem Identifier 0x2**XX** wobei XX = Knoten-Nr. des Sensors (HEX !!!) gilt.

z.B.: nach obiger Datenanforderung antwortet der Sensor mit 4 Datenpaketen, die sich wie folgt zusammensetzen

Ident	DLC	Byte 0 (Kommandobyte)	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7
0x207 (von Sensor 7)	7	0x4F (Datensatz 0) (Kommando-bestätigung)	0x00 (Daten)	0x00	0xC8	0x42	0x00	0x00	0xC8
0x207	7	0x01 (Datensatz 1)	0xC2	0x51	0x06	0x9E	0x3F	0x51	0x06
0x207	7	0x02 (Datensatz 2)	0x9E	0xBF	0xE6	0x87	0x45	0x41	0x00
0x207	5	0x03 (Datensatz 3)	0x00	0x00	0x00	0x01	0x00		

4.5 Standardisierung über Profile

CANopen beschreibt die Eigenschaften von Geräten in sogenannten Geräteprofilen. Diese sind im Prinzip definierte Variablensätze. Abhängig vom Gerätetyp werden bestimmte Daten bzw. Parameter (in CANopen als Objekte bezeichnet) fest definiert.

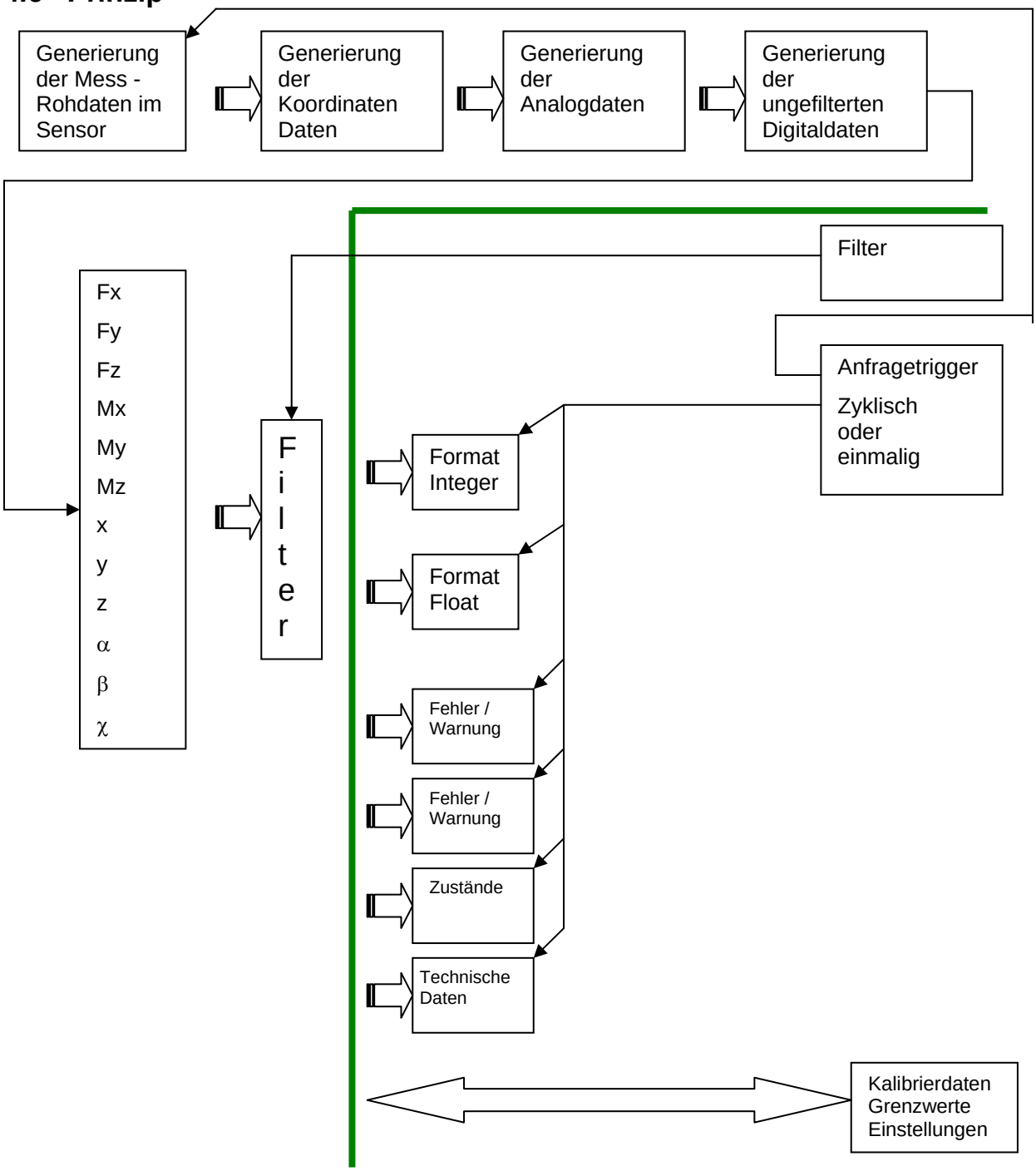
Die Geräteprofile wurden von der CiA in den Standards DS40x beschrieben.

Geräteprofile (CiA)	für die Gerätetypen
DS 401	Digitale und analoge E/A
DS 402	Antriebe
DS 403	Bedienen und Beobachten
DS 404	Sensoren / Regler
DS 405	Programmierbare Geräte
DS 406	Encoder
DS …	(weitere Geräteprofile)

Für die Systemintegration wird ein Profil generiert, um die steuerungstechnische Anbindung zu vereinfachen. Dieses Profil ist an die frei verfügbaren CANopen-Profile angelehnt. Die Standardisierung ermöglicht es, die Anbindung eines Sensors an bewegte Achsen (Roboter, Werkzeugmaschine,...) zu beschreiben und zu abstrahieren.

Als Basis dient das CANopen-Profile für generelle I/O Module (Device Profile for Generic I/O Modules).

4.6 Prinzip



5 Objektverzeichnis

Das Objektverzeichnis ist die Zusammenstellung aller Variablen und Parameter (Objekte) eines CANopen Geräts. Dabei enthalten die Daten das Prozessabbild und mit den Parametern kann das Funktionsverhalten eines CANopen Gerätes beeinflusst werden.

Ein Objektverzeichnis ist so aufgebaut, dass einige Parameter für alle Geräte dieser Kategorie zwingend vorgeschrieben sind und andere frei definiert und verwendet werden dürfen.

Die Objekte erhalten in CANopen vornehmlich eine Nummer (den sogenannten Index), mit der sie eindeutig identifiziert und auch adressiert werden können. Die Objekte können als einfache Datentypen wie z. B. Bytes, Integers, Longs oder auch Strings realisiert sein. Bei komplexeren Strukturen wie z. B. Arrays und Strukturen, wird zur Adressierung der einzelnen Elemente ein Subindex eingeführt.

Die Struktur des Objektverzeichnisses, die Vergabe der Index-Nummern sowie einige Pflichteinträge sind in den Geräteprofilen spezifiziert.

Für den Anwender ist das Objektverzeichnis als EDS-file (Electronic Data Sheet) gespeichert. Im EDS-file sind alle Objekte mit Index, Subindex, Name, Datentyp, Defaultwert, Minima, Maxima und Zugriffsmöglichkeiten (Lesen-/Schreiben, Übertragung nur per SDO oder auch per PDO usw.) gespeichert.

Mit der EDS-Datei wird somit die komplette Funktionalität eines CANopen Geräts beschrieben.

Index (hexadezimal)	**Objekt**	**Beschreibung**
0000	NC	Nicht benutzt
0001 – 001F	Static data types	Standard Datentypen wie Bool, Integer (8Bit, 16 Bit), float, string
0020 – 003F	Complex data types	Aus Standardtypen abgeleitete Datentypen (Strukturen)
0040 – 005F	Manufacturer specific datatypes	Wie „Complex data types“ jedoch spezifisch für ein bestimmtes Gerät
0060 – 0FFF	RESERVED	Reserviert für zukünftige Ereiterungen
1000 – 1FFF	Communication profile area	Für die Kommunikation über CAN spezifische Parameter (Global für alle Geräte)
2000 – 5FFF	Manufacturer specific profile area	Herstellerspezifische Profile (nicht standardisiert)
6000 – 9FFF	Standardised device profile area	Objekt für standardisierte Profile
A000 - FFFF	RESERVED	Reserviert für zukünftige Ereiterungen

Index und Subindex

Die Objekte innerhalb des Objektverzeichnisses werden über den 16-Bit Index angesprochen. Für einfache Objekte, die nur aus einer Variablen eines Datentyps bestehen, ist das ausreichend.

Bei komplexen Objekten aus mehreren Datentypen und Variablen (Strukturen, Array) wird über den Subindex auf die einzelnen Elemente zugegriffen. Das gesamte Objekt wird nach wie vor über den 16-Bit Index angesprochen.

Das Objektverzeichnis wurde nach CiA Draft Standard 404, CANopen Device Profile Measuring Devices and Closed- Loop Controllers aufgestellt. Die Aufteilung der Objektindizes wird dabei wie folgt durchgeführt:

Index	**Data Type**	**Länge**
6000_h ... $6FFF_h$	Floating point numbers (Real32)	32 Bit nach IEEE 754-1985
7000_h ... $7FFF_h$	Integer16	16 Bit
8000_h ... $8FFF_h$	Integer24	24 Bit
9000_h ... $9FFF_h$	Integer32	32 Bit

Außerdem wurde der Datentyp *Unsigned8* als 8-Bit Integer ohne Vorzeichen, der Datentyp *Unsigned16* als 16-Bit Integer ohne Vorzeichen und der Datentyp *Char* als 8-Bit Charakterwert auf die gleichen Indizes wie *Integer16* gelegt.

Um dem Benutzer einen leichten und einheitlichen Zugang zu den Funktionsblöcken zu geben, wird laut CiA Draft Standard 404 folgende Struktur genutzt:

Index	**Data Type**
$x000_h$ ... $x0FF_h$	Digital input block
$x100_h$ ... $x1FF_h$	Analogue input block
$x200_h$ ... $x2FF_h$	Digital output block
$x300_h$ ... $x3FF_h$	Analogue output block
$x400_h$ ... $x4FF_h$	Controller block
$x500_h$ ... $x5FF_h$	Alarm function block
$x600_h$ ... $xEFF_h$	reserved
$xF00_h$ ... $xFFF_h$	Device function block

Das ‚x' wird dabei durch die Ziffern 6, 7, 8 oder 9 für den Datentyp ersetzt.

5.1 Überblick über das Objektverzeichnis

5.1.1 Digital input block (Empfangs-PDOs, Steuerung sendet, Sensor empfängt)

Index	OC	Name	Datentyp	Attrib.	M/O
7036_h	VAR	Request Acceleration Float	Unsigned8	ro	O
7038_h	VAR	Request Acceleration Int	Unsigned8	ro	O
$703A_h$	VAR	Request Acceleration Translation	Unsigned8	ro	O
$703C_h$	VAR	Request Acceleration Rotation	Unsigned8	ro	O
7044_h	VAR	Request Force & Torque Float	Unsigned8	ro	O
7046_h	VAR	Request Test Data Float	Unsigned8	ro	O
7048_h	VAR	Request Force	Unsigned8	ro	O
$704A_h$	VAR	Request Torque	Unsigned8	ro	O
$704C_h$	VAR	Request Force & Torque Int	Unsigned8	ro	O
$704E_h$	VAR	Request Test Data Int	Unsigned8	ro	O
7050_h	VAR	Request Positions & Rotation Float	Unsigned8	ro	O
7052_h	VAR	Request Positions Int	Unsigned8	ro	O
7054_h	VAR	Request Rotation Int	Unsigned8	ro	O
7056_h	VAR	Request Positions & Rotation Int	Unsigned8	ro	O
7058_h	VAR	Request LimitControl Force & Torque	Unsigned8	ro	O
$705A_h$	VAR	Request LimitControl Positions & Rotation	Unsigned8	ro	O
$705C_h$	ARRAY	Write Limit in Sensor	ARRAY	wo	O
7078_h	ARRAY	Teach Limit in Sensor	ARRAY	wo	O
7064_h	VAR	Request for Position Information	Unsigned8	ro	M
7068_h	VAR	Request for Force Information	Unsigned8	ro	M
$707A_h$	VAR	Request Zero	Unsigned8	ro	O

5.1.2 Digital output block Sende-PDOs (Sensor sendet, Steuerung empfängt)

Index	OC	Name	Datentyp	Attrib.	M/O
6245_h	RECORD	Write Force & Torque Float	PDOmapping	wo	O
$624D_h$	RECORD	Write Force & Torque Int	PDOmapping	wo	O
6249_h	RECORD	Write Force Int	PDOmapping	wo	O
$624B_h$	RECORD	Write Torque Int	PDOmapping	wo	O
6259_h	RECORD	Write LimitControl Force & Torque	PDOmapping	wo	O
6251_h	RECORD	Write Positions & Rotation Float	PDOmapping	wo	O
6257_h	RECORD	Write Positions & Rotation Int	PDOmapping	wo	O
6253_h	RECORD	Write Positions Int	PDOmapping	wo	O
6255_h	RECORD	Write Rotation Int	PDOmapping	wo	O
$625B_h$	RECORD	Write LimitControl Positions & Rotation	PDOmapping	wo	O
6247_h	RECORD	Write Test Data Float	PDOmapping	wo	M
$624F_h$	RECORD	Write Test Data Int	PDOmapping	wo	M
$727B_h$	VAR	Send Zero (answer ‚OK')	Unsigned16	wo	O
6265_h	ARRAY	Write Force Information	Char	wo	M
6269_h	ARRAY	Write Position Information	Char	wo	M
6237_h	RECORD	Write Acceleration Float	PDOmapping	wo	O
6239_h	RECORD	Write Acceleration Int	PDOmapping	wo	O
$623B_h$	RECORD	Write Acceleration Translation Int	PDOmapping	wo	O
$725D_h$	RECORD	Limits Changed	Unsigned16	wo	O
7279_h	RECORD	Teached Limits Changed	Unsigned16	wo	O
$623D_h$	RECORD	Write Acceleration Rotation Int	PDOmapping	wo	O

5.1.3 Device Function block (Gerätespezifisch)

Index	OC	Name	Datentyp	Attrib.	M/O
$2F26_h$	ARRAY	Set Data Cycle	Char	ro	O
$2F27_h$	VAR	Data Cycle Changed	Unsigned16	wo	O
$2F30_h$	VAR	Request for Reset	Unsigned8	ro	M

$2F31_h$	VAR	Reset done (answer ‚OK' or ‚WP')	Unsigned16	wo	M
$2F32_h$	VAR	Start Cycle Mode	Unsigned8	ro	O
$2F34_h$	VAR	Stop Cycle Mode	Unsigned8	ro	O
$2F62_h$	VAR	Request for Sensor Information	Unsigned8	ro	M
$2F63_h$	ARRAY	Write Sensor Information	Char	wo	M
$2F6E_h$	ARRAY	Change Communication	Char	ro	M
$2F6F_h$	VAR	Communication Parameters Changed	Integer16	wo	M
$2F6A_h$	ARRAY	Set DEVNet ProducerSize	Char	ro	M
$2F6B_h$	VAR	DEVNet ProducerSize Changed	Unsigned16	wo	M
$2F6C_h$	VAR	Request Communication Information	Unsigned8	ro	M
$2F6D_h$	ARRAY	Communication Information	Char	wo	M
$2F72_h$	ARRAY	New CAN Baudrate	Char	ro	M
$2F73_h$	VAR	CAN Baudrate Changed	Unsigned16	wo	M
$2F74_h$	ARRAY	Set CAN ID	Char	ro	M
$2F75_h$	VAR	CAN ID Changed	Unsigned16	wo	M
$2F76_h$	ARRAY	New Baudrate	Char	ro	M
$2F77_h$	VAR	Baudrate Changed	Unsigned16	wo	M
$2F66_h$	VAR	Flash mode	Unsigned8	ro	M
$2F67_h$	VAR	Sensor Flashed	Unsigned16	ro	M
$2F90_h$	VAR	Sync	Unsigned8	ro	M

5.1.4 Fehler und Warnungen (Emergency-Botschaften)

Botschaften vom Typ "Emergency" werden verwendet, um Fehler eines Gerätes zu signalisieren. In dem Emergency-Telegramm wird ein Code übertragen, der den Fehler eindeutig identifiziert (definiert im Kommunikationsprofil DS-301 sowie in den jeweiligen Geräteprofilen DSP-40x).
Die folgende Tabelle zeigt einen Auszug der verfügbaren Fehlercodes. Die Emergency-Botschaft wird von jedem CANopen-Gerät selbständig gesendet. Mit der aktuellen Version 4.0 des CANopen Kommunikationsprofils kann die Aussendung einer Emergency-Botschaft auch abgeschaltet werden.

Code (hex)	Bedeutung
00xx	Kein Fehler
10xx	Nicht definierter Fehlertyp
20xx	Stromfehler
30xx	Spannungsfehler
40xx	Temperaturfehler
50xx	Fehler an der Hardware
60xx	Fehler in der Software
70xx	Zusatzmodule
80xx	Kommunikation
90xx	Externer Fehler
FFxx	Gerätespezifisch

Die Fehler sind außerdem Sensorspezifisch in drei Klassen unterteilt:

Warnungen

Es handelt sich entweder um Bedienfehler oder die Beschleunigungssensoren sind nicht installiert oder defekt.

Behebbare Fehler

Betriebsanzeige leuchtet dauerhaft ROT auf. Der Sensor stellt seinen Betrieb ein, der Fehler kann aber vom Benutzer behoben werden

Die Betriebsanzeige wechselt zwischen ROT und GRÜN. Eine der beiden Schnittstellen (seriell/CAN) arbeitet nicht ordnungsgemäß.

schwerwiegende Fehler

Die Betriebsanzeige leuchtet dauerhaft ROT auf. Der Sensor ist defekt und muss vom Kundendienst überprüft werden.

Betriebsfehler

Code	Name	Kategorie
5021_h	EEPROM Checksum1 wrong	Behebbar
5022_h	EEPROM Part A deleted	Behebbar
5025_h	Serial Defective	Behebbar
8029_h	CAN Communication error	Behebbar
2040_h	Current Error Measuring Cell X	Schwerwiegend
3041_h	Voltage Error Measuring Cell X	Schwerwiegend
5042_h	Spring Fracture	Schwerwiegend
5043_h	EEPROM Checksum2 Wrong	Schwerwiegend
5044_h	EEPROM Part B deleted	Schwerwiegend
5045_h	EEPROM Timeout	Schwerwiegend
5046_h	DSP_Error	Schwerwiegend
2047_h	Current Error horizontal	Schwerwiegend
3048_h	Voltage Error vertical	Schwerwiegend
5049_h	CAN Controller defective	Schwerwiegend
5050h	Spring-Ground Short-Circuit	Schwerwiegend

Bedienfehler

Code	Name	Kategorie
3003_h	Input Voltage too high	Behebbar
3004_h	Input Voltage too low	Behebbar
4005_h	Temperature too high	Behebbar
4006_h	Temperature too low	Behebbar
8023_h	Serial Baudrate wrong	Behebbar
8024_h	Serial Timeout	Behebbar
8026_h	CAN Baudrate wrong	Behebbar
8027_h	CAN MAC ID false	Behebbar
8028_h	CAN Timeout	Behebbar
0021_h	Unbekanntes Kommando	Warnung
0057_h	Falscher Parameter	Warnung

5.2 Detaillierte Objektbeschreibung

5.2.1 Digital input block

Objekt $7036_{h:}$ Request Acceleration Float

Objektbeschreibung

Index	7036_h
Variablenname	Request Acceleration Float
Objekt-Code	VAR
Datentyp	Unsigned8
Kategorie	Optional

Beschreibung

Zugriff	RO
Beschreibung	Anfrage für das Senden des Sensors der Beschleunigungsdaten als Floatwerte. Funktioniert dieses Kommando trotz installierter Beschleunigungssensoren nicht (Unbekanntes Kommando), ist mind. ein Beschleunigungssensor defekt.
Datentyp	Unsigned8
Länge	1
Wertebereich	36_h
Defaultwert	36_h

Objekt $7038_{h:}$ Request Acceleration Int

Objektbeschreibung

Index	7038_h
Variablenname	Request Acceleration Int
Objekt-Code	VAR
Datentyp	Unsigned8
Kategorie	Optional

Beschreibung

Zugriff	RO
Beschreibung	Anfrage für das Senden des Sensors der Beschleunigungsdaten als Integerwerte. Funktioniert dieses Kommando trotz installierter Beschleunigungssensoren nicht (Unbekanntes Kommando), ist mind. ein Beschleunigungssensor defekt.
Datentyp	Unsigned8
Länge	1
Wertebereich	38_h
Defaultwert	38_h

Objekt $703A_{h:}$ Request Acceleration Translation

Objektbeschreibung

Index	$703A_h$
Variablenname	Request Acceleration Translation
Objekt-Code	VAR
Datentyp	Unsigned8
Kategorie	Optional

Beschreibung

Zugriff	RO
Beschreibung	Anfrage für das Senden des Sensors der translatorischen Beschleunigungsdaten als Integerwerte. Funktioniert dieses Kommando trotz installierter Beschleunigungssensoren nicht (Unbekanntes Kommando), ist mind. ein Beschleunigungssensor defekt.
Datentyp	Unsigned8
Länge	1
Wertebereich	$3A_h$
Defaultwert	$3A_h$

Objekt $703C_{h:}$ Request Acceleration Rotation

Objektbeschreibung

Index	$703C_h$
Variablenname	Request Acceleration Rotation
Objekt-Code	VAR
Datentyp	Unsigned8
Kategorie	Optional

Beschreibung

Zugriff	RO
Beschreibung	Anfrage für das Senden des Sensors der rotatorischen Beschleunigungsdaten als Integerwerte. Funktioniert dieses Kommando trotz installierter Beschleunigungssensoren nicht (Unbekanntes Kommando), ist mind. ein Beschleunigungssensor defekt.
Datentyp	Unsigned8
Länge	1
Wertebereich	$3C_h$
Defaultwert	$3C_h$

Objekt $7044_{h:}$ Request Force & Torque Float

Objektbeschreibung

Index	7044_h
Variablenname	Request Force & Torque Float
Objekt-Code	VAR
Datentyp	Unsigned8
Kategorie	Optional

Beschreibung

Zugriff	RO
Beschreibung	Anfrage für das Senden des Sensors der Kraft- und Drehmomentedaten als Floatwerte.
Datentyp	Unsigned8
Länge	1
Wertebereich	44_h
Defaultwert	44_h

Objekt $7046_{h:}$ Request Test Data Float

Objektbeschreibung

Index	7046_h
Variablenname	Request Test Data Float
Objekt-Code	VAR
Datentyp	Unsigned8
Kategorie	Optional

Beschreibung

Zugriff	RO
Beschreibung	Fragt Testdaten als Float ab. Zum Testen eigener Treiber.
Datentyp	Unsigned8
Länge	1
Wertebereich	46_h
Defaultwert	46_h

Objekt $7048_{h:}$ Request Force

Objektbeschreibung

Index	7048_h
Variablenname	Request Force
Objekt-Code	VAR
Datentyp	Unsigned8
Kategorie	Optional

Beschreibung

Zugriff	RO
Beschreibung	Anfrage für das Senden des Sensors der Kraftdaten als Integerwerte.
Datentyp	Unsigned8
Länge	1
Wertebereich	48_h
Defaultwert	48_h

Objekt $704A_{h:}$ Request Torque

Objektbeschreibung

Index	$704A_h$
Variablenname	Request Torque
Objekt-Code	VAR
Datentyp	Unsigned8
Kategorie	Optional

Beschreibung

Zugriff	RO
Beschreibung	Anfrage für das Senden des Sensors der Drehmomentedaten als Integerwerte.
Datentyp	Unsigned8
Länge	1
Wertebereich	$4A_h$
Defaultwert	$4A_h$

Objekt $704C_{h:}$ Request Force & Torque Int

Objektbeschreibung

Index	$704C_h$
Variablenname	Request Force & Torque Int
Objekt-Code	VAR
Datentyp	Unsigned8
Kategorie	Optional

Beschreibung

Zugriff	RO
Beschreibung	Anfrage für das Senden des Sensors der Kraft- und Drehmomentedaten als Integerwerte.
Datentyp	Unsigned8
Länge	1
Wertebereich	$4C_h$
Defaultwert	$4C_h$

Objekt $704E_{h:}$ Request Test Data Int

Objektbeschreibung

Index	$704E_h$
Variablenname	Request Test Data Int
Objekt-Code	VAR
Datentyp	Unsigned8
Kategorie	Optional

Beschreibung

Zugriff	RO
Beschreibung	Fragt Testdaten ab. Zum Testen eigener Treiber.
Datentyp	Unsigned8
Länge	1
Wertebereich	$4E_h$
Defaultwert	$4E_h$

Objekt $7050_{h:}$ Request Positions & Rotation Float

Objektbeschreibung

Index	7050_h
Variablenname	Request Positions & Rotation Float
Objekt-Code	VAR
Datentyp	Unsigned8
Kategorie	Optional

Beschreibung

Zugriff	RO
Beschreibung	Anfrage für das Senden des Sensors der Positions- und Rotationsdaten als Floatwerte.
Datentyp	Unsigned8
Länge	1
Wertebereich	50_h
Defaultwert	50_h

Objekt $7052_{h:}$ Request Positions Int

Objektbeschreibung

Index	7052_h
Variablenname	Request Positions Int
Objekt-Code	VAR
Datentyp	Unsigned8
Kategorie	Optional

Beschreibung

Zugriff	RO
Beschreibung	Anfrage für das Senden des Sensors der Positionsdaten als Integerwerte.
Datentyp	Unsigned8
Länge	1
Wertebereich	52_h
Defaultwert	52_h

Objekt $7054_{h:}$ Request Rotation Int

Objektbeschreibung

Index	7054_h
Variablenname	Request Rotation Int
Objekt-Code	VAR
Datentyp	Unsigned8
Kategorie	Optional

Beschreibung

Zugriff	RO
Beschreibung	Anfrage für das Senden des Sensors der Rotationsdaten als Integerwerte.
Datentyp	Unsigned8
Länge	1
Wertebereich	54_h
Defaultwert	54_h

Objekt $7056_{h:}$ Request Positions & Rotation Int

Objektbeschreibung

Index	7056_h
Variablenname	Request Positions & Rotation Int
Objekt-Code	VAR
Datentyp	Unsigned8
Kategorie	Optional

Beschreibung

Zugriff	RO
Beschreibung	Anfrage für das Senden des Sensors der Positions- und Rotationsdaten als Integerwerte.
Datentyp	Unsigned8
Länge	1
Wertebereich	56_h
Defaultwert	56_h

Objekt $7058_{h:}$ Request LimitControl Force & Torque

Objektbeschreibung

Index	7058_h
Variablenname	Request LimitControl Force & Torque
Objekt-Code	VAR
Datentyp	Unsigned8
Kategorie	Optional

Beschreibung

Zugriff	RO
Beschreibung	Anfrage für das Senden des Sensors der Zustände der digitalen Grenzwerte für Kraft und Drehmomente.
Datentyp	Unsigned8
Länge	1
Wertebereich	58_h
Defaultwert	58_h

Objekt $705A_{h:}$ Request LimitControl Positions & Rotation

Objektbeschreibung

Index	$705A_h$
Variablenname	Request LimitControl Positions & Rotation
Objekt-Code	VAR
Datentyp	Unsigned8
Kategorie	Optional

Beschreibung

Zugriff	RO
Beschreibung	Anfrage für das Senden des Sensors der Zustände der digitalen Grenzwerte für Positionen und Rotation
Datentyp	Unsigned8
Länge	1
Wertebereich	$5A_h$
Defaultwert	$5A_h$

Objekt $705C_{h:}$ Write Limit in Sensor

Setzt das Limit für „digitale Positionen bzw. Kräfte“ manuell. Gesendet werden müssen 16 Char:

Bei Positionen und Rotationen (X,Y,Z,α,β,χ):

Char 1 gibt an, welcher Bereich geändert werden soll. Gültige Werte hierfür sind: ‘X’, Y’, ‘Z’, ‘A’, ‘B’, ‘C’

Char 2-7 Float Wert (Low Limit) mit 1 Vorkomma und 3 Nachkommastellen inkl. Vorzeichen => Char 4 = ‚.‘ Char 8-13 float Wert (High. Limit) mit 1 Vorkomma und 3 Nachkommastellen inkl. Vorzeichen => Char 10 = ‚.‘ Char 14-16 sind beliebig und werden ignoriert.

Beispiel:

Positionsbereich soll in Rotation B auf –0.56[°] - +1.783[°] eingestellt werden.

Schicke dem Sensor Kommando „Set Position Limit“ auf Antwort warten

0x93 (Kommando Bestätigung) + 0x07 (Anzahl der erwarteten Bytes)

Schicke dem Sensor „B-0.560+1.783xxx“ Sensor sollte nun mit „OK“ antworten => das neue Positionslimit konnte erfolgreich eingestellt werden und ist ab sofort aktiv.

Bei Kräften und Momenten (Fx,Fy,Fz,Mx,My,Mz):

Char 1 und 2 geben an, welcher Bereich geändert werden soll. Gültige Werte hierfür sind ‘Fx’, ‘Fy’, ‘Fz’, ‘Mx’, ‘My’, ‘Mz’

Char 3-9 float Wert (Low Limit) mit 3 Vorkomma und 2 Nachkommastellen inkl. Vorzeichen => Char 7 = ‚.‘ Char 10-16 float Wert (High Limit) mit 3 Vorkomma und 2 Nachkommastellen inkl. Vorzeichen => Char 14 = ‚.‘

Beispiel:

Kraftbereich soll in Fy auf –12.00[N] - +20.34[N] eingestellt werden.

- Schicke dem Sensor Kommando „Set Positiion Limit“ auf Antwort warten
- 0x93 (Kommando Bestätigung) + 0x07 (Anzahl der erwarteten Bytes)
- Schicke dem Sensor „Fy-012.00+020.34“ Sensor sollte nun mit „OK“ antworten => das neue Kraftlimit konnte erfolgreich eingestellt werden und ist ab sofort aktiv.

Objektbeschreibung

Index	$705C_h$
Variablenname	Write Limit in Sensor
Objekt-Code	ARRAY
Datentyp	Unsigned8
Kategorie	Optional

Beschreibung

Zugriff	RO
Beschreibung	Änderung eines Limitwertes
Datentyp	ARRAY
Länge	16
Wertebereich	s.o.
Defaultwert	-

Objekt 7078h: Teach Limit in Sensor

Teacht das Limit für „digitale Positionen bzw. Kräfte“. Gesendet werden müssen 2 Char:

Char 1 gibt an, welcher Bereich geteacht werden soll. Gültige Werte hierfür sind: ‘X’, ‘Y’, ‘Z’, ‘A’, ‘B’, ‘C’.

Char 2 gibt an ob das High oder Low Limit gespeichert werden soll. Gültige Werte : ‘+’, ‘-’ (High, Low).

Geteacht werden gleichzeitig Positions- und Kraftlimits. Ein getrenntes Einteachen von Positionen und Kräften ist nicht möglich. Sollen für Positionen und Kräfte verschiedene Limits eingestellt werden, muss mit "*Set Position Limit*" gearbeitet werden.

Beispiel:

Positionsbereich High soll in Rotation B bzw. Moment My auf den aktuellen Wert eingestellt werden, an dem sich der Sensor gerade befindet.

- Schicke dem Sensor Kommando " *Teach Limit in Sensor*"
- auf Antwort warten
- 0x79 (Kommando Bestätigung) + 0x02 (Anzahl der erwarteten Bytes)

- Schicke dem Sensor "B+"
- Der Sensor sollte nun mit "OK" antworten => das neue Positionslimit konnte erfolgreich eingestellt werden und ist ab sofort aktiv.

Verfügbar ab Software Version 1.10a.

Objektbeschreibung

Index	7078_h
Variablenname	Teach Limit in Sensor
Objekt-Code	ARRAY
Datentyp	Unsigned8
Kategorie	Optional

Beschreibung

Zugriff	RO
Beschreibung	Änderung eines Limitwertes
Datentyp	ARRAY
Länge	2
Wertebereich	s.o.
Defaultwert	-

Objekt $7064_{h:}$ Request for Position Information

Objektbeschreibung

Index	7064_h
Variablenname	Request for Position Information
Objekt-Code	VAR
Datentyp	Unsigned8
Kategorie	Optional

Beschreibung

Zugriff	RO
Beschreibung	Anfrage zum Senden des max. Positionsbereich des Sensors und die jeweils programmierten Schwellwerte angezeigt. X +01.500 –01.500 | +01.000 -00.916[mm] Max. Positionsbereich | eingestellter Schwellenwert
Datentyp	Unsigned8
Länge	1
Wertebereich	64_h
Defaultwert	64_h

Objekt $7068_{h:}$ Request for Force Information

Objektbeschreibung

Index	7068_h
Variablenname	Request for Force Information
Objekt-Code	VAR
Datentyp	Unsigned8
Kategorie	Optional

Beschreibung

Zugriff	RO
Beschreibung	Anfrage zum Senden des max. Kraftbereich des Sensors und die jeweils programmierten Schwellwerte angezeigt. Fx +800.00 -800.00 | +10.000 -06.916[N] Max. Kraftbereich | eingestellter Schwellenwert
Datentyp	Unsigned8
Länge	1
Wertebereich	68_h
Defaultwert	68_h

Objekt $707A_{h:}$ Request Zero

Objektbeschreibung

Index	$707A_h$
Variablenname	Request Zero
Objekt-Code	VAR
Datentyp	Unsigned8
Kategorie	Optional

Beschreibung

Zugriff	RO
Beschreibung	Befehl für den Sensor zur Nullstellung.
Datentyp	Unsigned8
Länge	1
Wertebereich	$7A_h$
Defaultwert	$7A_h$

5.2.2 Digital output block

Alle Datenfunktionen liefern im letzten Subindex ein Statuswort zurück, welches wie folgt kodiert ist:

Bit 0-7 werden gesetzt, falls an Messzelle 0-7 eine Spannung die eingestellten Spannungswerte (EEPROM) unter bzw. überschreitet.

Bit 8–15 werden gesetzt, falls an Messzelle 8-15 der Strom den eingestellten Wert (EEPROM) überschreitet. (Bei Unterschreiten des eingestellten Stromwertes liegt ein Hardwaredefekt vor).

Für den Anwender gilt: Bei Statuswort <> 0 ist der ausgegebene Messwert nicht mehr zuverlässig.

Bei „Write Force Int“, „Write Torque Int“, „Write Positions Int“, „Write Rotation Int” ist das Stauswort bei Anfragen über CAN gekürzt ((Bit 0-3 OR Bit 4-7) + (Bit 8-11 OR Bit 12-15)). Eine Zuordnung, welche Messzelle einen Fehler verursacht hat, ist somit nur bedingt möglich!

Objekt 6245_h: Write Force & Torque Float

Auslesen von 6x32 Bit Werte (Float) von Fx, Fy, Fz, Mx, My, Mz (Kräfte und Momente) und das Statuswort als Integer16. Kräfte und Momente werden in [N] bzw. [Nm] dargestellt.

Objektbeschreibung

Index	6245_h
Variablenname	Write Force & Torque Float
Objekt-Code	RECORD
Datentyp	PDOmapping
Kategorie	Optional

Beschreibung

Sub-Index	0
Zugriff	RO
Beschreibung	Kraft in x-Richtung (Fx) [N]
Kategorie	M
Datentyp	Float
Wertebereich	Unbeschränkt (Float)
Defaultwert	0

Sub-Index	1
Zugriff	RO
Beschreibung	Kraft in y-Richtung (Fy) [N]
Kategorie	M
Datentyp	Float
Wertebereich	Unbeschränkt (Float)
Defaultwert	0

Sub-Index	2
Zugriff	RO
Beschreibung	Kraft in z-Richtung (Fz) [N]
Kategorie	M
Datentyp	Float
Wertebereich	Unbeschränkt (Float)
Defaultwert	0

Sub-Index	3
Zugriff	RO
Beschreibung	Drehmoment in x-Richtung (Mx) [Nm]
Kategorie	M
Datentyp	Float
Wertebereich	Unbeschränkt (Float)
Defaultwert	0

Sub-Index	4
Zugriff	RO
Beschreibung	Drehmoment in y-Richtung (My) [Nm]
Kategorie	M
Datentyp	Float
Wertebereich	Unbeschränkt (Float)
Defaultwert	0

Sub-Index	5
Zugriff	RO
Beschreibung	Drehmoment in z-Richtung (Mz) [Nm]
Kategorie	M
Datentyp	Float
Wertebereich	Unbeschränkt (Float)
Defaultwert	0

Sub-Index	6
Zugriff	RO
Beschreibung	Angaben über den aktuellen Status
Kategorie	M
Datentyp	Unsigned16
Wertebereich	Unbeschränkt (Unsigned16)
Defaultwert	0

Objekt $624D_h$: Write Force & Torque Int

Auslesen von 6x16 Bit Werte (Int) von Fx, Fy, Fz, Mx, My, Mz (Kräfte und Momente) und das Statuswort als Integer16. Kräfte und Momente werden in [N] bzw. [Nm] dargestellt.

Umrechnung INT → [N] : /32

Umrechnung INT → [Nm] : /1024

Objektbeschreibung

Index	$624D_h$
Variablenname	Write Force & Torque Int
Objekt-Code	RECORD
Datentyp	PDOmapping
Kategorie	Optional

Beschreibung

Sub-Index	0
Zugriff	RO
Beschreibung	Kraft in x-Richtung (Fx) [N * 32]
Kategorie	M
Datentyp	Integer16
Wertebereich	Unbeschränkt (Integer16)
Defaultwert	0

Sub-Index	1
Zugriff	RO
Beschreibung	Kraft in y-Richtung (Fy) [N * 32]
Kategorie	M
Datentyp	Integer16
Wertebereich	Unbeschränkt (Integer16)
Defaultwert	0

Sub-Index	2
Zugriff	RO
Beschreibung	Kraft in z-Richtung (Fz) [N * 32]
Kategorie	M
Datentyp	Integer16
Wertebereich	Unbeschränkt (Integer16)
Defaultwert	0

Sub-Index	3
Zugriff	RO
Beschreibung	Drehmoment in x-Richtung (Mx) [Nm * 1024]
Kategorie	M
Datentyp	Integer16
Wertebereich	Unbeschränkt (Integer16)
Defaultwert	0

Sub-Index	4
Zugriff	RO
Beschreibung	Drehmoment in y-Richtung (My) [Nm * 1024]
Kategorie	M
Datentyp	Integer16
Wertebereich	Unbeschränkt (Integer16)
Defaultwert	0

Sub-Index	5
Zugriff	RO
Beschreibung	Drehmoment in z-Richtung (Mz) [Nm * 1024]
Kategorie	M
Datentyp	Integer16
Wertebereich	Unbeschränkt (Integer16)
Defaultwert	0

Sub-Index	6
Zugriff	RO
Beschreibung	Angaben über den aktuellen Status
Kategorie	M
Datentyp	Unsigned16
Wertebereich	Unbeschränkt (Unsigned16)
Defaultwert	0

Objekt 6249_h: Write Force Int

Auslesen von 3x16 Bit Werte (Int) von Fx, Fy, Fz (Kräfte) und das Statuswort als Integer16. Kräfte werden in [N] dargestellt.

Umrechnung INT → [N] : /32

Objektbeschreibung

Index	6249_h
Variablenname	Write Force Int
Objekt-Code	RECORD
Datentyp	PDOmapping
Kategorie	Optional

Beschreibung

Sub-Index	0
Zugriff	RO
Beschreibung	Kraft in x-Richtung (Fx) [N * 32]
Kategorie	M
Datentyp	Integer16
Wertebereich	Unbeschränkt (Integer16)
Defaultwert	0

Sub-Index	1
Zugriff	RO
Beschreibung	Kraft in y-Richtung (Fy) [N * 32]
Kategorie	M
Datentyp	Integer16
Wertebereich	Unbeschränkt (Integer16)
Defaultwert	0

Sub-Index	2
Zugriff	RO
Beschreibung	Kraft in z-Richtung (Fz) [N * 32]
Kategorie	M
Datentyp	Integer16
Wertebereich	Unbeschränkt (Integer16)
Defaultwert	0

Sub-Index	3
Zugriff	RO
Beschreibung	Angaben über den aktuellen Status
Kategorie	M
Datentyp	Unsigned16
Wertebereich	Unbeschränkt (Unsigned16)
Defaultwert	0

Objekt $624B_h$: Write Torque Int

Auslesen von 3x16 Bit Werte (Int) von Mx, My, Mz (Momente) und das Statuswort als Integer16. Momente werden in [Nm] dargestellt.

Umrechnung INT → [Nm] : /1024

Objektbeschreibung

Index	$624B_h$
Variablenname	Write Torque Int
Objekt-Code	RECORD
Datentyp	PDOmapping
Kategorie	Optional

Beschreibung

Sub-Index	0
Zugriff	RO
Beschreibung	Drehmoment in x-Richtung (Mx) [Nm * 1024]
Kategorie	M
Datentyp	Integer16
Wertebereich	Unbeschränkt (Integer16)
Defaultwert	0

Sub-Index	1
Zugriff	RO
Beschreibung	Drehmoment in y-Richtung (My) [Nm * 1024]
Kategorie	M
Datentyp	Integer16
Wertebereich	Unbeschränkt (Integer16)
Defaultwert	0

Sub-Index	2
Zugriff	RO
Beschreibung	Drehmoment in z-Richtung (Mz) [Nm * 1024]
Kategorie	M
Datentyp	Integer16
Wertebereich	Unbeschränkt (Integer16)
Defaultwert	0

Sub-Index	3
Zugriff	RO
Beschreibung	Angaben über den aktuellen Status
Kategorie	M
Datentyp	Unsigned16
Wertebereich	Unbeschränkt (Unsigned16)
Defaultwert	0

Objekt 6259_h: Write LimitControl Force & Torque

Gibt an, welche Limits bei den Kraft- und Momentenwerten überschritten wurden.

1x16 Bit Wert (Integer16). Bits 0-5 Kraftbereich wurde überschritten. Bits 8 – 13 Kraftbereich wurde unterschritten.

Reihenfolge:
Bit 0 / 8 = Fx
Bit 1 / 9 = Fy
Bit 2 / 10 = Fz
Bit 3 / 11 = Mx
Bit 4 / 12 = My
Bit 5 / 13 = Mz
Bit 6 / 14 reserved
Bit 7 / 15 reserved

Objektbeschreibung

Index	6259_h
Variablenname	Write LimitControl Force & Torque
Objekt-Code	RECORD
Datentyp	PDOmapping
Kategorie	Optional

Beschreibung

Sub-Index	0
Zugriff	RO
Beschreibung	Integer zur Abfrage der Einhaltung der Limits (s.o.)
Kategorie	M
Datentyp	Integer16
Wertebereich	Unbeschränkt (Integer16)
Defaultwert	0

Sub-Index	1
Zugriff	RO
Beschreibung	Angaben über den aktuellen Status
Kategorie	M
Datentyp	Unsigned16
Wertebereich	Unbeschränkt (Unsigned16)
Defaultwert	0

Objekt 6251_h: Write Positions & Rotation Float

Enthält 6x32 Bit Werte (Float) nach IEEE754 für die aktuelle Position und die Rotation (X,Y,Z,α,β,χ).

Positionen und Rotationen werden in Einheiten [mm] bzw. [°] dargestellt.

Objektbeschreibung

Index	6251_h
Variablenname	Write Positions & Rotation Float
Objekt-Code	RECORD
Datentyp	PDOmapping
Kategorie	Optional

Beschreibung

Sub-Index	0
Zugriff	RO
Beschreibung	x-Position [mm]
Kategorie	M
Datentyp	Float
Wertebereich	Unbeschränkt (Float)
Defaultwert	0

Sub-Index	1
Zugriff	RO
Beschreibung	y-Position [mm]
Kategorie	M
Datentyp	Float
Wertebereich	Unbeschränkt (Float)
Defaultwert	0

Sub-Index	2
Zugriff	RO
Beschreibung	z-Position [mm]
Kategorie	M
Datentyp	Float
Wertebereich	Unbeschränkt (Float)
Defaultwert	0

Sub-Index	3
Zugriff	RO
Beschreibung	Rotation um x [°]
Kategorie	M
Datentyp	Float
Wertebereich	Unbeschränkt (Float)
Defaultwert	0

Sub-Index	4
Zugriff	RO
Beschreibung	Rotation um y [°]
Kategorie	M
Datentyp	Float
Wertebereich	Unbeschränkt (Float)
Defaultwert	0

Sub-Index	5
Zugriff	RO
Beschreibung	Rotation um z [°]
Kategorie	M
Datentyp	Float
Wertebereich	Unbeschränkt (Float)
Defaultwert	0

Sub-Index	6
Zugriff	RO
Beschreibung	Angaben über den aktuellen Status
Kategorie	M
Datentyp	Unsigned16
Wertebereich	Unbeschränkt (Unsigned16)
Defaultwert	0

Objekt 6257_h: Write Positions & Rotation Int

Enthält 6x16 Bit Werte (Integer16) für die aktuelle Position und die Rotation (X,Y,Z, ,α,β,χ).

Positionen und Rotationen werden in Einheiten [mm] bzw. [°] dargestellt.

Umrechnung INT → [mm] : /4096

Umrechnung INT → [°] : $\cdot \frac{180}{\pi \cdot 65536}$

Objektbeschreibung

Index	6257_h
Variablenname	Write Positions & Rotation Int
Objekt-Code	RECORD
Datentyp	PDOmapping
Kategorie	Optional

Beschreibung

Sub-Index	0
Zugriff	RO
Beschreibung	x-Position [mm*4096]
Kategorie	M
Datentyp	Integer16
Wertebereich	Unbeschränkt (Integer16)
Defaultwert	0

Sub-Index	1
Zugriff	RO
Beschreibung	y-Position [mm*4096]
Kategorie	M
Datentyp	Integer16
Wertebereich	Unbeschränkt (Integer16)
Defaultwert	0

Sub-Index	2
Zugriff	RO
Beschreibung	z-Position [mm*4096]
Kategorie	M
Datentyp	Integer16
Wertebereich	Unbeschränkt (Integer16)
Defaultwert	0

Sub-Index	3
Zugriff	RO
Beschreibung	Rotation um x [$° \cdot \frac{\pi \cdot 65536}{180}$]
Kategorie	M
Datentyp	Integer16
Wertebereich	Unbeschränkt (Integer16)
Defaultwert	0

Sub-Index	4
Zugriff	RO
Beschreibung	Rotation um y [$° \cdot \frac{\pi \cdot 65536}{180}$]
Kategorie	M
Datentyp	Integer16
Wertebereich	Unbeschränkt (Integer16)
Defaultwert	0

Sub-Index	5
Zugriff	RO
Beschreibung	Rotation um z [$° \cdot \frac{\pi \cdot 65536}{180}$]
Kategorie	M
Datentyp	Integer16
Wertebereich	Unbeschränkt (Integer16)
Defaultwert	0

Sub-Index	6
Zugriff	RO
Beschreibung	Angaben über den aktuellen Status
Kategorie	M
Datentyp	Unsigned16
Wertebereich	Unbeschränkt (Unsigned16)
Defaultwert	0

Objekt 6253_h: Write Positions Int

Enthält 3x16 Bit Werte (Integer16) für die aktuelle Position (X,Y,Z).

Positionen werden in Einheiten [mm].

Umrechnung INT → [mm] : /4096

Objektbeschreibung

Index	6253_h
Variablenname	Write Positions Int
Objekt-Code	RECORD
Datentyp	PDOmapping
Kategorie	Optional

Beschreibung

Sub-Index	0
Zugriff	RO
Beschreibung	x-Position [mm*4096]
Kategorie	M
Datentyp	Integer16
Wertebereich	Unbeschränkt (Integer16)
Defaultwert	0

Sub-Index	1
Zugriff	RO
Beschreibung	y-Position [mm*4096]
Kategorie	M
Datentyp	Integer16
Wertebereich	Unbeschränkt (Integer16)
Defaultwert	0

Sub-Index	2
Zugriff	RO
Beschreibung	z-Position [mm*4096]
Kategorie	M
Datentyp	Integer16
Wertebereich	Unbeschränkt (Integer16)
Defaultwert	0

Sub-Index	3
Zugriff	RO
Beschreibung	Angaben über den aktuellen Status
Kategorie	M
Datentyp	Unsigned16
Wertebereich	Unbeschränkt (Unsigned16)

Objekt 6255_h: Write Rotation Int

Enthält 3x16 Bit Werte (Integer16) für die aktuelle Rotation (,α,β,χ).

Rotationen werden in Einheiten [°] dargestellt.

Umrechnung INT → [°] : $\cdot \frac{180}{\pi \cdot 65536}$

Objektbeschreibung

Index	6255_h
Variablenname	Write Rotation Int
Objekt-Code	RECORD
Datentyp	PDOmapping
Kategorie	Optional

Beschreibung

Sub-Index	0
Zugriff	RO
Beschreibung	Rotation um x [° $\cdot \frac{\pi \cdot 65536}{180}$]
Kategorie	M
Datentyp	Integer16
Wertebereich	Unbeschränkt (Integer16)
Defaultwert	0

Sub-Index	1
Zugriff	RO
Beschreibung	Rotation um y [° $\cdot \frac{\pi \cdot 65536}{180}$]
Kategorie	M
Datentyp	Integer16
Wertebereich	Unbeschränkt (Integer16)
Defaultwert	0

Sub-Index	2
Zugriff	RO
Beschreibung	Rotation um z [$° \cdot \frac{\pi \cdot 65536}{180}$]
Kategorie	M
Datentyp	Integer16
Wertebereich	Unbeschränkt (Integer16)
Defaultwert	0

Sub-Index	3
Zugriff	RO
Beschreibung	Angaben über den aktuellen Status
Kategorie	M
Datentyp	Unsigned16
Wertebereich	Unbeschränkt (Unsigned16)

Objekt $625B_{h:}$ Write LimitControl Positions & Rotation

Gibt an, welche Limits bei den Positions- und Rotationswerten überschritten wurden.

1x16 Bit Wert (INT16) Bits 0-5 Positionsbereich wurde überschritten Bits 8 – 13 Positionsbereich wurde unterschritten.

Reihenfolge:
Bit 0 / 8 = X
Bit 1 / 9 = Y
Bit 2 / 10 = Z
Bit 3 / 11 = α
Bit 4 / 12 = β
Bit 5 / 13 = χ
Bit 6 / 14 reserved
Bit 7 / 15 reserved

Objektbeschreibung

Index	$625B_h$
Variablenname	Write LimitControl Positions & Rotation
Objekt-Code	RECORD
Datentyp	PDOmapping
Kategorie	Optional

Beschreibung

Sub-Index	0
Zugriff	RO
Beschreibung	Integer zur Abfrage der Einhaltung der Limits (s.o.)
Kategorie	M
Datentyp	Integer16
Wertebereich	Unbeschränkt (Integer16)
Defaultwert	0

Sub-Index	1
Zugriff	RO
Beschreibung	Angaben über den aktuellen Status
Kategorie	M
Datentyp	Unsigned16
Wertebereich	Unbeschränkt (Unsigned16)
Defaultwert	0

Objekt 6247_h: Write Test Data Float

Enthält 6x32 Bit (FLOAT) Werte nach IEEE754: 100.0 / -100.0 / 1.235678 / -1.2345678 / 12.345678 / 0.0. Das Objekt dient zum Testen eigener Treiber.

1x16 Bit Statuswort = 0x0001.

Objektbeschreibung

Index	6247_h
Variablenname	Write Test Data Float
Objekt-Code	RECORD
Datentyp	PDOmapping
Kategorie	Optional

Beschreibung

Sub-Index	0
Zugriff	RO
Beschreibung	Wert 100.0
Kategorie	M
Datentyp	Float
Wertebereich	100.0
Defaultwert	100.0

Sub-Index	1
Zugriff	RO
Beschreibung	Wert -100.0
Kategorie	M
Datentyp	Float
Wertebereich	-100.0
Defaultwert	-100.0

Sub-Index	2
Zugriff	RO
Beschreibung	Wert 1.235678
Kategorie	M
Datentyp	Float
Wertebereich	1.235678
Defaultwert	1.235678

Sub-Index	3
Zugriff	RO
Beschreibung	Wert -1.235678
Kategorie	M
Datentyp	Float
Wertebereich	-1.235678
Defaultwert	-1.235678

Sub-Index	4
Zugriff	RO
Beschreibung	Wert 12.345678
Kategorie	M
Datentyp	Float
Wertebereich	12.345678
Defaultwert	12.345678

Sub-Index	5
Zugriff	RO
Beschreibung	Wert 0.0
Kategorie	M
Datentyp	Float
Wertebereich	0.0
Defaultwert	0.0

Sub-Index	6
Zugriff	RO
Beschreibung	Angaben über den aktuellen Status
Kategorie	M
Datentyp	Unsigned16
Wertebereich	0x0001
Defaultwert	0x0001

Objekt $624F_{h:}$ Write Test Data Int

Enthält 6x16 Bit (Interger16) Werte: -1 / 0 / -1024 / 255 / 511 / 256. Das Objekt dient zum Testen eigener Treiber.

1x16 Bit Statuswort = 0xFE00.

Objektbeschreibung

Index	$624F_h$
Variablenname	Write Test Data Int
Objekt-Code	RECORD
Datentyp	PDOmapping
Kategorie	Optional

Beschreibung

Sub-Index	0
Zugriff	RO
Beschreibung	Wert -1
Kategorie	M
Datentyp	Integer16
Wertebereich	-1
Defaultwert	-1

Sub-Index	1
Zugriff	RO
Beschreibung	Wert 0
Kategorie	M
Datentyp	Integer16
Wertebereich	0
Defaultwert	0

Sub-Index	2
Zugriff	RO
Beschreibung	Wert -1024
Kategorie	M
Datentyp	Integer16
Wertebereich	-1024
Defaultwert	-1024

Sub-Index	3
Zugriff	RO
Beschreibung	Wert 255
Kategorie	M
Datentyp	Integer16
Wertebereich	255
Defaultwert	255

Sub-Index	4
Zugriff	RO
Beschreibung	Wert 511
Kategorie	M
Datentyp	Integer16
Wertebereich	511
Defaultwert	511

Sub-Index	5
Zugriff	RO
Beschreibung	Wert 256
Kategorie	M
Datentyp	Integer16
Wertebereich	256
Defaultwert	256

Sub-Index	6
Zugriff	RO
Beschreibung	Angaben über den aktuellen Status
Kategorie	M
Datentyp	Unsigned16
Wertebereich	0xFE00
Defaultwert	0xFE00

Objekt $727B_{h:}$ Send Zero

Antwort für die Nullsetzung des Sensors. Die Antwort ist immer „OK“ (0x4B4F).

Objektbeschreibung

Index	$727B_h$
Variablenname	Send Zero
Objekt-Code	RECORD
Datentyp	PDOmapping
Kategorie	Optional

Beschreibung

Sub-Index	0
Zugriff	RO
Beschreibung	„OK“ (0x4B4F)
Kategorie	M
Datentyp	Integer16
Wertebereich	0x4B4F
Defaultwert	0x4B4F

Sub-Index	1
Zugriff	RO
Beschreibung	Angaben über den aktuellen Status
Kategorie	M
Datentyp	Unsigned16
Wertebereich	Unbeschränkt (Unsigned16)
Defaultwert	0

Objekt 6265_h: Write Force Information

Hiermit wird der max. Kraftbereich des Sensors abgefragt und die jeweils programmierten Schwellwerte angezeigt. Die Antwort erscheint im Klartext (Bytes = ASCII Code).

Fx +800.00 -800.00 | +10.000 -06.916[N]

Max. Kraftbereich | eingestellter Schwellwert

Objektbeschreibung

Index	6265_h
Variablenname	Write Force Information
Objekt-Code	ARRAY
Datentyp	Char
Kategorie	Optional

Beschreibung

Sub-Index	0
Zugriff	RO
Beschreibung	Anzahl der folgenden Bytes
Kategorie	M
Datentyp	Integer16
Wertebereich	Unbeschränkt (Unsigned16)
Defaultwert	0

Sub-Index	1
Zugriff	RO
Beschreibung	Kraftbereich des Sensors und die jeweils programmierten Schwellwerte
Kategorie	M
Datentyp	Char
Wertebereich	Char
Defaultwert	Empty

Objekt 6269_h: Write Position Information

Hiermit wird der max. Positionsbereich des Sensors abgefragt und die jeweils programmierten Schwellwerte angezeigt. Die Antwort erscheint im Klartext (Bytes = ASCII Code).

X +01.500 –01.500 | +01.000 -00.916[mm]

Max. Positionsbereich | eingestellter Schwellenwert

Objektbeschreibung

Index	6269_h
Variablenname	Write Position Information
Objekt-Code	ARRAY
Datentyp	Char
Kategorie	Optional

Beschreibung

Sub-Index	0
Zugriff	RO
Beschreibung	Anzahl der folgenden Bytes
Kategorie	M
Datentyp	Integer16
Wertebereich	Unbeschränkt (Unsigned16)
Defaultwert	0

Sub-Index	1
Zugriff	RO
Beschreibung	Positionsbereich des Sensors und die jeweils programmierten Schwellwerte
Kategorie	M
Datentyp	Char
Wertebereich	Char
Defaultwert	Empty

Objekt 6237_h: Write Acceleration Float

Enthält die Beschleunigungswerte (6x32 Bit (Float) Werte nach IEEE754 AccX, AccY, AccZ, Accα, Accβ, Accχ in [m/s²] bzw. [°/s²]). Funktioniert ab Firmware 2.00a und installierten Messzellen mit Beschleunigungssensoren (optional).

Funktioniert diese Kommando trotz installierter Beschleunigungssensoren nicht (Unbekanntes Kommando), ist mind. ein Beschleunigungssensor defekt.

1x16 Bit Status: unter 8Bit = Messbereichsüberschreitung

Objektbeschreibung

Index	6237_h
Variablenname	Write Acceleration Float
Objekt-Code	RECORD
Datentyp	PDOmapping
Kategorie	Optional

Beschreibung

Sub-Index	0
Zugriff	RO
Beschreibung	Beschleunigung in x-Richtung [m/s²]
Kategorie	M
Datentyp	Float
Wertebereich	Unbeschränkt (Float)
Defaultwert	0

Sub-Index	1
Zugriff	RO
Beschreibung	Beschleunigung in y-Richtung [m/s²]
Kategorie	M
Datentyp	Float
Wertebereich	Unbeschränkt (Float)
Defaultwert	0

Sub-Index	2
Zugriff	RO
Beschreibung	Beschleunigung in z-Richtung [m/s²]
Kategorie	M
Datentyp	Float
Wertebereich	Unbeschränkt (Float)
Defaultwert	0

Sub-Index	3
Zugriff	RO
Beschreibung	Rotationsbeschleunigung bei Rotation um x [°/s²]
Kategorie	M
Datentyp	Float
Wertebereich	Unbeschränkt (Float)
Defaultwert	0

Sub-Index	4
Zugriff	RO
Beschreibung	Rotationsbeschleunigung bei Rotation um y [°/s²]
Kategorie	M
Datentyp	Float
Wertebereich	Unbeschränkt (Float)
Defaultwert	0

Sub-Index	5
Zugriff	RO
Beschreibung	Rotationsbeschleunigung bei Rotation um z [°/s²]
Kategorie	M
Datentyp	Float
Wertebereich	Unbeschränkt (Float)
Defaultwert	0

Sub-Index	6
Zugriff	RO
Beschreibung	Angaben über den aktuellen Status
Kategorie	M
Datentyp	Unsigned16
Wertebereich	Unbeschränkt (Unsigned16)
Defaultwert	0

Objekt 6239_h: Write Acceleration Int

Enthält die Beschleunigungswerte als Integer16 (6x16 Bit (Integer16) Werte AccX, AccY, AccZ, Accα, Accβ, Accχ in [m/s²] bzw. [°/s²]). Funktioniert ab Firmware 2.00a und installierten Messzellen mit Beschleunigungssensoren (optional).

Funktioniert diese Kommando trotz installierter Beschleunigungssensoren nicht (Unbekanntes Kommando), ist mind. ein Beschleunigungssensor defekt.

1x16 Bit Status: unter 8Bit = Messbereichsüberschreitung

Objektbeschreibung

Index	6239_h
Variablenname	Write Acceleration Int
Objekt-Code	RECORD
Datentyp	PDOmapping
Kategorie	Optional

Beschreibung

Sub-Index	0
Zugriff	RO
Beschreibung	Beschleunigung in x-Richtung [m/s²]
Kategorie	M
Datentyp	Integer16
Wertebereich	Unbeschränkt (Integer16)
Defaultwert	0

Sub-Index	1
Zugriff	RO
Beschreibung	Beschleunigung in y-Richtung [m/s²]
Kategorie	M
Datentyp	Integer16
Wertebereich	Unbeschränkt (Integer16)
Defaultwert	0

Sub-Index	2
Zugriff	RO
Beschreibung	Beschleunigung in z-Richtung [m/s²]
Kategorie	M
Datentyp	Integer16
Wertebereich	Unbeschränkt (Integer16)
Defaultwert	0

Sub-Index	3
Zugriff	RO
Beschreibung	Rotationsbeschleunigung bei Rotation um x [°/s²]
Kategorie	M
Datentyp	Integer16
Wertebereich	Unbeschränkt (Integer16)
Defaultwert	0

Sub-Index	4
Zugriff	RO
Beschreibung	Rotationsbeschleunigung bei Rotation um y [°/s²]
Kategorie	M
Datentyp	Integer16
Wertebereich	Unbeschränkt (Integer16)
Defaultwert	0

Sub-Index	5
Zugriff	RO
Beschreibung	Rotationsbeschleunigung bei Rotation um z [°/s²]
Kategorie	M
Datentyp	Integer16
Wertebereich	Unbeschränkt (Integer16)
Defaultwert	0

Sub-Index	6
Zugriff	RO
Beschreibung	Angaben über den aktuellen Status
Kategorie	M
Datentyp	Unsigned16
Wertebereich	Unbeschränkt (Unsigned16)
Defaultwert	0

Objekt $623B_h$: Write Acceleration Translation Int

Enthält die Beschleunigungswerte für die Translation als Integer16 (3x16 Bit (Integer16) Werte AccX, AccY, AccZ in [m/s²]). Funktioniert ab Firmware 2.00a und installierten Messzellen mit Beschleunigungssensoren (optional).

Funktioniert diese Kommando trotz installierter Beschleunigungssensoren nicht (Unbekanntes Kommando), ist mind. ein Beschleunigungssensor defekt.

1x8 Bit Status: unter 8Bit = Messbereichsüberschreitung

Die letzten 16 Bit sind auf 8 Bit verkürzt, damit alle Daten in nur einer CAN Message Platz finden.

Objektbeschreibung

Index	$623B_h$
Variablenname	Write Acceleration Translation Int
Objekt-Code	RECORD
Datentyp	PDOmapping
Kategorie	Optional

Beschreibung

Sub-Index	0
Zugriff	RO
Beschreibung	Beschleunigung in x-Richtung [m/s²]
Kategorie	M
Datentyp	Integer16
Wertebereich	Unbeschränkt (Integer16)
Defaultwert	0

Sub-Index	1
Zugriff	RO
Beschreibung	Beschleunigung in y-Richtung [m/s²]
Kategorie	M
Datentyp	Integer16
Wertebereich	Unbeschränkt (Integer16)
Defaultwert	0

Sub-Index	2
Zugriff	RO
Beschreibung	Beschleunigung in z-Richtung [m/s²]
Kategorie	M
Datentyp	Integer16
Wertebereich	Unbeschränkt (Integer16)
Defaultwert	0

Sub-Index	3
Zugriff	RO
Beschreibung	Angaben über den aktuellen Status
Kategorie	M
Datentyp	Unsigned8
Wertebereich	Unbeschränkt (Unsigned8)
Defaultwert	0

Objekt $725D_h$: Limits Changed

Die Antwort auf eine Änderung der Limits des Sensors. Die Antwort ist „OK“ (0x4B4F) bei erfolgreicher Ausführung, ansonsten „WP“ (0x5057).

Objektbeschreibung

Index	$725D_h$
Variablenname	Limits Changed
Objekt-Code	VAR
Datentyp	Char
Kategorie	Optional

Beschreibung

Sub-Index	0
Zugriff	RO
Beschreibung	Antwort
Kategorie	M
Datentyp	Integer16
Wertebereich	0x4B4F (‚OK’), 0x5057 (‚WP’)
Defaultwert	0x5057

Objekt 7279_h: Teached Limits Changed

Die Antwort auf eine Änderung der Limits des Sensors. Die Antwort ist „OK“ (0x4B4F) bei erfolgreicher Ausführung, ansonsten „WP“ (0x5057).

Objektbeschreibung

Index	7279_h
Variablenname	Teached Limits Changed
Objekt-Code	VAR
Datentyp	Char
Kategorie	Optional

Beschreibung

Sub-Index	0
Zugriff	RO
Beschreibung	Antwort
Kategorie	M
Datentyp	Integer16
Wertebereich	0x4B4F (‚OK'), 0x5057 (‚WP')
Defaultwert	0x5057

Objekt $623D_{h:}$ Write Acceleration Rotation Int

Enthält die Beschleunigungswerte für die Rotationen als Integer16 (3x16 Bit (Integer16) Werte Accα, Accβ, Accχ in [°/s²]). Funktioniert ab Firmware 2.00a und installierten Messzellen mit Beschleunigungssensoren (optional).

Funktioniert diese Kommando trotz installierter Beschleunigungssensoren nicht (Unbekanntes Kommando), ist mind. ein Beschleunigungssensor defekt.

1x16 Bit Status: unter 8Bit = Messbereichsüberschreitung

Objektbeschreibung

Index	$623D_h$
Variablenname	Write Acceleration Rotation Int
Objekt-Code	RECORD
Datentyp	PDOmapping
Kategorie	Optional

Beschreibung

Sub-Index	1
Zugriff	RO
Beschreibung	Rotationsbeschleunigung bei Rotation um x [°/s²]
Kategorie	M
Datentyp	Integer16
Wertebereich	Unbeschränkt (Integer16)
Defaultwert	0

Sub-Index	2
Zugriff	RO
Beschreibung	Rotationsbeschleunigung bei Rotation um y [°/s²]
Kategorie	M
Datentyp	Integer16
Wertebereich	Unbeschränkt (Integer16)
Defaultwert	0

Sub-Index	3
Zugriff	RO
Beschreibung	Rotationsbeschleunigung bei Rotation um z [°/s²]
Kategorie	M
Datentyp	Integer16
Wertebereich	Unbeschränkt (Integer16)
Defaultwert	0

Sub-Index	4
Zugriff	RO
Beschreibung	Angaben über den aktuellen Status
Kategorie	M
Datentyp	Unsigned16
Wertebereich	Unbeschränkt (Unsigned16)
Defaultwert	0

5.2.3 Device Function block (Gerätespezifisch)

Objekt $2F26_{h:}$ Set Data Cycle

Setzt die Zykluszeit für periodisch zu wiederholende Befehle. Die Zykluszeit in [ms] wird mit 5 Char übermittelt; gültige Werte: 00001 ... 65767.

Nach Empfangen von „Start Cycle Mode“ wird der zuletzt eingestellte Befehl zyklisch abgearbeitet. Mit „Stop Cycle Mode“ wird der zyklische Betrieb beendet. Der eingestellte Wert wird im EEPROM gespeichert.

Vorsicht! Sollte der letzte Befehl an einen der Sensoren ein "Einstellungs-befehl" gewesen sein kann dies zu unvorhergesehenen Effekten führen.

Diese Funktion ist ab Firmware Version 1.30a verfügbar

Objektbeschreibung

Index	$2F26_h$
Variablenname	Set Data Cycle
Objekt-Code	ARRAY
Datentyp	Char
Kategorie	Optional

Beschreibung

Sub-Index	0
Zugriff	RO
Beschreibung	Anzahl der folgenden Bytes
Kategorie	M
Datentyp	Integer16
Wertebereich	5
Defaultwert	5

Sub-Index	1
Zugriff	RO
Beschreibung	Zykluszeit in [ms]
Kategorie	M
Datentyp	Char
Wertebereich	„00001“ – „65767“
Defaultwert	„01000“

Objekt $2F27_{h:}$ Data Cycle Changed

Die Antwort auf einen Änderung der Zykluszeit. Die Antwort ist „OK“ (0x4B4F) bei erfolgreicher Ausführung, ansonsten „WP“ (0x5057).

Objektbeschreibung

Index	$2F27_h$
Variablenname	Data Cycle Changed
Objekt-Code	VAR
Datentyp	Char
Kategorie	Optional

Beschreibung

Sub-Index	0
Zugriff	RO
Beschreibung	Antwort
Kategorie	M
Datentyp	Integer16
Wertebereich	0x4B4F (‚OK'), 0x5057 (‚WP')
Defaultwert	0x5057

Objekt $2F30_h$: Request for Reset

Sensor führt vollständigen Reset aus und bootet neu. Nach Ausführung dieses Befehls muss ein Passwort angegeben werden. Funktioniert auch im Fehlerfall.

Objektbeschreibung

Index	$2F30_h$
Variablenname	Request for Reset
Objekt-Code	VAR
Datentyp	Unsigned8
Kategorie	Mandatory

Beschreibung

Zugriff	RO
Beschreibung	Sensor führt vollständigen Reset aus und bootet neu. Funktioniert auch im Fehlerfall.
Datentyp	Unsigned8
Länge	1
Wertebereich	30_h
Defaultwert	30_h

Objekt $2F31_h$: Reset Done

Die Antwort auf einen Reset des Sensors. Die Antwort ist „OK" (0x4B4F) bei richtigem Passwort, ansonsten „WP" (0x5057).

Objektbeschreibung

Index	$2F30_h$
Variablenname	Reset Done
Objekt-Code	VAR
Datentyp	Char
Kategorie	Mandatory

Beschreibung

Sub-Index	0
Zugriff	RO
Beschreibung	Antwort
Kategorie	M
Datentyp	Integer16
Wertebereich	0x4B4F (‚OK'), 0x5057 (‚WP')
Defaultwert	0x5057

Objekt $2F32_{h:}$ Start Cycle Mode

Der zuletzt eingestellte Befehl wird zyklisch abgearbeitet.

Vorsicht! Sollte der letzte Befehl an einen der Sensoren ein "Einstellungs-befehl" gewesen sein kann dies zu unvorhergesehenen Effekten führen.

Ab Firmware Ver. 1.30a.

Objektbeschreibung

Index	$2F32_h$
Variablenname	Start Cycle Mode
Objekt-Code	VAR
Datentyp	Unsigned8
Kategorie	Optional

Beschreibung

Zugriff	RO
Beschreibung	Der zuletzt eingestellte Befehl wird zyklisch abgearbeitet.
Datentyp	Unsigned8
Länge	1
Wertebereich	32_h
Defaultwert	32_h

Objekt $2F34_{h:}$ Stop Cycle Mode

Stoppt die zyklische Abarbeitung des zuletzt eingestellten Befehls.

Ab Firmware Ver. 1.30a.

Objektbeschreibung

Index	$2F34_h$
Variablenname	Stop Cycle Mode
Objekt-Code	VAR
Datentyp	Unsigned8
Kategorie	Optional

Beschreibung

Zugriff	RO
Beschreibung	Stoppt die zyklische Abarbeitung des zuletzt eingestellten Befehls
Datentyp	Unsigned8
Länge	1
Wertebereich	34_h
Defaultwert	34_h

Objekt $2F62_h$: Request for Sensor Information

Objektbeschreibung

Index	$2F62_h$
Variablenname	Request for Sensor Information
Objekt-Code	VAR
Datentyp	Unsigned8
Kategorie	Mandatory

Beschreibung

Zugriff	RO
Beschreibung	Anfrage zum Senden der Sensorinformationen (Hardware Version, Software Version, Anzahl nachgiebiger Element, eingesetzter DSP, Anzahl der bisher erfolgreich ausgeführten Startvorgänge)
Datentyp	Unsigned8
Länge	1
Wertebereich	62_h
Defaultwert	62_h

Objekt $2F63_h$: Write Sensor Information

Hiermit werden alle wichtigen Sensorinformationen angezeigt ((Hardware Version, Software Version, Anzahl nachgiebiger Element, eingesetzter DSP, Anzahl der bisher erfolgreich ausgeführten Startvorgänge).

Objektbeschreibung

Index	$2F63_h$
Variablenname	Write Sensor Information
Objekt-Code	ARRAY
Datentyp	Char
Kategorie	Mandatory

Beschreibung

Sub-Index	0
Zugriff	RO
Beschreibung	Anzahl der folgenden Bytes
Kategorie	M
Datentyp	Integer16
Wertebereich	Unbeschränkt (Unsigned16)
Defaultwert	0

Sub-Index	1
Zugriff	RO
Beschreibung	alle wichtigen Sensorinformationen
Kategorie	M
Datentyp	Char
Wertebereich	Char
Defaultwert	Empty

Objekt $2F6E_{h:}$ Change Communication

Die Betriebsart der Kommunikation kann hiermit eingestellt werden. Gültige Werte: „RS 232“,“RS 485“(ab HW 3.2), “CAN[SPACE][SPACE][SPACE]“, “DEVNet“ (optional, muss freigeschaltet werden).

Objektbeschreibung

Index	$2F6E_h$
Variablenname	Change Communication
Objekt-Code	ARRAY
Datentyp	Char
Kategorie	Mandatory

Beschreibung

Zugriff	RO
Beschreibung	Änderung eines Limitwertes
Datentyp	ARRAY
Länge	6
Wertebereich	beliebig
Defaultwert	beliebig

Objekt $2F6F_{h:}$ Communication Parameters Changed

Die Antwort auf eine Änderung der Schnittstellenparameter. Die Antwort ist „OK“ (0x4B4F) bei erfolgreicher Ausführung, ansonsten „WP“ (0x5057). Bei ‚OK’ bootet der Sensor neu.

Objektbeschreibung

Index	$2F6F_h$
Variablenname	Communication Parameters Changed
Objekt-Code	VAR
Datentyp	Char
Kategorie	Mandatory

Beschreibung

Zugriff	RO
Beschreibung	Antwort auf eine Änderung der Schnittstellenparameter
Kategorie	M
Datentyp	Integer16
Länge	1
Wertebereich	0x4B4F (‚OK’), 0x5057 (‚WP’)
Defaultwert	0x4B4F

Objekt $2F6A_{h:}$ Set DEVNet ProducerSize

Falls DEVNet vorhanden und aktiviert kann hiermit die Producer Size eingestellt werden. Die Producer Size wird mit 3 Char übermittelt. Gültige Werte : 000 – 028

Nach erfolgreicher Ausführung bootet der Sensor mit den neuen Einstellungen.

Objektbeschreibung

Index	$2F6A_h$
Variablenname	Set DEVNet ProducerSize
Objekt-Code	ARRAY
Datentyp	Char
Kategorie	Mandatory

Beschreibung

Sub-Index	0
Zugriff	RO
Beschreibung	Anzahl der folgenden Bytes
Kategorie	M
Datentyp	Integer16
Wertebereich	3
Defaultwert	3

Sub-Index	1
Zugriff	RO
Beschreibung	DEVNet Producer Size
Kategorie	M
Datentyp	Char
Wertebereich	„000“-„028“
Defaultwert	

Objekt $2F6B_{h:}$ DEVNet ProducerSize Changed

Die Antwort auf einen Änderung der Producer Size Die Antwort ist „OK“ (0x4B4F) bei erfolgreicher Ausführung, ansonsten „WP“ (0x5057). Nach erfolgreicher Ausführung bootet der Sensor neu.

Objektbeschreibung

Index	$2F6B_h$
Variablenname	DEVNet ProducerSize Changed
Objekt-Code	VAR
Datentyp	Char
Kategorie	Mandatory

Beschreibung

Sub-Index	0
Zugriff	RO
Beschreibung	Antwort
Kategorie	M
Datentyp	Integer16
Wertebereich	0x4B4F (‚OK’), 0x5057 (‚WP’)
Defaultwert	0x5057

Objekt $2F6C_{h:}$ Request Communication Information

Befehl zur Abfrage der Schnittstelleninformationen.

Objektbeschreibung

Index	$2F6C_h$
Variablenname	Request Communication Information
Objekt-Code	VAR
Datentyp	Unsigned8
Kategorie	Mandatory

Beschreibung

Zugriff	RO
Beschreibung	Anfrage zum Senden aller Parameter der Schnittstellen betreffend. (CAN/DEVNet + Baudrate, CAN ID, RS232/RS485 + Baudrate) Ab Firmware Ver. 1.30a zusätzlich Zyklus Zeit für zyklischen Betrieb
Datentyp	Unsigned8
Länge	1
Wertebereich	$6C_h$
Defaultwert	$6C_h$

Objekt $2F6D_h$: Communication Information

Hiermit werden alle Parameter der Schnittstellen angezeigt (CAN/DEVNet + Baudrate, CAN ID, RS232/RS485 + Baudrate). Ab Firmware Ver. 1.30a wird zusätzlich Zyklus Zeit für zyklischen Betrieb angezeigt. Die Antwort erscheint im Klartext (Bytes = ASCII Code).

Objektbeschreibung

Index	$2F6D_h$
Variablenname	Write Communication Information
Objekt-Code	ARRAY
Datentyp	Char
Kategorie	Mandatory

Beschreibung

Sub-Index	0
Zugriff	RO
Beschreibung	Anzahl der folgenden Bytes
Kategorie	M
Datentyp	Integer16
Wertebereich	Unbeschränkt (Unsigned16)
Defaultwert	0

Sub-Index	1
Zugriff	RO
Beschreibung	Parameter der Schnittstellen
Kategorie	M
Datentyp	Char
Wertebereich	Char
Defaultwert	Empty

Objekt $2F72_{h:}$ New CAN Baudrate

Befehl zum Ändern der CAN Baudrate. Die Baudrate wird mit 4 Char übermittelt; gültige Werte in Baud: 0010, 0025, 0050, 0100, 0125, 0250, 0500, 0800, 1000

Nach erfolgreicher Ausführung bootet der Sensor mit den neuen Einstellungen (Defaulteinstellung 0500 [kBaud]).

Objektbeschreibung

Index	$2F72_h$
Variablenname	New CAN Baudrate
Objekt-Code	ARRAY
Datentyp	Char
Kategorie	Mandatory

Beschreibung

Sub-Index	0
Zugriff	RO
Beschreibung	Anzahl der folgenden Bytes
Kategorie	M
Datentyp	Integer16
Wertebereich	4
Defaultwert	4

Sub-Index	1
Zugriff	RO
Beschreibung	Neue Baudrate
Kategorie	M
Datentyp	Char
Wertebereich	Char
Defaultwert	„0500“

Objekt $2F73_{h:}$ CAN Baudrate Changed

Die Antwort auf einen Änderung der CAN Baudrate. Die Antwort ist „OK“ (0x4B4F) bei erfolgreicher Ausführung, ansonsten „WP“ (0x5057). Nach erfolgreicher Ausführung bootet der Sensor neu.

Objektbeschreibung

Index	$2F73_h$
Variablenname	CAN Baudrate Changed
Objekt-Code	VAR
Datentyp	Char
Kategorie	Mandatory

Beschreibung

Sub-Index	0
Zugriff	RO
Beschreibung	Antwort
Kategorie	M
Datentyp	Integer16
Wertebereich	0x4B4F (‚OK’), 0x5057 (‚WP’)
Defaultwert	0x5057

Objekt $2F74_{h:}$ Set CAN ID

Befehl zum Ändern der CAN Identifizierung des Sensors. Die ID wird mit 2 Char übermittelt. Gültige Werte : 00 – 63. Im Identifier muss dieser Wert in HEX umgerechnet werden!!

Nach erfolgreicher Ausführung bootet der Sensor mit den neuen Einstellungen (Defaulteinstellung 05).

Objektbeschreibung

Index	$2F74_h$
Variablenname	Set CAN ID
Objekt-Code	ARRAY
Datentyp	Char
Kategorie	Mandatory

Beschreibung

Sub-Index	0
Zugriff	RO
Beschreibung	Anzahl der folgenden Bytes
Kategorie	M
Datentyp	Integer16
Wertebereich	2
Defaultwert	2

Sub-Index	1
Zugriff	RO
Beschreibung	CAN ID
Kategorie	M
Datentyp	Char
Wertebereich	Char
Defaultwert	„05“

Objekt $2F75_{h:}$ CAN ID Changed

Die Antwort auf einen Änderung der CAN ID. Die Antwort ist „OK“ (0x4B4F) bei erfolgreicher Ausführung, ansonsten „WP“ (0x5057). Nach erfolgreicher Ausführung bootet der Sensor neu.

Objektbeschreibung

Index	$2F75_h$
Variablenname	CAN ID Changed
Objekt-Code	VAR
Datentyp	Char
Kategorie	Mandatory

Beschreibung

Sub-Index	0
Zugriff	RO
Beschreibung	Antwort
Kategorie	M
Datentyp	Integer16
Wertebereich	0x4B4F (‚OK'), 0x5057 (‚WP')
Defaultwert	0x5057

Objekt $2F76_{h:}$ New Baudrate

Befehl zum Ändern der Baudrate. Die Baudrate wird mit 6 Char übermittelt; gültige Werte in Baud: 001200, 002400, 009600, 019200, 038400, 057600, 115200, 230400, 460800, 921600

Nach erfolgreicher Ausführung bootet der Sensor mit den neuen Einstellungen (Defaulteinstellung 009600).

Objektbeschreibung

Index	$2F76_h$
Variablenname	New Baudrate
Objekt-Code	ARRAY
Datentyp	Char
Kategorie	Mandatory

Beschreibung

Sub-Index	0
Zugriff	RO
Beschreibung	Anzahl der folgenden Bytes
Kategorie	M
Datentyp	Integer16
Wertebereich	6
Defaultwert	6

Sub-Index	1
Zugriff	RO
Beschreibung	Neue Baudrate
Kategorie	M
Datentyp	Char
Wertebereich	Char
Defaultwert	„009600“

Objekt $2F77_{h:}$ Baudrate Changed

Die Antwort auf einen Änderung der Baudrate. Die Antwort ist „OK“ (0x4B4F) bei erfolgreicher Ausführung, ansonsten „WP“ (0x5057). Nach erfolgreicher Ausführung bootet der Sensor neu.

Objektbeschreibung

Index	$2F77_h$
Variablenname	Baudrate Changed
Objekt-Code	VAR
Datentyp	Char
Kategorie	Mandatory

Beschreibung

Sub-Index	0
Zugriff	RO
Beschreibung	Antwort
Kategorie	M
Datentyp	Integer16
Wertebereich	0x4B4F (‚OK'), 0x5057 (‚WP')
Defaultwert	0x5057

Objekt $2F66_h$: Flash mode

Leitet das Einspielen neuer Firmware ein.

Nach Eingabe des Passwortes und einer kurzen Wartzeit von ca. 2 Sek. ist der Bootloader aktiv (LED ist dunkel). Nun kann die neue Firmware über die serielle Schnittstelle mit dem Flash-Tool von TI aufgespielt werden. Nach erfolgreichem Update muss der Sensor für mindestens 25 Sek. von der Spannungsversorgung getrennt werden und kann anschließend wieder in Betrieb gehen. Alle Einstellungen wie Baudrate, ID, Positions- und Kraftgrenzen bleiben dabei erhalten.

Objektbeschreibung

Index	$2F66_h$
Variablenname	Flash mode
Objekt-Code	VAR
Datentyp	Unsigned8
Kategorie	Mandatory

Beschreibung

Zugriff	RO
Beschreibung	Leitet das Einspielen neuer Firmware ein.
Datentyp	Unsigned8
Länge	1
Wertebereich	66_h
Defaultwert	66_h

Objekt $2F67_h$: Sensor Flashed

Die Antwort auf einen Flash des Sensors. Die Antwort ist „OK" (0x4B4F) bei richtigem Passwort, ansonsten „WP" (0x5057).

Objektbeschreibung

Index	$2F67_h$
Variablenname	Sensor Flashed
Objekt-Code	VAR
Datentyp	Char
Kategorie	Mandatory

Beschreibung

Sub-Index	0
Zugriff	RO
Beschreibung	Antwort
Kategorie	M
Datentyp	Integer16
Wertebereich	0x4B4F (‚OK'), 0x5057 (‚WP')
Defaultwert	0x5057

Objekt $2F90_{h:}$ Sync

Der Sensor antwortet wie auf den letzten Befehl. Es antworten alle am Bus angeschlossenen Sensoren mit dem zuletzt eingestellten Befehl.

Vorsicht! Sollte der letzte Befehl an einen der Sensoren ein "Einstellungs-befehl" gewesen sein kann dies zu unvorhergesehenen Effekten führen.

Objektbeschreibung

Index	$2F90_h$
Variablenname	Sync
Objekt-Code	VAR
Datentyp	Unsigned8
Kategorie	Mandatory

Beschreibung

Zugriff	RO
Beschreibung	Der Sensor antwortet wie auf den letzten Befehl.
Datentyp	Unsigned8
Länge	1
Wertebereich	90_h
Defaultwert	90_h

5.2.4 Fehler und Warnungen

Objekt 2001_h: Error

Das Fehlerobjekt beginnt immer mit dem Wert 0x21 (Unsigned 8), anschließend folgt ein 16 Bit (Integer16) Wert mit dem Fehlercode. Es wird versucht die Fehlermeldungen auf allen Schnittstellen auszugeben. Die Fehlermeldungen werden grob in Betriebsfehler und Bedienfehler und in die Fehlerklassen „Warnungen", „behebbare Fehler" und „schwerwiegende Fehler" eingeteilt.

Objektbeschreibung

Index	2001_h
Variablenname	Error
Objekt-Code	RECORD
Datentyp	PDOmapping
Kategorie	Mandatory

Beschreibung

Sub-Index	0
Zugriff	RO
Beschreibung	Identifier für den Fehler
Kategorie	M
Datentyp	Unsigned8
Wertebereich	0x21
Defaultwert	0x21

Sub-Index	0
Zugriff	RO
Beschreibung	Fehlercode
Kategorie	M
Datentyp	Integer16
Wertebereich	Unbeschränkt (Integer16)
Defaultwert	-

Fehlercodes

Betriebsfehler: Behebbar

Code	Name	Beschreibung
5021_h	EEPROM Checksum1 wrong	Cecksum1 im EEPROM stimmt nicht. Defaultboot (DIP 4) behebt diesen Fehler. Funktioniert dies nicht, ist das EEPROM defekt.
5022_h	EEPROM Part A deleted	Part A des EEPROMs ist gelöscht worden. Defaultboot (DIP 4) behebt diesen Fehler. Funktioniert dies nicht, ist das EEPROM defekt.
5025_h	Serial Defective	Die serielle Schnittstelle hat ein Problem. Der Sensor versucht selbstständig den Fehler zu beheben. Erscheint diese Fehlermeldung häufiger ist die Serielle Schnittstelle des Sensors defekt.
8029_h	CAN Communication error	Die CAN Kommunikation funktioniert nicht ordnungsgemäß. Kabel und Einstellungen prüfen.

Betriebsfehler: Schwerwiegend

Code	Name	Beschreibung
2040_h	Current Error Measuring Cell X	Stromfehler in Messzelle 0-7. Wird beim Hochfahren des Sensors ermittelt. Wahrscheinlich gibt es einen Kurzschluss in der entsprechenden Messzelle.
3041_h	Voltage Error Measuring Cell X	Spannungsfehler in Messzelle 0-7. Wird beim Hochfahren des Sensors ermittelt. Wahrscheinlich gibt es einen Kurzschluss in der entsprechenden Messzelle.
5042_h	Spring Fracture	Federbruch. Ein elastisches Element im Sensor ist defekt.
5043_h	EEPROM Checksum2 Wrong	Checksum2 im EEPROM stimmt nicht. Die sensorspezifischen Parameter sind ungültig Das EEPROM wurde manipuliert oder ist defekt.
5044_h	EEPROM Part B deleted	EEPROM Part B ist gelöscht worden. Das EEPROM wurde manipuliert oder ist defekt.
5045_h	EEPROM Timeout	EEPROM konnte nicht geschrieben werden. Das EEPROM ist defekt.
5046_h	DSP_Error	DSP ist nicht bekannt. (Darf niemals passieren)
2047_h	Current Error horizontal	Strom Fehler im horizontalen Messsystem. Der Strom im horizontalen Messsystem ist zu niedrig => eine Messzelle ist defekt.
3048_h	Voltage Error vertical	Strom Fehler im vertikalen Messsystem. Der Strom im vertikalen Messsystem ist zu niedrig => eine Messzelle ist defekt.
5049_h	CAN Controller defective	CAN Controller defekt. Der Sensor kann nur noch über die serielle Schnittstelle betrieben werden. Der CAN Controller muss ausgetauscht werden.
5050h	Spring-Ground Short-Circuit	Feder hat Masseschluss. Ein elastisches Element verursacht einen Kurzschluss mit Masse => Ein defektes Element kann nicht mehr erkannt werden (ab HW 3.2).

Bedienfehler: Warnung

Code	Name	Beschreibung
0021_h	Unbekanntes Kommando	Bei installierter Option Beschleunigungssensor und Versuch

		eines der folgenden Kommandos abzusetzen ist mindestens ein Beschleunigungssensor defekt und muss getauscht werden. Kommandos: Request Acceleration Float, Request Acceleration Int, Request Acceleration Translation, Request Acceleration Rotation
0057_h	Falscher Parameter	Es wurde dem Sensor ein falscher Parameter bzw. das falsche Passwort übermittelt.

Bedienfehler: Behebar

Code	**Name**	**Beschreibung**
3003_h	Input Voltage too high	Eingangsspannung zu hoch. LED leuchtet dauerhaft rot! Eingangsspannung auf 11 - 26 V senken!
3004_h	Input Voltage too low	Eingangsspannung zu niedrig. LED leuchtet dauerhaft rot! Eingangsspannung auf 11 - 26 V erhöhen!
4005_h	Temperature too high	
4006_h	Temperature too low	
8023_h	Serial Baudrate wrong	Serielle Baudrate ist ungültig. Serielle Baudrate wird auf 9600Baud eingestellt. Gültige Baudrate einstellen bzw. Defaultboot behebt diesen Fehler.
8024_h	Serial Timeout	CAN Timeout. Bei Übergabe von Parametern wurde zu lange gewartet. Befehl wiederholen.
8026_h	CAN Baudrate wrong	Die CAN Baudrate ist ungültig und wird auf 500kBit eingestellt. Gültige Baudrate einstellen bzw. Defaultboot (FIME : LINK) behebt diesen Fehler.
8027_h	CAN MAC ID false	Ungültige MAC ID. MAC ID wird auf 5 gestellt. Gültige MAC ID einstellen bzw. Defaultboot behebt diesen Fehler.
8028_h	CAN Timeout	CAN Timeout. Bei Übergabe von Parametern wurde zu lange gewartet. Befehl wiederholen.

PAPAS

PLUG AND PLAY ANTRIEBS- UND STEUERUNGSKONZEPTE FÜR DIE PRODUKTION VON MORGEN.

Geräteprofil für PFO

PAPAS-Geräteprofil für Programmierbare Fokussieroptik (PFO) in Kombination mit externen, bewegten Achsen

Projektpartner

DLR KUKA AMATEC TRUMPF Lenze SCHUNK Spann- und Greiftechnik

1. Motivation

Dieses Dokument beschreibt eine mögliche Datenübertragungs-Schnittstelle zwischen der PFO (Programmierbare Fokusieroptik) und einem Roboter auf Basis, des im BMBF-Förderprojektes PAPAS, erarbeiteten Protokollstandards.

Beschrieben sind:

- Parametrierdaten
- Steuerungsdaten
- Zustandsmaschinen

Grundlage war das CANopen-Profil für „Drives and motion control (CiA DSP-402 V2.0) sowie das CIA Standard-Dokument DS-301.

Diverse Variablen sind derzeit noch festen Einheiten zugeordnet und nicht über Variablen für die Einheit frei parametrierbar.
Diese und andere Unzulänglichkeiten für eine offene Lösung bedürfen einer Anpassung in einem weiteren Schritt.
Auch wurde nicht detailliert untersucht inwieweit eine Anpassung auf das bestehende Profil Variable und Objekte bereits ersetzen könnte, bzw. Überschneidungen zu anderen Profilen, außer den unten genannten, existieren.

2. Referenzen

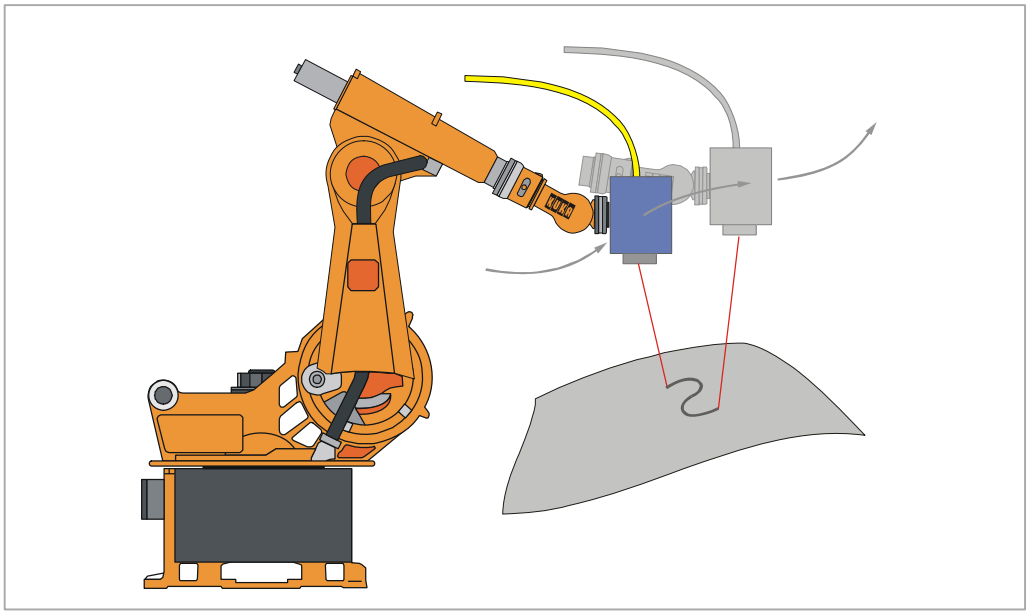
Besprechungsprotokoll zum PAPAS Projekt 2004-07-08
CIA Draft Standard 301, Version 4.02
CANopen-Profil für I/O-Module (CiA DSP-401 V1.4)
CANopen-Profil für „Drives and motion control (CiA DSP-402 V2.0)

3. Verwendete Abkürzungen

CAN	Controller Area Network. Spezifiziert in ISO11898
IPO-Takt	Interpolationstakt des Roboters
PAPAS	Plug And Play in der Antriebs- und Steuerungstechnik
PFO	Programmierbare Fokussieroptik
TCP	Tool Center Point
WD	Working Distance; Arbeitsabstand gemessen von der Objektivunterseite

4. Prinzipielle Funktionsweise

Ein Roboter führt eine Strahlführungseinheit (PFO) über ein Werkstück, die, synchronisiert mit der Position des Roboters, während der Bewegung Konturprogramme schweißen wird.



Im Rahmen dieses Projekts wird die Schnittstelle zwischen Robotersteuerung und Strahlführungseinheit (PFO) beschrieben, die zur Synchronisierung benötigt wird.

Der Roboter führt die PFO kontinuierlich im objektivspezifischen Arbeitsabstand (WD, Working Distance) über das Werkstück. Die PFO steht flächennormal zum Werkstück und deren Spiegeloptik erlaubt es, in einem ebenfalls objektivspezifischen Fenster (z.B.: 180 x 100 mm) zweidimensional den Strahl abzulenken. In der PFO sind mehrere Figurprogramme abrufbar.

Ziel ist es, eine Koordination der Roboterbewegung mit der PFO zu erreichen. Dazu überträgt der Roboter seine aktuelle Position zyklisch an die PFO, diese koordiniert dies mit der eigenen Bewegung und ermöglicht so das Schweißen von Figuren während der Roboter eine Bahn abfährt.

4.1. Kommunikation Roboter-PFO

Die PFO benötigt die Positionsdaten interpoliert und schneller als mit 12 ms.

Typischer Interpolationstakt: 2 ms.

Der Roboter fährt eine Kontur über dem Bauteil ab, diese Bewegung besteht aus einem oder mehreren Punkten. Während dieser Bewegung des Roboters wird durch den PFO eine Figur erzeugt.

Die Figur-ID (Geometrieselektion) muss mindestens 40 ms vor dem eigentlichen Start an die PFO übermittelt werden.

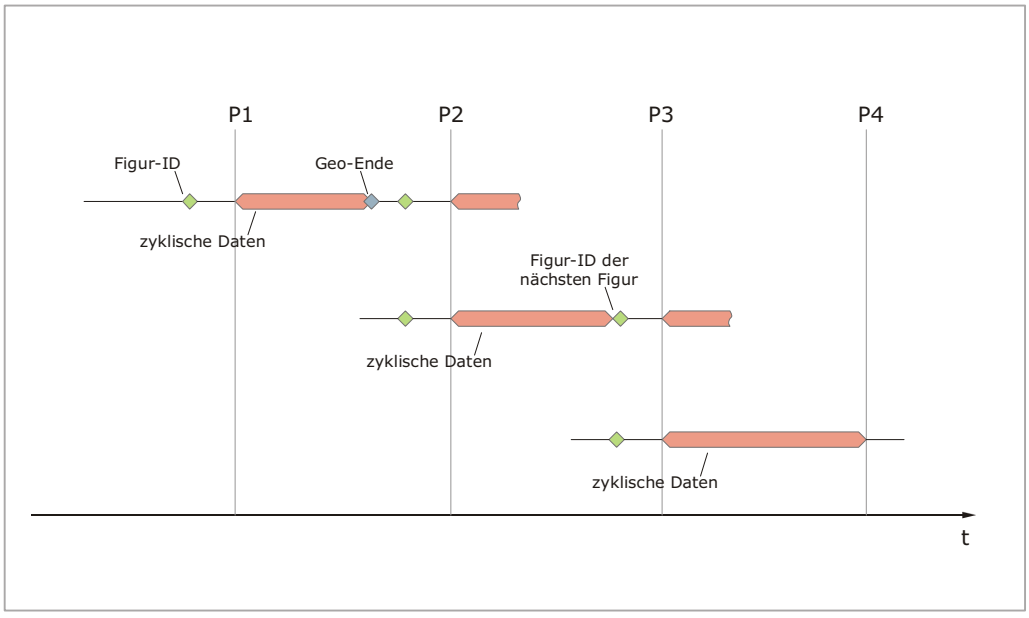
Der durch den Roboter gegebene Startpunkt ist gleichzeitig Ursprung des lokalen Koordinatensystems der Figur (PFO-Koordinatensystem).

Am Startpunkt beginnt der Roboter mit der zyklischen Übertragung von Positionsdaten im 2 ms Zeitraster. Die Positionsdaten sind konfigurierbar absolut oder relativ, bezogen auf den Startpunkt als lokales Basis-System (PFO-Koordinatensystem).

Der Roboter interpoliert die Positionsdaten linear im 2 ms Takt, basierend auf 12 ms Interpolationstakt.

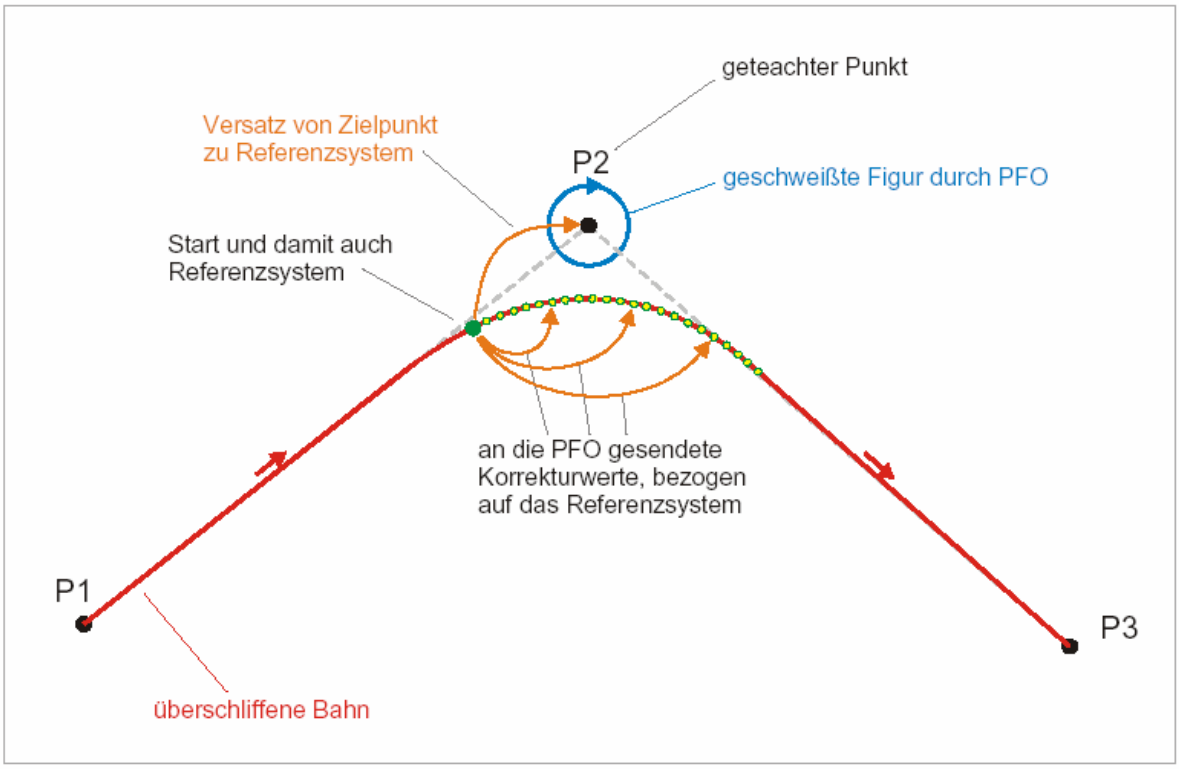
Die Übertragung von zyklischen Daten endet unter folgenden Bedingungen:

- wenn die PFO das Ende der Geometrie (Signal *Geo-Ende*) signalisiert (Normalfall)
- wenn vor dem Ende der Geometrie (*Geo-Ende*), durch das Roboterprogramm die nächste Figur-ID übermittelt wird, d.h. die die Figur konnte nicht beendet werden.
- wenn vor dem Ende der Geometrier (*Geo-Ende*), das Ende des Bewegungssatzes erreicht wird – verursacht dadurch, dass der Interpolator keine gültigen Daten mehr liefert.
- durch ein STOP-Kommando im Roboterprogramm



In allen Fällen wird als letzte zyklische Message ein "Positions-Daten.1" gesendet, wobei das Stop-Bit gesetzt ist.

Um die exakte Position der geteachten Geometrie zu gewährleisten, wird bei der Selektion der Schweißgeometrie zusätzlich eine Raumkoordinate (Position der Geometrie bezogen auf Referenzsystem) an die PFO übertragen. Somit wird die Position (P2) der Geometrie auf dem Werkstück invariant vom Roboterinterpolationstakt (IPO-Takt) und eventuell vorhandenen Überschleifparametern (geteachter Punkt liegt, durch Überschleifen, nicht auf der Roboterbahn (rote Kurve)).



4.2. Standardisierung über Profile

Für Systemintegration soll ein Profil generiert werden um die steuerungstechnische Anbindung zu vereinfachen.

Dieses Profil wird an die frei verfügbaren, offenen CANopen-Profile angelehnt.

Die Standardisierung soll es ermöglichen, die Anbindung einer Scanneroptik an bewegte Achsen (Roboter, Werkzeugmaschine,...) zu beschreiben und zu abstrahieren.

Als Basis kann das bestehende CANOpen-Profil für „Antriebe und Bewegungskontrolle“ dienen, welches um einen „Transformationsmode“ erweitert wird.

Die beschriebenen Mappings sind, in dem bei CANOpen definierten, herstellerspezifischen Bereich.

4.3. Objektverzeichnis

Das Objektverzeichnis stellt eine Gruppierung von Objekten dar, die mit Hilfe eines 16-Bit Index adressierbar sind. Die grundsätzliche Gliederung entspricht der von anderen Standard-Bussystemen (z.B.: Profibus).

Index (hexadezimal)	**Objekt**	**Beschreibung**
0000	NC	Nicht benutzt
0001 – 001F	Static data types	Standard Datentypen wie Bool, Integer 8Bit, Integer 16Bit, float, string
0020 – 003F	Complex data types	Aus Standardtypen abgeleitete Datentypen (Strukturen)
0040 – 005F	Manufacturer specific datatypes	Wie „complex data types“ jedoch spezifisch für ein bestimmtes Gerät
0060 – 0FFF	RESERVED	Reserviert für zukünftige Erweiterungen
1000 – 1FFF	Communication profile area	Für die Kommunikation über CAN spezifische Parameter (Global für alle Geräte)
2000 – 5FFF	Manufacturer specific profile area	Herstellerspezifische Profile (nicht standardisiert)
6000 – 9FFF	Standardisised Device profile area	Objekte für standardisierte Profile
A000 - FFFF	RESERVED	Reserviert für zukünftige Erweiterungen

Index und Subindex

Die Objekte innerhalb des Objektverzeichnisses werden über den 16-Bit Index angesprochen.

Für einfache Objekte, die nur aus einer Variablen eines Datentypes bestehen ist das ausreichend. Komplexe Objekte aus mehreren Datentypen und Variablen (Strukturen, Array) wir über den Subindex auf die einzelnen Elemente zugegriffen; das gesamte Objekt wird nach wie vor über den 16-Bit Index angesprochen.

5. Fehlermeldungen

Fehlermeldungen (Emergency-Botschaften) werden vom Gerät selbständig übermittelt, um einen internen Fehlerzustand zu signalisieren.

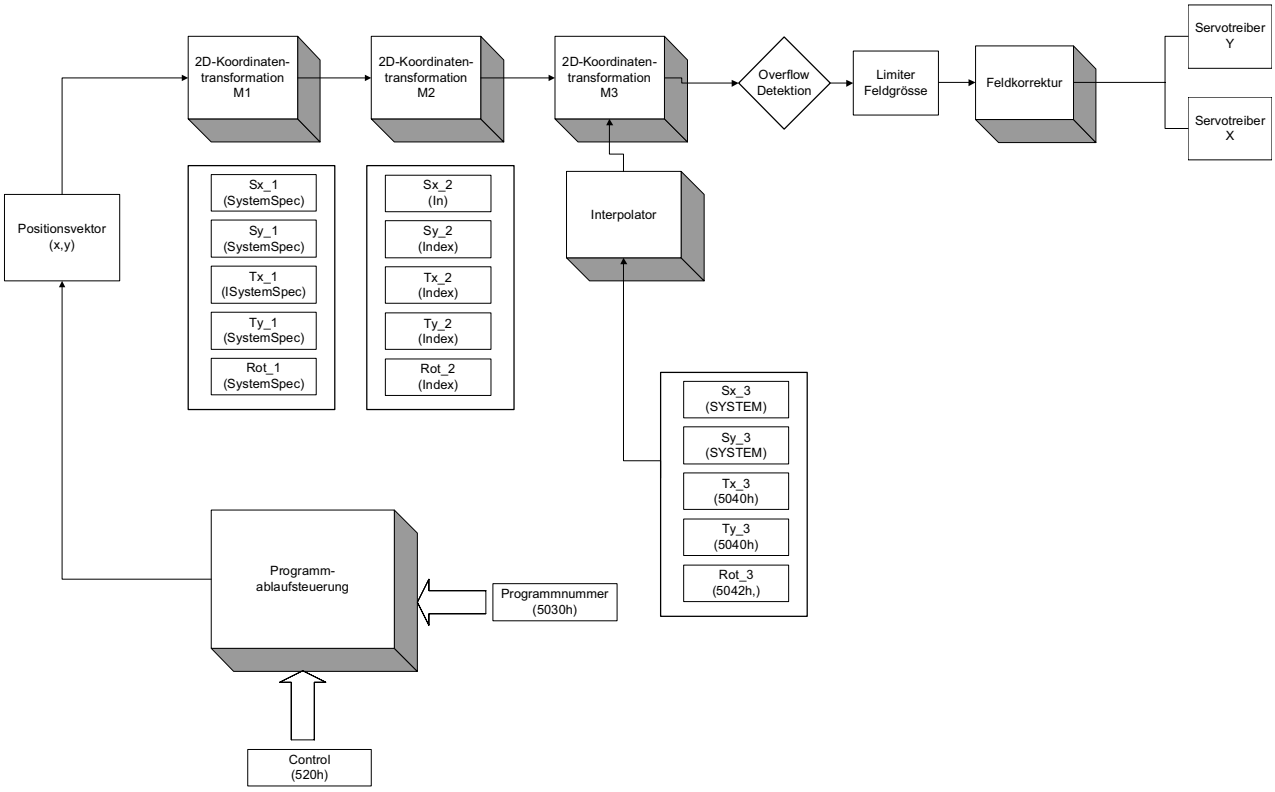
Der übertragene 16-Bit-Fehlercode ist gemäß dem DS-301- Standard in verschieden standardisierte Gruppen aufgeteilt (z.B.: Spannungsfehler, Stromfehler, Temperaturfehler, .. siehe Tabelle) und enthält eine gerätespezifischen Bereich für Meldungen die nicht in eine der Standardgruppen fallen.

Code	Bedeutung	Definiert durch:
0xxxh	Kein Fehler	DS 301
1xxxh	Nicht definierter Fehlertyp	DS 301
2xxxh	Stromfehler	DS 301
3xxxh	Spannungsfehler	DS 301
3800h		Diese Profil
3810h	Überspannung interne Versorgung	Diese Profil
3811h	+ 24 V	Diese Profil
3812h	- 24 V	Diese Profil
3813h	+ 15 V	Diese Profil
3814h	- 15 V	Diese Profil
3815h	+ 5 V	Diese Profil
3820h	Unterspannung interne Versorgung	Diese Profil
3821h	+ 24 V	Diese Profil
3822h	- 24 V	Diese Profil
3823h	+ 15 V	Diese Profil
3824h	- 15 V	Diese Profil
3825h	+ 5 V	Diese Profil
40xxh	Temperaturfehler	DS 301
4810h	Übertemperatur	Diese Profil
4811h	Steuerungsplatine	Diese Profil
4812h	Scanner X	Diese Profil
4813h	Scanner Y	Diese Profil
4814h	Flansch X-Scanner	Diese Profil
4815h	Flansch Y-Scanner	Diese Profil
4816h	Kühlwasser	Diese Profil
4820h	Untertemperatur	Diese Profil
4821h	Steuerungsplatine	Diese Profil
4822h	Scanner X	Diese Profil
4823h	Scanner Y	Diese Profil
4824h	Flansch X-Scanner	Diese Profil
4825h	Flansch Y-Scanner	Diese Profil
4826h	Kühlwasser	Diese Profil
50xxh	Hardwarefehler	DS 301
60xxh	Softwarefehler	DS 301
70xxh	Zusatzmodule	DS 301
80xxh	CAN-Kommunikation	DS 301
90xxh	Externer Fehler	DS 301

Code	Bedeutung	Definiert durch:
FFxxh	Gerätespezifische Fehler	DS 301
FF10h	Fehler bei der Geometrieausführung	Diese Profil
FF11h	Falscher Brennweite	Diese Profil
FF12h	X-Auslenkung außerhalb Bearbeitungsfeld	Diese Profil
FF13h	Y-Auslenkung außerhalb Bearbeitungsfeld	Diese Profil
FF14h	X,Y-Auslenkung außerhalb Bearbeitungsfeld	Diese Profil
FF15h	Zahlenüberlauf bei Koordinatenberechnung	Diese Profil
FF16h	Geometrienummer nicht in PFO vorhanden	Diese Profil
FF17h	Abbruch der Geometrie vor Fertigstellung	Diese Profil
FF18h	Timeout zyklische Interpolationsdaten	Diese Profil
FF19h	Inkompatible Version des Roboters	Diese Profil
FF20h	Fehler Kommunikation mit Lasergerät	Diese Profil
FF21h	Keine Verbindung zu Lasergerät (Busoff)	Diese Profil
FF22h	Eingangspuffer Überlauf	Diese Profil
FF23h	Ausgangspuffer Überlauf	Diese Profil
FF30h	Fehler Synchronistion mit Lasergerät	Diese Profil
FF31h	Programmstart und Laser nicht bereit	Diese Profil
FF32h	Timeout PULSDAUER aktiv	Diese Profil
FF33h	Timeout PULSDAUER inaktiv	Diese Profil
FF34h	Unbekannter Lasertyp	Diese Profil
FF35h	Laser-Programm abgebrochen	Diese Profil

6. Grundsätzliche Definitionen

6.1. Prinzip



6.2. PDO-Mapping im Objektverzeichnis

Herstellerspezifischer Profilbereich ab 5000h und PDO ab 21 (Konfliktvermeidung mit DSP-402)

Empfangs-PDOs

PDO Nummer	Mapping Objekt Index	Mapping Objekt Name	M/O	Bemerkung
21	5020 5040	Controlword Translation	M	Controlword zur Kontrolle der Statemaschine (Start, Stopp, Modus) Translatorischer Anteil der Roboterbewegung X, Y, Z)
22	5042	Rotation	M	Rotatorischer Anteil der Roboterbewegung (A-Winkel (→ Winkel um Z-Achse) B-Winkel (→ Winkel um Y-Achse) C-Winkel (→ Winkel um X-Achse)
23	5030 5060	Geonumber 2D-Transformation	M	Geometrienummer Koordinatentransformation für Geometrieposition (Translation X, Translation Y, Rotation um Z)
24	5070	Teachbetrieb	M	Simulationsmodus

Sende-PDOs

PDO Nummer	Mapping Objekt Index	Mapping Objekt Name	M/O	Bemerkung
21	5021	Statuswort	M	Statuswort
22	5080	Tooldata X	M	Tooldaten x-Achse
23	5081	Tooldata Y	M	Tooldaten y-Achse
22	5082	Tooldata Z	M	Tooldaten z-Achse
22	5083	Tooldata A	M	Tooldaten A-Winkel
22	5084	Tooldata B	M	Tooldaten B-Winkel
22	5085	Tooldata C	M	Tooldaten C-Winkel

1. Empfangs-PDO

Index	Subindex	Name	Default-Wert
1414h	0	Number of entries	
	1	COB-ID used by PDO	
	2	Transmission type	
	3	Inhibit time	
	4	Reserved	
	5	Event timer	

Index	Subindex	Name	Default-Wert
1614h	0	Number of mapped objekts	2
	1	Controlword	5020
	2	Kartesische Achsposition	5040

2. Empfangs-PDO

Index	Subindex	Name	Default-Wert
1415h	0	Number of entries	
	1	COB-ID used by PDO	
	2	Transmission type	
	3	Inhibit time	
	4	Reserved	
	5	Event timer	

Index	Subindex	Name	Default-Wert
1615h	0	Number of mapped objekts	4
	1	Eulerwinkel Achspositionen	5042 0000

3. Empfangs-PDO

Index	Subindex	Name	Default-Wert
1416h	0	Number of entries	
	1	COB-ID used by PDO	
	2	Transmission type	
	3	Inhibit time	
	4	Reserved	
	5	Event timer	

Index	Subindex	Name	Default-Wert
1616h	0	Number of mapped objekts	2
	1	Geometrienummer	5030 0000
	2	2D-Transformation der Geometrie	5060 0000

4. Empfangs-PDO

Index	Subindex	Name	Default-Wert
1417h	0	Number of entries	
	1	COB-ID used by PDO	
	2	Transmission type	
	3	Inhibit time	
	4	Reserved	
	5	Event timer	

Index	Subindex	Name	Default-Wert
1617h	0	Number of mapped objekts	
	1	Testmode für Einrichtbetrieb	5070 0000

1. Sende-PDO

Index	Subindex	Name	Default-Wert
1814h	0	Number of entries	
	1	COB-ID used by PDO	
	2	Transmission type	
	3	Inhibit time	
	4	Reserved	
	5	Event timer	

Index	Subindex	Name	Default-Wert
1A14h	0	Number of mapped objekts	1
	1	Statusword	5050 0000

2. Sende-PDO

Index	Subindex	Name	Default-Wert
1815h	0	Number of entries	
	1	COB-ID used by PDO	
	2	Transmission type	
	3	Inhibit time	
	4	Reserved	
	5	Event timer	

Index	Subindex	Name	Default-Wert
1A15h	0	Number of mapped objekts	1
	1	Tooldata X	5080 0000

3. Sende-PDO

Index	Subindex	Name	Default-Wert
1816h	0	Number of entries	
	1	COB-ID used by PDO	
	2	Transmission type	
	3	Inhibit time	
	4	Reserved	
	5	Event timer	

Index	Subindex	Name	Default-Wert
1A16h	0	Number of mapped objekts	1
	1	Tooldata Y	5081 0000

4. Sende-PDO

Index	Subindex	Name	Default-Wert
1817h	0	Number of entries	
	1	COB-ID used by PDO	
	2	Transmission type	
	3	Inhibit time	
	4	Reserved	
	5	Event timer	

Index	Subindex	Name	Default-Wert
1A17h	0	Number of mapped objekts	1
	1	Tooldata Z	5082 0000

5. Sende-PDO

Index	Subindex	Name	Default-Wert
1818h	0	Number of entries	
	1	COB-ID used by PDO	
	2	Transmission type	
	3	Inhibit time	
	4	Reserved	
	5	Event timer	

Index	Subindex	Name	Default-Wert
1A18h	0	Number of mapped objekts	1
	1	Tooldata A	5083 0000

6. Sende-PDO

Index	Subindex	Name	Default-Wert
1819h	0	Number of entries	
	1	COB-ID used by PDO	
	2	Transmission type	
	3	Inhibit time	
	4	Reserved	
	5	Event timer	

Index	Subindex	Name	Default-Wert
1A19h	0	Number of mapped objekts	1
	1	Tooldata B	5084 0000

7. Sende-PDO

Index	Subindex	Name	Default-Wert
181Ah	0	Number of entries	
	1	COB-ID used by PDO	
	2	Transmission type	
	3	Inhibit time	
	4	Reserved	
	5	Event timer	

Index	Subindex	Name	Default-Wert
1A1Ah	0	Number of mapped objekts	1
	1	Tooldata C	5085 0000

7. Objektverzeichnis

Erweiterung des Objektverzeichnisses gemäß Profil DSP-402 V2.0 um folgende Objekte im herstellerspezifischen Profilbereich

Index	Objekt	Name	Datentyp	Zugriff	M/O
5010	ARRAY	Version		R/W	
5020	VAR	Controlword, Kontrollwort zur Steuerung der Zustandsmaschinen	UNSIGNED16	R/W	M
5021	VAR	Statusword, Status der internen Zustandsmaschinen	UNSIGNED16	R	M
5030	VAR	Geometrienummer	UNSIGNED16	W	M
5040	ARRAY	Kartesische Koordinaten der Roboterbahn für 3D-Koordinatentransformation (Translatorischer Anteil)	SIGNED16	W	M
5042	ARRAY	Kartesische Koordinaten der Roboterbahn für 3D-Koordinatentransformation (Rotatorischer Anteil)	UNSIGNED16	W	M
5044	ARRAY	Skalierungsdaten für die Koordinatenachsen	SIGNED16	W	O
5060	ARRAY	2D-Koordinatentransformation für Geometrietransformation		W	M
5070	VAR	Testmodus für Einrichtbetrieb		W	M
5080	ARRAY	TOOLDATEN	INTEGER32	R	O

8. Beschreibung der Objekte

Objekt 5010h : Version

Dieses Objekt wurde für Versionierung geschaffen und ermöglicht ein Abgleich zwischen Roboter und PFO auf kompatible Versionen.

INDEX	5010h
Variablenname	Version
Objekt-Code	VAR
Datentyp	ARRAY
Kategorie	Mandatory

Beschreibung

Sub-Index	0
Zugriff	Read / write
Beschreibung	Anzahl Einträge
Datentyp	
Länge	
Wertebereich	
Defaultwert	0

Sub-Index	1
Zugriff	Read / write
Beschreibung	Identifikationsstring (z.B.: PFO) Kennzeichnet den Teilnehmer
Datentyp	STRING
Länge	4
Wertebereich	STRING
Defaultwert	„PFO/0“

Sub-Index	2
Zugriff	Read / write
Beschreibung	Major build version Muß auf beiden Teilnehmern identisch sein
Datentyp	UNSIGNED8
Länge	
Wertebereich	UNSIGNED8
Defaultwert	0

Sub-Index	3
Zugriff	Read / write
Beschreibung	Minor build version Ebenfalls identisch auf beiden Teilnhmern
Datentyp	UNSIGNED8
Länge	
Wertebereich	UNSIGNED8
Defaultwert	0

Sub-Index	4
Zugriff	Read / write
Beschreibung	Private build version Nur zur Versionierung, wird nicht verglichen.
Datentyp	UNSIGNED16
Länge	
Wertebereich	UNSIGNED16
Defaultwert	0

Objekt 5020h : Controlword

Objektbeschreibung:

INDEX	5020h
Variablenname	ControlWord
Objekt-Code	VAR
Datentyp	UNSIGNED16
Kategorie	Mandatory

Beschreibung

Zugriff	Read / write
Beschreibung	Stauswort zur Kontrolle der Statemachine
Datentyp	UNSIGNED16
Länge	
Wertebereich	UNSIGNED16
Defaultwert	0

Beschreibung der Daten:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Heratbeat								Absolue						Stop	Start

Bit	Name	Beschreibung
0	Start	Startet ein Geometrieprogramm. Die Nummer (Geo-ID) muß vorab mit Variable 5030 geschrieben werden.
1	Stop	Beendet ein Geometrieprogramm. Am Ende einer zyklischen Datenübertragung wird dieses Bit gesetzt.
3-6	NC	Reserve
7	Absolute	Setzt den Positionsmodus auf absolute Koordinate bezogen auf das Referenz-Koordinatensystem. Immer 1 da relative Daten zurzeit nicht unterstützt werden.
8-14	NC	Reserve
15	Heartbeat	Wechselt bei jedem Telegramm und ermöglicht es fehlende Telegramme zu erkennen (→ Interpolationsfehler).

Objekt 5021h : Statuswort

INDEX	5021h
Variablenname	StatusWord
Objekt-Code	VAR
Datentyp	UNSIGNED16
Kategorie	Mandatory

Beschreibung

Zugriff	Read / write
Beschreibung	Stauswort zur Kontrolle der Statemaschinen
Datentyp	UNSIGNED16
Länge	
Wertebereich	UNSIGNED16
Defaultwert	0

Beschreibung der Daten:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
													Geo.Fehler	Geo.Edne	Geo.Aktiv

Bit	Name	Beschreibung
0	GeoActiv	Ein Geometrieprogramm ist aktiv
1	GeoEnde	Geometrieprogramm beendet. Quittiert das Controlbit Stop bzw. wird im Fehlerfahl gesetzt um den Host zu informieren das die Geometrie beendet wurde
3	Geofehler	Fehler während der Abarbeitung der Geometrie.
3-15	NC	Reserve

Objekt 5030h : Geometrienummer

INDEX	5030h
Variablenname	GeoNum
Objekt-Code	VAR
Datentyp	UNSIGNED16
Kategorie	Mandatory

Beschreibung

Zugriff	Read / write
Beschreibung	Nummer der Geometrie die ausgeführt werden soll (Figur-ID)
Datentyp	UNSIGNED16
Länge	
Wertebereich	UNSIGNED16
Defaultwert	0

Beschreibung der Daten:

GeoNummer	**Beschreibung**
0h	Reserviert
1h – 3E7h	Endsprechend gespeicherte Geometrie aus PFO-Speicher
3E8 - FFFC	Reserve
FFFD	Koordinatenkreuz anzeigen
FFFE	Maximales Bearbeitungsfeld anzeigen
FFFF	Mittelpunkt des Bearbeitungsfeldes (TCP) anzeigen

Objekt 5040h : Translatorische Daten

INDEX	5040h
Variablenname	RobTranslation
Objekt-Code	ARRAY
Datentyp	INTEGER16
Kategorie	Mandatory

Beschreibung

Sub-Index	0
Zugriff	Read / write
Beschreibung	Anzahl Einträge
Datentyp	
Länge	
Wertebereich	
Defaultwert	0

Sub-Index	1
Zugriff	Read / write
Beschreibung	Translation in X-Richtung
Datentyp	INTEGER16
Länge	
Wertebereich	-30000 - 30000
Defaultwert	0

Sub-Index	2
Zugriff	Read / write
Beschreibung	Translation in Y-Richtung
Datentyp	INTEGER16
Länge	
Wertebereich	-30000 - 30000
Defaultwert	0

Sub-Index	3
Zugriff	Read / write
Beschreibung	Translation in Z-Richtung
Datentyp	INTEGER16
Länge	
Wertebereich	-30000 - 30000
Defaultwert	0

Maßeinheit: In Inkrementen von $10^{-PosResolution}$

Objekt 5042h : Rotatorische Daten

INDEX	5042h
Variablenname	RobRotation
Objekt-Code	ARRAY
Datentyp	UNSIGNED16
Kategorie	Mandatory

Beschreibung

Sub-Index	0
Zugriff	Read / write
Beschreibung	Anzahl Einträge
Datentyp	
Länge	
Wertebereich	
Defaultwert	0

Sub-Index	1
Zugriff	Read / write
Beschreibung	A-Winkel-Rotation: Rotation um Z-Achse
Datentyp	INTEGER16
Länge	
Wertebereich	0 – FFFFh endspricht 0 - 2π

Sub-Index	2
Zugriff	Read / write
Beschreibung	B-Winkel-Rotation: Rotation um Y-Achse
Datentyp	INTEGER16
Länge	
Wertebereich	0 – FFFFh endspricht 0 - 2π

Sub-Index	3
Zugriff	Read / write
Beschreibung	C-Winkel-Rotation: Rotation um X-Achse
Datentyp	INTEGER16

Länge	
Wertebereich	0 – FFFFh endspricht 0 - 2π

Objekt 5044h : Skalierungs-Daten

INDEX	5044h
Variablenname	RobScale
Objekt-Code	ARRAY
Datentyp	INTEGER16
Kategorie	Mandatory

Beschreibung

Sub-Index	0
Zugriff	Read / write
Beschreibung	Anzahl Einträge
Datentyp	
Länge	
Wertebereich	
Defaultwert	0

Sub-Index	1
Zugriff	Read / write
Beschreibung	Skalierung in X-Richtung
Datentyp	INTEGER16
Länge	
Wertebereich	-30000 – 30000, /1000 → in 0.001 Schritten von –30.000 – 30.000
Defaultwert	0

Sub-Index	2
Zugriff	Read / write
Beschreibung	Skalierung in Y-Richtung
Datentyp	INTEGER16
Länge	
Wertebereich	-30000 – 30000, /1000 → in 0.001 Schritten von –30.000 – 30.000

Defaultwert	0

Sub-Index	3
Zugriff	Read / write
Beschreibung	Skalierung in Z-Richtung
Datentyp	INTEGER16
Länge	
Wertebereich	-30000 – 30000, /1000 → in 0.001 Schritten von –30.000 – 30.000
Defaultwert	0

Objekt 5060h : 2D-Koordinatentransformation

INDEX	5060h
Variablenname	BV-Trans
Objekt-Code	ARRAY
Datentyp	INTEGER16
Kategorie	Mandatory

Beschreibung

Sub-Index	0
Zugriff	Read / write
Beschreibung	Anzahl Einträge
Datentyp	
Länge	
Wertebereich	
Defaultwert	0

Sub-Index	1
Zugriff	Read / write
Beschreibung	A-Winkel-Rotation: Rotation um Z-Achse
Datentyp	INTEGER16
Länge	
Wertebereich	0 – FFFFh endspricht 0 - 2π

Sub-Index	2
Zugriff	Read / write
Beschreibung	Translation in X-Richtung
Datentyp	INTEGER16
Länge	
Wertebereich	-30000 - 30000

Sub-Index	3
Zugriff	Read / write
Beschreibung	Translation in X-Richtung
Datentyp	INTEGER16
Länge	
Wertebereich	-30000 - 30000

Maßeinheit für translatorische Objekte: In Inkrementen von $10^{-PosResolution}$

Objekt 5070h : Testmode

INDEX	5070h
Variablenname	Demomode
Objekt-Code	ARRAY
Datentyp	UNSIGNED16
Kategorie	Mandatory

Beschreibung

Sub-Index	0
Zugriff	Read / write
Beschreibung	Anzahl Einträge
Datentyp	
Länge	
Wertebereich	
Defaultwert	0

Sub-Index	1
Zugriff	Read / write
Beschreibung	Statuswort für Teachmodus Bit 0 : Teachmode ein („1“) / Teachmode aus („0“) Bit 1- Bit15: RESEVE
Datentyp	INTEGER16
Länge	
Wertebereich	INTEGER16

Sub-Index	2
Zugriff	Read / write
Beschreibung	LoopCounter: Definiert die Anzahl Wiederholungen der Teachgeometrie: 1-254: Anzahl Wiederhohlungen 255: Endlos wiederholen bis Abruch durch Wiederholungsanzahl 0 erfolgt 0: Abbruch von endloser Wiederholung
Datentyp	UNSIGNED8
Länge	
Wertebereich	UNSIGNED8

Sub-Index	3
Zugriff	Read / write
Beschreibung	Teach-Geometrienummer
Datentyp	UNSIGNED16
Länge	
Wertebereich	UNSIGNED16

Beschreibung der Daten:

GeoNummer	**Beschreibung**
0h	Reserviert
1h – 3E7h	Endsprechend gespeicherte Geometrie aus PFO-Speicher
3E8 - FFFC	Reserve
FFFD	Koordinatenkreuz anzeigen
FFFE	Maximales Bearbeitungsfeld anzeigen
FFFF	Mittelpunkt des Bearbeitungsfeldes (TCP) anzeigen

Objekt 5080h : Tooldaten

INDEX	5080h
Variablenname	TCP
Objekt-Code	ARRAY
Datentyp	UNSIGNED32
Kategorie	Mandatory

Beschreibung

Sub-Index	0
Zugriff	Read / write
Beschreibung	Anzahl Einträge
Datentyp	
Länge	
Wertebereich	
Defaultwert	0

Sub-Index	1
Zugriff	Read / write
Beschreibung	Tooldaten X-Richtung
Datentyp	INTEGER32
Länge	
Wertebereich	INTEGER32 (1µm / Digit)

Sub-Index	2
Zugriff	Read / write
Beschreibung	Tooldaten Y-Richtung
Datentyp	INTEGER32 (1µm / Digit)
Länge	
Wertebereich	INTEGER32

Sub-Index	3
Zugriff	Read / write
Beschreibung	Tooldaten Z-Richtung
Datentyp	INTEGER32
Länge	
Wertebereich	INTEGER32 (1µm / Digit)

Sub-Index	4
Zugriff	Read / write
Beschreibung	Tooldrehung um Z-Achse (A-Winkel)
Datentyp	INTEGER32
Länge	
Wertebereich	INTEGER32 (0.01° / Digit)

Sub-Index	5
Zugriff	Read / write
Beschreibung	Tooldrehung um Y-Achse (B-Winkel)
Datentyp	INTEGER32
Länge	
Wertebereich	INTEGER32 (0.01° / Digit)

Sub-Index	6
Zugriff	Read / write
Beschreibung	Tooldrehung um X-Achse (C-Winkel)
Datentyp	INTEGER32
Länge	
Wertebereich	INTEGER32 (0.01° / Digit)

„Plug-And-Play" spezifiziert als Kommunikationsprofil bzw. Anwendungsprofil (Lenze)

1 Zielstellung

Ziel ist die Definition von Geräte- und Anwendungsprofilen, die eine einfache Konfiguration, Inbetriebnahme und Wartung zukunftsweisender Antriebs- und Steuerungskonzepte gewährleisten (PAPAS-PnP-Spezifikation).

1.1 Abkürzungen und Begriffe

Die Ausführungen in diesem Dokument beziehen sich nicht auf ein bestimmtes Kommunikationssystem. Demzufolge kann nicht auf eine spezielle Begriffswelt zurückgegriffen werden. Da in den verschiedenen Spezifikationen unterschiedliche Begriffe synonym verwendet werden oder ein Begriff unterschiedliche Sachverhalte bezeichnet ist es erforderlich, die Kernbegriffe für dieses Dokument zu definieren.

Begriff	Definition
Anwendungsprofil	Spezifikation von Objekten, Diensten, Verhaltensweisen und Kodierungen für einen Anwendungsbereich (z. B. Fertigungsroboter), um interoperable Kommunikation zwischen den auf unterschiedlichen Geräten laufenden Teilanwendungen dieses Anwendungsbereiches zu gewährleisten
Anwendungsprogramm	Programm, das eine Automatisierungsaufgabe auf einem Knoten des Kommunikationsnetzwerkes bearbeitet und zur Kommunikation mit andere Knoten Dienste des Anwendungsprofils, des Geräteprofils, des Kommunikationsprofils bzw. der Anwendungsschicht des Kommunikationssystems nutzt
Dienst	Schnittstellenelement, das Kommunikationsfunktionalität und -argumente festlegt
Entfernte Anwendung	Teilanwendung, die auf einem anderen als den betrachteten Knoten eines Kommunikationsnetzwerkes bearbeitet wird Im betrachteten Knoten wird ein Treiber genutzt, um mit der entfernten Anwendung zu kommunizieren.
Gerät	Allgemeine Bezeichnung für einen Slave-Node im Kommunikationsnetzwerk aus Anwendungssicht
Geräteanwendungsidentifikation	Identifikation einer Teilanwendung der verteilten Anwendung auf einem Slave-Node Jede Teilanwendung ist in der verteilten Anwendung durch die Geräteanwendungsidentifikation eindeutig identifizierbar.
Geräteidentifikation	Identifikation eines Slave-Nodes Die Beziehung zwischen Slave-Node und Geräteidentifikation ist eineindeutig.
Geräteklasse	Zusammenfassung gleichartiger Geräte wie Sensoren, Umrichter, Antriebe, Kamerasysteme oder IO-Baugruppen usw. Eigenschaften und Verhalten einer Geräteklasse werden in einem Root-Device-Profile nach **Fehler! Verweisquelle konnte nicht gefunden werden.** beschrieben. Das Root-Device-Profile wird durch Arbeitsgruppen in Standardisierungsgremien oder Nutzerorganisationen erstellt. Ein Gerät kann durch die Geräteklassenidentifikation eindeutig einer Geräteklasse zugeordnet werden.
Geräteklassenidentifikation	Identifikation der Klasse der ein Gerät zugehört, wie Sensor, Umrichter, Antrieb, Kamerasystem oder IO-Baugruppe

Begriff	Definition
Gerätekonfigurationsidentifikation	Identifikation einer definierten Geräte-Konfiguration (Geräte-Parameterwerte) für einen Slave-Node Es kann mehrere Slave-Nodes mit gleicher Geräte-Konfiguration in der verteilten Anwendung geben.
Geräte-Parametersatz	Sammlung der im Gerät implementierten Objekte eines Geräteprofils und deren mögliche Wertebereiche
Geräte-Parameterwerte	Sammlung der im Gerät implementierten Objekte eines Geräteprofils und deren konfigurierten Werte
Geräteprofil	Spezifikation von Objekten, Diensten, Verhaltensweisen und Kodierungen für eine Klasse von Geräten (z. B. Antriebe), um interoperable Kommunikation zwischen unterschiedlichen Geräten dieser Klasse zu gewährleisten
Gerätetyp	Zusammenfassung gleichartiger Geräte ein und der selben Geräteklasse. Eigenschaften und Verhalten eines Gerätetyps werden in einem Manufacturer-Device-Profile nach **Fehler! Verweisquelle konnte nicht gefunden werden.** beschrieben. Das Manufacturer-Device-Profile wird durch Arbeitsgruppen in Nutzerorganisationen oder durch Hersteller erstellt. Ein Gerät kann durch die Gerätetypidentifikation eindeutig einem Gerätetyp zugeordnet werden.
Gerätetypidentifikation	Identifikation des Typs dem ein Gerät zugehört Im Netzwerk können mehrere Geräte eines Typs vorkommen.
Kommunikationsprofil	Zusammenstellung der Objekte, Dienste, Verhaltensweisen und Kodierungen aus den Spezifikationen gemäß ISO/OSI-Schichtenmodell (soweit sie für das Kommunikationssystem spezifiziert sind) zur Gewährleistung der Interoperabilität zwischen Geräten der Automatisierungstechnik
Konfiguration	Gesamtheit der Kommunikations-, Geräte- und Anwendungs-Parameterwerte
Lokale Anwendung	Teilanwendung, die auf dem betrachteten Knoten eines Kommunikationsnetzwerkes bearbeitet wird
Master-Node	Knoten im Kommunikationsnetzwerk, der das Geräte- und Anwendungsmanagement beinhaltet
Objekt	Der Begriff Objekt wird im Sinne von CANopen verwendet. Es handelt sich um ein Attribut, dass die Kommunikation, das Gerät oder die Anwendung näher charakterisiert. Implementierungstechnisch handelt es sich um eine einfache oder strukturierte Variable.
PAPAS	Profil für Plug-And-Play Antriebs- und Steuerungskonzepte für die Produktion von morgen
PAPAS-PnP-Spezifikation	Kommunikationsobjekte (Attribute, Dienste) und Kommunikationsprotokolle (Kodierungen, Verhalten) zur Realisierung der Plug-and-Play-Aufgaben, die bei der Spezifikation der Geräte- und Anwendungsprofile zu berücksichtigen sind
Plug-And-Play (PnP)	Konzept zur aufwandsminimierten Inbetriebnahme und Wartung von IT-Systemen
Protokoll	Beschreibung des Ablaufes der Kommunikation
Slave-Node	Knoten im Netzwerk, dessen Geräte- und Anwendungseinstellungen und -verhalten von einem Master-Node gemanagt wird

Begriff	Definition
Treiber	Anwendungsprogramm im Master-Node, das Anwendungsfunktionen zur Verfügung stellt, um eine entfernte Anwendung nutzen zu können
Verteilte Anwendung	Automatisierungsaufgabe, die auf Knoten eines Kommunikationsnetzwerkes verteilt ist und kooperativ bearbeitet wird

1.2 Referenzen

- Ethernet Powerlink V2.0, Communication Profile Specification, Draft Standard Proposal, Version 0.1.0
- VERBUNDPROJEKT Plug-And-Play Antriebs- und Steuerungskonzepte für die Produktion von Morgen (PAPAS), RAHMENKONZEPT
- Protokoll der Sitzung des Arbeitskreises Kommunikation Hameln, Fa. Lenze/ 08.07.2004
- Präsentation "Begriffsdefinitionen und Anforderungen an die Kommunikation", KUKA Controls GmbH – Rainer Bischoff
- KUKA ControlWeb, Pflichtenheft zum Projektteil „Plug&Work“, Bearbeiter: Eckart Wiesenhütter, Stand: 05.05.2003
- IEC 62390: Common Automation Device Profile Guideline

2 Anforderungen an die PAPAS-PnP-Spezifikation

2.1 Anwendungsbereich

Zukunftsweisende Antriebs- und Steuerungskonzepte sind die Voraussetzung um verteilte Anwendungen in der Fertigungstechnik mit hoher Effizienz zu realisieren. Sie beinhalten Geräte mit unterschiedlicher Informationsverarbeitungskapazität, die Daten über ein serielles Kommunikationssystem austauschen. Zu diesen Geräten gehören u. a.[4]:

- einfache Sensoren (z. B. zur Positionserfassung),
- Powermodule,
- Umrichter,
- Antriebe,
- Kamerasysteme,
- Speicherprogrammierbare Steuerungen (SPS) und
- Standard-Personal-Computer (PC z. B. für Steuerungs- und Regelungsanwendungen).

An die Kommunikation werden hohe Anforderungen bzgl. Zykluszeit (125µs) und Jitter (<1µs) aber auch bzgl. der Anschaltkosten gestellt.
Die PAPAS-PnP-Spezifikation muss diesen Voraussetzungen gerecht werden. Die Spezifikation ist universell beschrieben, so dass sie auf verschiedene Kommunikationssysteme abgebildet werden kann [2].

2.2 Plug-and-Play-Use-Cases

Die PAPAS-PnP-Spezifikation soll die Use-Cases Systemanlauf, Gerätetausch und Diagnose berücksichtigen.
Beim Systemanlauf kommt es darauf an, die Anwendung mit möglichst geringem Konfigurationsaufwand in Betrieb zu setzen. Unabhängig vom Hersteller sollen Geräte unterschiedlicher Geräteklassen untereineinender Kommunikationsbeziehungen aufbauen und die für die Anwendung erforderliche Kommunikation möglichst automatisch aufnehmen. Es sollen dadurch Fehlkonfigurierungen vermieden und Inbetriebnahmezeiten entscheidend verkürzt werden.
Beim Gerätetausch kommt es darauf an, defekte Geräte möglichst schnell durch Ersatzgeräte auszutauschen. Das Ersatzgerät sollte nicht vom gleichen Hersteller sein

müssen. Es sollte jedes Gerät des gleichen Gerätetyps einsetzbar sein können. Das Gerät sollte möglichst ohne Konfiguration und ohne Programmierung die Aufgabe des defekten Gerätes übernehmen können. Damit können Stillstandszeiten verkürzt und somit die Verfügbarkeit der Anlage erhöht werden.
Bei der Diagnose geht es darum, entsprechende Geräte in das System integrieren zu können, ohne den Regelbetrieb zu beeinflussen. Ohne Neuanlauf muss es den Diagnosegeräten möglich sein, alle erforderlichen Systeminformationen zu erfassen. Damit können z. B. Wartungsprozesse unterstützt werden, ohne die Fertigung zu unterbrechen.
Im folgenden Absatz werden die genannten Use-Cases anhand einzelner Szenarien weiter untersetzt.

2.3 Plug-and-Play-Szenarien

Im Folgenden werden einige Szenarien beschrieben, die durch die PAPAS-PnP-Spezifikation abgedeckt werden sollen. Die Szenarien beziehen sich auf einen Roboter, an den Geräten wie

- zusätzliche Antriebe,
- Greifer,
- Kameras,
- Kraft-Momenten-Sensoren,
- Programmierbare Fokussieroptiken und
- Regler (als Teil eines modularen Steuerungskonzeptes)

angeschlossen werden können [4]. Nach einer kurzen Darstellung des Szenariums wird auf die Funktionen eingegangen, die die PAPAS-PnP-Spezifikation berücksichtigen muss.

Szenario 1: Lastdatenermittlung

Die dynamischen Daten (Masse, Schwerpunkt, Trägheitstensor) eines am Flansch des Roboters befestigten Gerätes (z. B. Greifer) werden durch entsprechende Bewegung des Gerätes ermittelt. Die ermittelten Daten sollen im Geräte gespeichert werden.
Der Anschluss eines entsprechenden Gerätes ist dem System anzuzeigen, einschließlich der Information zur Geräteklasse, zum Gerätetyp bzw. zu den relevanten Geräteeigenschaften (Geräte-Parametersatz). Die Mechanismen zum Abspeichern der Lastdaten im Gerät sowie zur Bereitstellung der Daten durch das Gerät bei einem Wiederanlauf sind zu spezifizieren.

Szenario 2: Anschluss von Antrieben oder von Programmierbarer Fokussieroptik (Vorkonfigurierte Geräte)

Zusätzliche Antriebe (evtl. mit Getriebe) für einen Hubtisch oder eine Programmierbare Fokussieroptik (PFO) für eine Laserschweiß-Anwendung sind zu konfigurieren. Diese Geräte sind durch eine hohe Flexibilität gekennzeichnet und erfordern dadurch die Einstellung von einer Vielzahl von Parametern. Für die Konfiguration der Geräte wird eine Engineering-Software auf einem externen PC genutzt. Bei der Verbindung der Geräte mit der Robotersteuerung wird deren Konfiguration an die Steuerung übertragen.
Die Konfiguration von Geräten, durch ein Engineering-Tool sowie die dauerhafte Speicherung der Daten muss gewährleistet werden. Geräteklasse, Gerätetyp und die konfigurierte Anwendung müssen identifizierbar sein.

Szenario 3: Anschluss von Kameras oder von Kraft-Momenten-Sensoren (Geräte mit Funktionsbausteinen)

Zusätzliche Geräte für Bild- oder Kraft-Momenten-basierte Regelungskreise sollen an das System angeschlossen und konfiguriert werden. Diese Geräte sind durch Funktionsbausteine gekennzeichnet, die zur Realisierung der Anwendung nicht in den Geräten, sondern in der Steuerung ablaufen. Je nach Gerät werden die Daten in den Geräten mehr oder weniger vorverarbeitet. Die Inbetriebnahme der Geräte beinhaltet neben der Konfiguration der Geräte auch die Auswahl der passenden Funktionalität, die in der Steuerung ablaufen soll, und deren Konfiguration.

Neben der Identifikation des Gerätetyps sind Informationen über die externen Funktionsbausteine (Treiber) erforderlich. Nach Anschluss der Geräte ist die Treiberauswahl und die konsistente Konfiguration von Gerät und Treiber zu unterstützen.

Szenario 4: Gerätetausch

Ein Gerät kann aufgrund eines Defektes durch ein anderes (nicht zwangsläufig identisches) ersetzt werden. Die Steuerung muss erkennen, ob das Ersatzgerät in der Lage ist, die Funktionalität des zuvor angeschlossenen Gerätes zu übernehmen. Das Ersatzgerät muss entsprechend der Funktion in der Anwendung konfiguriert werden.
Der Ausfall eines Gerätes muss angezeigt werden, wobei Informationen über Gerätetyp und Anwendungsfunktion bereitzustellen sind. Der Anschluss des neuen Gerätes ist zu erkennen und der Gerätetyp bzw. die relevanten Parameter zu identifizieren. Die Übertragung der Konfiguration von der Steuerung zum Gerät ist zu unterstützen.

Szenario 5: Greiferwechsel

Der Roboter nutzt Greifer mit unterschiedlichen dynamischen Daten und unterschiedlichen Tool-Center-Points (TCP). Ein Gegenstand soll durch verschiedene (zufällig angeflanschte) Greifer adäquat gegriffen werden können.
Die Robotersteuerung ist hierfür beim Anschluss des Greifers über die entsprechenden Daten zu informieren. Die automatische Konfiguration der Robotersteuerung ist zu unterstützen.

Szenario 6: Alternative Geräte

An den Roboterarm werden völlig verschiedene Werkzeuge für unterschiedliche Bearbeitungsaufgaben angebaut.
Beim Anschluss des Gerätes sind Informationen zur Anwendung und zum Gerätetyp zwischen der Robotersteuerung und dem Werkzeug auszutauschen.

Szenario 7: Diagnose

Zur Geräte- und Systemdiagnose kann ein Standard-PC (z. B. Laptop) mit dem System verbunden werden. Es kann die Kommunikation verfolgt, Konfigurationsdaten geändert und sogar Geräte gesteuert werden.
Dem Diagnosetool müssen alle Informationen über Gerätekonfigurationen sowie über Geräte- und Systemzustände zugänglich gemacht werden. Weiterhin muss das Diagnosetool über Informationen zu den relevanten Geräte- und Anwendungsprofilen verfügen.

Szenario 8: Reglerauslegung für Roboter mit Rapid-Prototyping

Für die einfachere, schnellere und weniger fehlerträchtige Entwicklung neuer, aufwendigerer Reglerstrukturen wird ein Rapid-Prototyping realisiert, wobei die Algorithmen statt auf einem Digital-Signal-Processor (DSP) auf eine Standard-PC ablaufen. Zu diesem Zwecke wird der PC, auf dem auch die Algorithmen zur Steuerung des Testablaufes abgearbeitet werden, an das System angeschlossen.
Der Rapid-Prototyping-PC wird als solcher im System erkannt und kann Aufgaben des Anwendungsmasters, einschließlich der Netzwerksmanagementaufgaben, übernehmen.

Szenario 9: Anschluss von nichtprojektierten Anwendungen

Um das System für zukünftige Erweiterungen offen zu halten sollen auch Anwendungen in der PAPAS-PnP-Spezifikation berücksichtigt werden, die bei der Projektierung noch nicht bekannt sind.
Das Hinzufügen entsprechender Geräte ist zu erkennen, Anwendung und Parameterwerte zu identifizieren. Die Einordnung der Anwendung muss durch Nutzerinteraktion erfolgen.

3 Plug-and-Play-Varianten

Die Beschreibung der Szenarien lässt verschiedene Plug-and-Play-Varianten erkennen, die im folgenden zusammengefasst werden.

3.1 Cold-Plug-and-Play

Die Geräte werden zunächst verbunden und anschließend die Spannung nacheinander oder insgesamt zugeschaltet. Die Geräte können konfiguriert sein oder während des Systemanlaufs durch einen Configuration Manager konfiguriert werden. Der Systemanlauf kann vollautomatisch erfolgen oder für einzelne Geräte eine Nutzerinteraktion erfordern.

3.2 Hot-Plug-and-Play

Ein Gerät wird zu einem laufenden System hinzugefügt. Die Spannung kann vor oder nach dem Hinzufügen zugeschaltet werden. Das Gerät ist konfiguriert oder wird während der Anlaufphase konfiguriert. Die Anlaufphase kann vollautomatisch erfolgen oder eine Nutzerinteraktion erfordern. In keinem Fall darf die laufende Anwendung in ihrem betriebsgemäßen Ablauf beeinflusst werden. Zum betriebsgemäßen Ablauf kann ein Notprogramm vor dem Hinzufügen des Gerätes gehören.

3.3 Coordinated-Plug-and-Play

Ein Gerät wird zu einem laufenden System in definierten Zuständen hinzugefügt oder entfernt. Die Spannung kann vor oder nach dem Hinzufügen zugeschaltet werden. Das Gerät ist konfiguriert oder wird während der Anlaufphase konfiguriert. Die Anlaufphase kann vollautomatisch erfolgen oder eine Nutzerinteraktion erfordern. Vor dem Entfernen ist dem System durch eine Nutzerinteraktion oder programmgesteuert das bevorstehende Entfernen mitzuteilen und somit ein definierter Zustand einzunehmen. In keinem Fall darf die laufende Anwendung in ihrem betriebsgemäßen Ablauf beeinflusst werden. Zum betriebsgemäßen Ablauf kann ein Notprogramm vor dem Hinzufügen oder nach dem Entfernen eines Gerätes gehören.

3.4 Plug-and-Play-Stufen

Die Beschreibung der Szenarien und der Plug-and-Play-Varianten lässt verschiedene Plug-and-Play-Stufen erkennen, die im folgenden zusammengefasst werden.

3.5 Vollständiges Plug-and-Play

Wenn ein Gerät nach dem Anlauf kommunikationsfähig ist, wird der Gerätetyp vom System erkannt und das Gerät wenn erforderlich automatisch konfiguriert. Dazu gehört auch die automatische Auswahl, Konfiguration und Einbindung eines Treibers im Anwendungsmaster. Ist das Gerät dem System bereits bekannt, kann die Konfiguration von Gerät und Treiber entfallen, da die Daten nichtflüchtig gespeichert sind. Das Gerät steht der Anwendung zur Verfügung.

3.6 Halbautomatisches Plug-and-Play

Wenn ein Gerät nach dem Anlauf kommunikationsfähig ist, wird der Gerätetyp vom System erkannt und das Gerät wenn erforderlich automatisch konfiguriert. Die Auswahl eines Treibers im Anwendungsmaster erfordert eine Nutzerinteraktion. Die Konfiguration und Einbindung des Treibers wird automatisch ausgeführt.
Konfigurierbares Plug-and-Play
Wenn ein Gerät nach dem Anlauf kommunikationsfähig ist, wird der Gerätetyp vom System erkannt. Die Konfiguration des Gerätes bzw. die Auswahl, Konfiguration und Einbindung eines Treibers im Anwendungsmaster erfordert eine Nutzerinteraktion.

4 Plug-and-Play-Ziele

4.1 Interoperabilität

Die PAPAS-PnP-Spezifikation soll Interoperabilität zwischen Geräten unterschiedlicher Geräteklassen für die Plug-and-Play-Funktionalität (Konfiguration, Austausch, Diagnose) sichern. Das heißt, es gibt eine Durchschnittsmenge der Plug-and-Play-Funktionen, die alle Geräte implementieren müssen. Je nach Umfang der Implementierung dieser Plug-and-Play-Funktionen werden die verschiedenen Plug-and-Play-Varianten und Plug-and-Play-Stufen umgesetzt. Der Umfang der Implementierung wird in erster Linie durch das Gerät-Design vorgegeben.

4.2 Interchangebility

Die PAPAS-PnP-Spezifikation soll Interchangebility zwischen Geräten eines Gerätetyps sichern. Das heißt, die Geräte haben ein und dasselbe Geräteprofil implementiert. Das Geräte-Design gibt vor, in welchem Umfang die Plug-and-Play-Funktionen umgesetzt werden können und welche Plug-and-Play-Varianten und Plug-and-Play-Stufen realisierbar sind.

4.3 Geräte-Architektur

In diesem Absatz werden die wesentlichen Bestandteile beschrieben, die ein Plug-and-Play-fähiges Geräte haben muss bzw. haben kann (siehe Abbildung 1). Grundvoraussetzung ist ein Kommunikationsmodul, mit dessen Hilfe Produktivdaten (meist zyklisch) und Parameterdaten (meist azyklisch) zwischen Geräten ausgetauscht werden können. Das Kommunikationsmodul muss ein Interface zur Verfügung stellen, das von einem Geräteprofil bzw. von einem Anwendungsprofil genutzt werden kann. In das Geräteprofil sind gerätespezifische Funktionen zur Konfiguration, zum Gerätetausch und zur Diagnose aufzunehmen. In das Anwendungsprofil sind anwendungsspezifische Funktionen zur Konfiguration, zum Gerätetausch und zur Diagnose aufzunehmen. In der PAPAS-PnP-Spezifikation ist festzuhalten, welche Funktionen optional und welche Pflicht sind. Einen wesentlichen Einfluss auf die Plug-and-Play-Funktionalität hat die Möglichkeit Daten nicht-flüchtig im Gerät abzulegen und Funktionsmodule im Gerät ändern zu können. Diese Möglichkeiten sind durch das Gerät-Design vorgegeben und werden nicht durch die PAPAS-PnP-Spezifikation definiert.

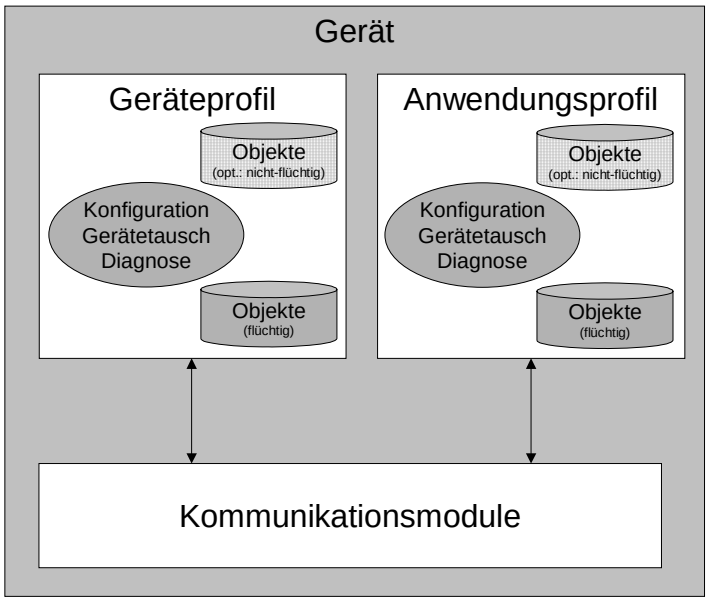


Abbildung 1: Prinzipielle Architektur eines Gerätes entsprechend der PAPAS-PnP-Spezifikation

5 Plug-and-Play und Determinismus

5.1 Systeme mit hohen Determinismusanforderungen

Systeme mit hohen Determinismusanforderungen verlangen zwingend die Festlegung aller betriebsbedingt am isochronen Datenaustausch beteiligten Slave-Nodes während der Projektierung. Diese projektierten Slave-Nodes könnten, ohne Beeinflussung des Determinismus, hinzugefügt und entfernt werden.
Slave-Nodes die ausschließlich am asynchronem Datenaustausch beteiligt sind, beeinflussen den Determinismus nicht und können jederzeit hinzugefügt und entfernt werden.

5.2 Systeme mit geringeren Determinismusanforderungen

Bei Systemen mit geringeren Determinismusanforderungen kann eine Variation der Anzahl der betriebsbedingt am isochronen Datenaustausch beteiligten Slave-Nodes akzeptiert werden. Es können also jederzeit Slave-Nodes hinzugefügt und entfernt werden, unabhängig davon ob sie am isochronen oder asynchronen Datenaustausch teilnehmen. Die Slave-Nodes müssen nicht unbedingt schon bei der Projektierung berücksichtigt werden. In diesem Fall wird beim Hinzufügen eine Nutzerinteraktion erforderlich sein.

6 Beschreibung des PAPAS-PnP anhand von Szenarien

6.1 Szenario 1: Lastdatenermittlung

6.1.1 Cold-Plug-and-Play

Bei diesem Szenario ermittelt die Anwendung relevante Anwendungs-Parameterwerte die einem Slave-Node zugeordnet werden. Die Anwendungs-Parameterwerte werden entweder im Slave-Node oder im Master-Node nicht-flüchtig gespeichert.
Folgender Ablauf ist bei einem Erstanlauf notwendig:

- Identifizierung des Gerätetyps
- Identifizierung des Gerätekonfigurationszustands (Lastdaten noch nicht ermittelt)
- Auswahl des Lastdatenermittlungsalgorithmus
- Abarbeitung des Lastdatenermittlungsalgorithmus und Erfassung der Anwendungs-Parameterwerte im Master-Node
- Nicht-flüchtige Speicherung der Anwendungs-Parameterwerte im Slave-Node oder im Master-Node
- Änderung des Gerätezustands

Übermittlung der Anwendungs-Parameterwerte an die Anwendung Folgender Ablauf ist bei einem Wiederanlauf notwendig:

- Identifizierung des Gerätetyps
- Identifizierung des Gerätekonfigurationszustands (Lastdaten bereits ermittelt)
- Lesen der Anwendungs-Parameterwerte aus dem Slave-Node oder dem Master-Node
- Übermittlung der Anwendungs-Parameterwerte an die Anwendung

Ein vollständiges Plug-and-Play beim Erstanlauf ist möglich, wenn dem Gerätetyp (z. B. einem Greifer) eindeutig ein Lastdatenermittlungsalgorithmus (Treiber) zugeordnet werden kann. Beim Wiederanlauf ist ein vollständiges Plug-and-Play möglich.

6.1.2 Hot-Plug-and-Play

In diesem Szenario ist das unkoordinierte Hinzufügen oder Entfernen eines Gerätes für den Regelbetrieb nicht sinnvoll.

6.1.3 Coordinated-Plug-and-Play

Das koordinierte Hinzufügen oder Entfernen eines Gerätes im Regelbetrieb ist für dieses Szenario sinnvoll. Das Gerät wird angesprochen, wenn es entsprechend des Programmablaufs dem Netzwerk hinzugefügt wurde.

Anschließend ist das Verhalten identisch mit dem beim Cold-Plug-and-Play. Während der Abarbeitung des Lastdatenermittlungsalgorithmus sind Prozessdaten innerhalb der isochronen Periode zu übertragen.

6.2 Szenario 2: Anschluss von Antrieben oder von Programmierbarer Fokussieroptik (Vorkonfigurierte Geräte)

6.2.1 Cold-Plug-and-Play

Bei diesem Szenario ermittelt die Anwendung auf dem Master-Node beim Systemanlauf den Gerätetyp und die konfigurierte Anwendung aus einem oder mehren Slave-Nodes. Der Ablauf ist wie folgt:

- Identifizierung des Gerätetyps
- Identifizierung der Geräteanwendung
- Auswahl des Treibers
- Übertragung der Anwendungsdaten vom Slave-Node an den Master-Node
- Suche nach weiteren Geräten

Ein vollständiges Plug-and-Play ist möglich, wenn dem Gerätetyp (z. B. einem Antrieb) eindeutig ein Treiber zugeordnet werden kann.

6.2.2 Hot-Plug-and-Play

Das spontane Hinzufügen und Entfernen eines z. B. PFOs im Regelbetrieb ist nicht ausgeschlossen. Es müssen Dienste und Protokolle spezifiziert werden, mit denen feststellbar ist, dass ein Gerät hinzugefügt oder entfernt wurde. Außerdem sind die Prozessdaten des Slave-Nodes dem Datenaustausch der isochronen Periode hinzuzufügen.

6.2.3 Coordinated-Plug-and-Play

Für das koordinierte Hinzufügen oder Entfernen eines Gerätes im Regelbetrieb ist es nicht erforderlich festzustellen, ob ein Gerät unerwartet hinzugefügt oder entfernt wurde, wie beim Hot-Plug-and-Play. Allerdings sind wiederum die Prozessdaten des Slave-Nodes dem Datenaustausch der isochronen Periode hinzuzufügen.

6.3 Szenario 3: Anschluss von Kameras oder von Kraft-Momenten-Sensoren (Geräte mit Funktionsbausteinen)

6.3.1 Cold-Plug-and-Play

Nach dem Systemanlauf sind die Geräte (Slave-Nodes) noch nicht einsatzfähig. Aufgrund der vielfältigen Funktionen, die im Master-Node ablaufen ist beim Erstanlauf nur ein konfigurierbares Plug-and-Play sinnvoll.
Folgender Ablauf ist bei einem Erstanlauf notwendig:

- Identifizierung des Gerätetyps
- Identifizierung des Gerätes
- Identifizierung des Gerätezustands (bisher nicht konfiguriert)
- Auswahl der Funktionalität durch Nutzerinteraktion
- Konfiguration durch Nutzerinteraktion
- Nichtflüchtige Speicherung der Anwendungsdaten im Slave-Node oder im Master-Node
- Änderung des Gerätezustands:

- Suche nach weiteren Geräten Folgender Ablauf ist bei einem Wiederanlauf notwendig:

- Identifizierung des Gerätetyps
- Identifizierung des Gerätezustands (Konfiguration bereits vorhanden)
- Identifizierung der Anwendung
- Konfiguration der Funktionsbausteine im Master-Node und Konfiguration der Parameter im Slave-Node und im Master-Node

Beim Wiederanlauf kann ein Vollständiges Plug-and-Play realisiert werden.

6.3.2 Hot-Plug-and-Play

Das spontane Hinzufügen und Entfernen von einem Gerät (z. B. Kamera) im Regelbetrieb ist nicht ausgeschlossen. Die Geräte sind bei Erstanschluss nicht sofort einsatzfähig. Eine Nutzerinteraktion ist erforderlich. Es müssen Dienste und Protokolle spezifiziert werden, mit denen feststellbar ist, dass ein Gerät hinzugefügt oder entfernt wurde. Außerdem sind die Prozessdaten des Slave-Nodes dem Datenaustausch der isochronen Periode hinzuzufügen.

6.3.3 Coordinated-Plug-and-Play

Das koordinierte Hinzufügen oder Entfernen eines Gerätes im Regelbetrieb ist für dieses Szenario sinnvoll. Das Gerät wird angesprochen, wenn es entsprechend des Programmablaufs dem Netzwerk hinzugefügt wurde. Anschließend ist das Verhalten identisch mit dem beim Cold-Plug-and-Play. Allerdings sind die Prozessdaten des Slave-Nodes dem Datenaustausch der isochronen Periode hinzuzufügen.

6.4 Szenario 4: Gerätetausch

6.4.1 Cold-Plug-and-Play

Während des Systemanlaufes wird festgestellt, dass ein oder mehrere Slave-Nodes mit einer erwarteten Anwendung nicht verfügbar ist.
Folgender Ablauf ist bei einem Erstanlauf notwendig:

- Feststellen eines Gerätefehlers
- Identifizierung der fehlenden Anwendungsfunktionalität und des Gerätetyps
- Unterstützung des Gerätewechsels durch Nutzerinteraktion
- Feststellen des Vorhandenseins eines Austauschgerätes
- Identifikation des Gerätetyps
- Prüfen ob das Gerät die geforderte Funktionalität übernehmen kann
- Treiberauswahl und Übertragung der Anwendungsdaten auf den Slave-Node ggf. durch Nutzerinteraktion
- Suche nach weiteren defekten Geräten

Ein vollständiges Plug-and-Play bei Einbau eines anderen Gerätetyps erfordert eine entsprechend umfangreiche Datenbank, auf die der Master-Node zugreifen kann, um Treiber und Konfiguration für den anderen Gerätetyp ermitteln zu können.

6.4.2 Hot-Plug-and-Play

Der Ausfall eines Gerätes muss unabhängig von der Anwendung festgestellt werden. Ist der Regelbetrieb vom Ausfall nicht betroffen, ist das unkoordinierte Hinzufügen möglich. Beim Hinzufügen sind parallel zum Regelbetrieb die bei Cold-Plug-and-Play beschriebenen Aktionen durchzuführen. Ggf. ändern sich z. B. Typ, Anzahl bzw. Länge der Prozessdaten des Slave-Nodes die in der isochronen Periode übertragen werden.

6.4.3 Coordinated-Plug-and-Play

Während der Ausfall eines Gerätes naturgemäß unkoordiniert auftritt kann das Hinzufügen auch koordiniert erfolgen. Das trifft insbesondere zu, wenn das System nach dem Ausfall in einen sicherer Zustand überführt wurde. Ggf. ändern sich z. B. Typ, Anzahl bzw. Länge der Prozessdaten des Slave-Nodes die in der isochronen Periode übertragen werden.
Szenario 5: Greiferwechsel

6.4.4 Cold-Plug-and-Play

Es ist festzustellen welcher Greifer angeflanscht ist. Es erfolgt die Zuordnung eines Treibers und der Geräte-Parameterwerte im Master-Node. Ggf. sind die Geräte-Parameterwerte vom Slave-Node zu übertragen. Typ, Anzahl bzw. Länge der Prozessdaten der Slave-Nodes (Greifer) können abweichen.

6.4.5 Hot-Plug-and-Play

Ein unkoordinierter Greiferwechsel ist nicht sinnvoll.

6.4.6 Coordinated-Plug-and-Play

Beim Coordinated-Plug-and-Play wird ein Greifer mit seinem Treiber und den Geräte-Parameterwerte abgemeldet und der andere angemeldet. Damit liegen immer alle Informationen vor, um einen Gegenstand mit unterschiedlichen Greifern adäquat greifen zu können. Typ, Anzahl bzw. Länge der Prozessdaten der Slave-Nodes (Greifer) können abweichen und müssen somit für die Übertragung in der isochronen Periode geändert werden.

6.5 Szenario 5: Alternative Geräte

6.5.1 Cold-Plug-and-Play

Es ist festzustellen welches Werkzeug angeflanscht ist. Die Anwendung ist zu identifizieren und automatisch oder mittels Nutzerinteraktion in die verteilte Anwendung einzuordnen. Der Treiber und die Geräte-Parameterwerte werden ermittelt. Die Länge der Prozessdaten der Slave-Nodes (Werkzeuge) sollen identisch sein, um den isochronen Datenaustausch nicht zu beeinflussen.

6.5.2 Hot-Plug-and-Play

In diesem Szenario ist das unkoordinierte Hinzufügen oder Entfernen eines alternativen Gerätes für den Regelbetrieb nicht sinnvoll.

6.5.3 Coordinated-Plug-and-Play

Beim Coordinated-Plug-and-Play wird ein Werkzeug mit seiner Anwendung und ggf. mit Treiber und Geräte-Parameterwerte abgemeldet und ein anderes angemeldet. Die Länge der Prozessdaten der Slave-Nodes (Werkzeuge) sollten identisch sein.

6.6 Szenario 7: Diagnose

In der ersten Version der PAPAS-PnP-Spezifikation wird ausgeschlossen, dass das Diagnose-Gerät Funktionen des Master-Node vollständig übernimmt. Damit können keine Geräte gesteuert werden. Die Definition eines entsprechenden Protokolls würde den Rahmen des laufenden Projektes sprengen. Das heißt auch, dass der Datenaustausch in der isochronen Periode unberührt bleibt.

6.6.1 Cold-Plug-and-Play

Soll das Diagnose-Gerät den gesamten Anlauf protokollieren können, muss es aktiv sein, wenn der Master-Node mit der Kommunikation beginnt. Hierzu wäre ein Dienst erforderlich um das Vorhandensein des Diagnose-Gerätes festzustellen. Bei dieser Plug-and-Play-Variante ist eine Wartezeit einzuplanen, um sicher zu stellen, dass ein eventuell vorhandenes Diagnose-Gerät aktiv ist. Um die Kommunikation lückenlos verfolgen zu können, ist das Diagnosegerät an einer Stelle des Netzwerks zu platzieren, an der keine Telegramme durch Switches oder "intelligente Hubs" gefiltert werden. Das Diagnose-Gerät ist mit Rechten zu versehen, um Geräte- bzw. Anwendungs-Parameterwerte in den Slave-Nodes lesen und schreiben zu können.

6.6.2 Hot-Plug-and-Play

Das spontane Hinzufügen und Entfernen eines Diagnose-Gerätes im Regelbetrieb gehört zu den elementaren Aktivitäten einer Diagnose. Es müssen Dienste und Protokolle spezifiziert werden, mit denen feststellbar ist, dass das Diagnose-Gerät hinzugefügt oder entfernt wurde.

6.6.3 Coordinated-Plug-and-Play

Zum Coordinated-Plug-and-Play gehört, dass dem Master-Node bekannt gemacht wird, dass ein Diagnose-Gerät zum Netzwerk hinzugefügt wird. Das kann beim Anlauf durch einen Nutzereingriff erfolgen oder im Regelbetrieb wenn ein Fehler registriert wird. In diesen Fällen wären die Dienste und Protokolle zu aktivieren, die den Betrieb eines Diagnose-Gerätes ermöglichen. Andernfalls ist dieser zusätzlich Traffic nicht erforderlich.

6.7 Szenario 7: Reglerauslegung für Roboter mit Rapid-Prototyping

Dieses Szenario erfordert die teilweise oder vollständige Übernahme der Funktionen eines Master-Nodes, einschließlich der Steuerung von Slave-Nodes durch zyklische Prozessdatenübertragung (Process Data Objects). Ein Protokoll zur teilweisen Übernahme würde den Rahmen des laufenden Projektes sprengen. Für die vollständige Übernahme wird ein entsprechendes Protokoll spezifiziert.

6.7.1 Cold-Plug-and-Play

Das Gerät mit der Rapid-Prototyping-Funktionalität muss vom Master-Node erkannt werden. Es ist auszuhandeln, welche Slave-Nodes von welchem Gerät gesteuert werden.

6.7.2 Hot-Plug-and-Play

In diesem Szenario ist das unkoordinierte Hinzufügen oder Entfernen eines Gerätes mit Rapid-Prototyping-Funktionalität für den Regelbetrieb nicht sinnvoll.

6.7.3 Coordinated-Plug-and-Play

Das koordinierte Hinzufügen oder Entfernen eines Gerätes im Regelbetrieb ist für dieses Szenario sinnvoll. Das Gerät wird angesprochen, wenn es entsprechend des Programmablaufs dem Netzwerk hinzugefügt wurde. Anschließend ist das Verhalten identisch mit dem beim Cold-Plug-and-Play.

6.8 Szenario 8: Anschluss von nicht projektierten Anwendungen

6.8.1 Cold-Plug-and-Play

Das Vorhandensein eines nichtprojektierten Gerätes wird erkannt. Anwendungs- und Geräte-Parameterwerte und Treiber sind zu identifizieren und der Nutzerinteraktion zur Verfügung zu stellen. Damit ist nur ein konfigurierbares Plug-and-Play möglich.

6.8.2 Hot-Plug-and-Play

In diesem Szenario ist das unkoordinierte Hinzufügen oder Entfernen eines Gerätes möglich.

6.8.3 Coordinated-Plug-and-Play

Das koordinierte Hinzufügen oder Entfernen eines Gerätes im Regelbetrieb ist für dieses Szenario sinnvoll.

7 Bewertung der geforderten Plug-and-Play-Funktionalität

Ausgehend von der verbalen Beschreibung der Plug-and-Play-Funktionen in den vorgegebenen Szenarien wird in **Fehler! Verweisquelle konnte nicht gefunden werden.** eine Bewertung der Plug-and-Play-Funktionalität vorgenommen. Es lassen sich folgende Schlussfolgerungen ziehen:

- Ein unkoordiniertes Hinzufügen oder Entfernen von Geräten ist zwar nicht prinzipiell auszuschließen, es besteht aber, außer bei Ausfall eines Gerätes, immer die Möglichkeit diese Aktionen mit dem Steuerungsprozess des gesamten Systems zu koordinieren. Deshalb sollte die PAPAS-PnP-Spezifikation neben dem Trivialfall "Cold-

Plug-and-Play" vorrangig das "Coordinated-Plug-and-Play" berücksichtigen. Bezüglich des Hot-Plug-and-Plays ist lediglich der Geräteausfall zu berücksichtigen.

- Außer bei Geräten, die eine Vielzahl an Funktionsbausteinen anbieten (Szenario 3) bzw. die bei der Projektierung nicht bekannt sind (Szenario 9) ist grundsätzlich ein vollständiges Plug-and-Play empfehlenswert und sollte durch die PAPAS-PnP-Spezifikation berücksichtigt werden.

	Szenario								
	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
Plug-and-Play-Varianten									
Cold-Plug-and-Play	+	E	E	x	x	x	x	E	+
Hot-Plug-and-Play	-	x	x	x	-	-	+	-	x
Coordinated-Plug-and-Play	E	+	+	x	E	E	E	+	+
Plug-and-Play-Stufen									
Vollständiges Plug-and-Play	E	E	x*	E	E	E	E	+	-
Halbautomatisches Plug-and-Play	x	+	-	x	x	x	x	-	-
Konfigurierbares Plug-and-Play	x	-	E	x	x	x	-	E	E

Szenario 1: Lastdatenermittlung
Szenario 2: Vorkonfigurierte Geräte
Szenario 3: Geräte mit Funktionsbausteinen
Szenario 4: Gerätetausch
Szenario 5: Greiferwechsel
Szenario 6: Alternative Geräte
Szenario 7: Diagnose
Szenario 8: Rapid-Prototyping
Szenario 9: Nichtprojektierte Geräte

\- : nicht sinnvoll
x : möglich
\+ : sinnvoll
E : empfohlen

* bei Wiederanlauf sonst nicht sinnvoll

8 Strukturen der PAPAS-PnP-Spezifikation

8.1 Anwendungsstruktur

Die Anwendungsstruktur der PAPAS-PnP-Spezifikation einschließlich der Relationen zu den einzelnen Spezifikations-Ebenen ist in Abbildung 2 dargestellt.

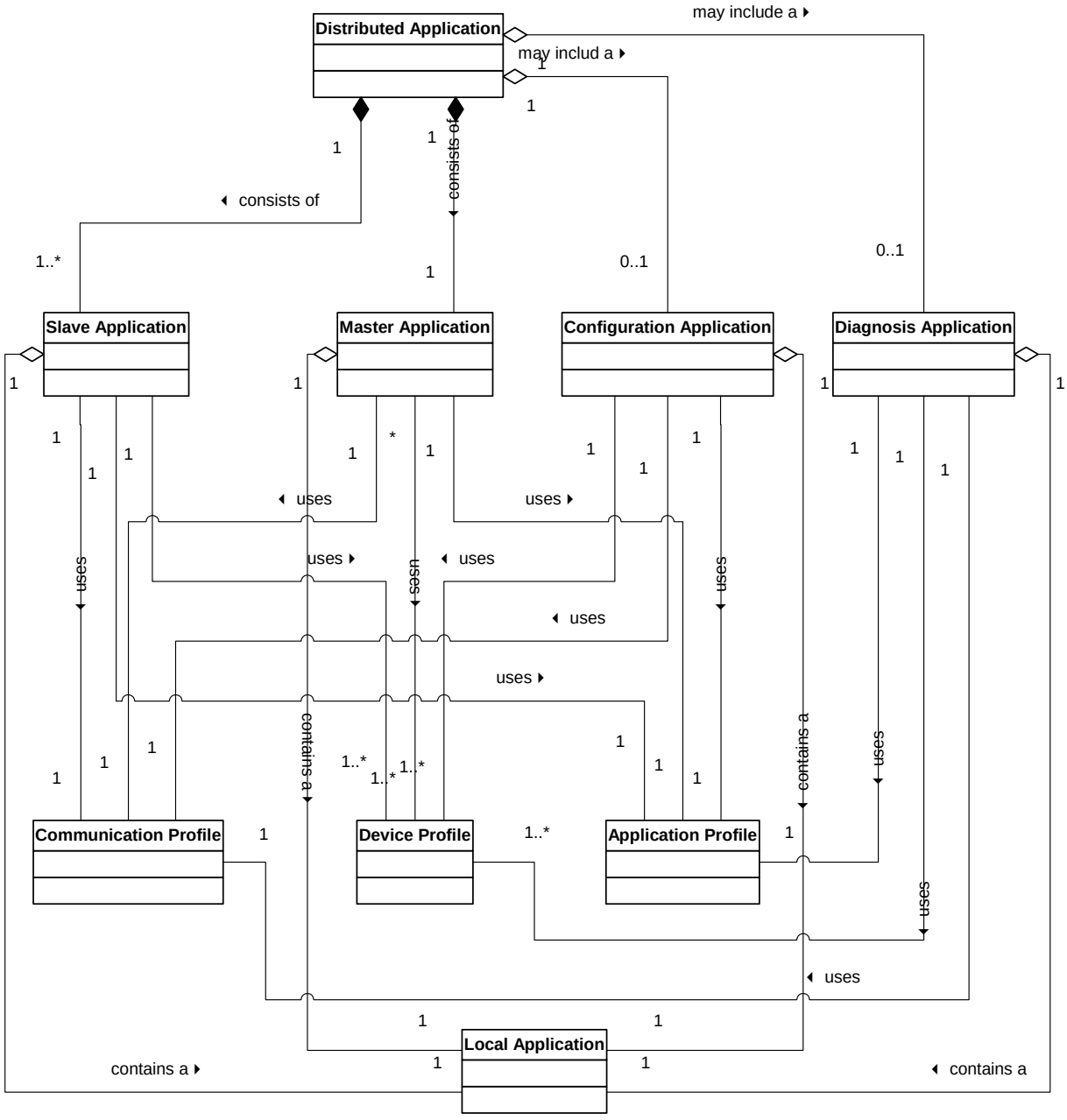


Abbildung 1: Anwendungsstruktur der PAPAS-PnP-Spezifikation

Die verteilte Anwendung beinhaltet eine Master-Anwendung und mehrere Slave-Anwendungen. Außerdem können eine Diagnose-Anwendung und eine Konfigurationsanwendung integriert sein. Die PAPAS-PnP-Funktionalität wird dadurch sichergestellt, dass die Anwendungen auf den Spezifikationen des Kommunikationsprofils, des Anwendungsprofils und der Geräteprofile aufsetzen. Die entsprechenden Implementierungen der Profilobjekte und des Profilverhaltens werden durch die lokalen Anwendungs-Implementierungen ergänzt, die keiner Spezifikation unterliegen müssen.

8.2 Netzwerkstruktur

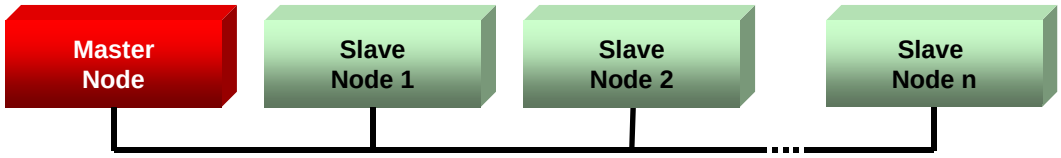


Abbildung 2: Logische Kommunikationsarchitektur der PAPAS-PnP-Spezifikation

Abbildung 3 zeigt die Netzwerkstruktur, die der PAPAS-PnP-Spezifikation zugrunde liegt. Ein Master-Node kommuniziert azyklisch bzw. zyklisch, aber nicht zwingend isochron, mit einem oder mehreren Slave-Nodes gleichzeitig. Der Master-Node ist dabei für folgende Aufgaben verantwortlich:

- Verwaltung von Daten (Objekten), sowohl lokal als auch in den Slave-Nodes, die für die Konfiguration, Regelbetrieb und Wartung der verteilten Anwendung erforderlich sind
- Registrierung von Ereignissen, sowohl lokal als auch in den Slave-Nodes, die eine Konfiguration oder Wartung erforderlich machen
- Initiierung bzw. Überwachung von Aktionen zur Konfiguration und Diagnose, sowohl lokal als auch in den Slave-Nodes

Der Master-Node kann eine SPS oder ein PC sein. Die Slave-Nodes können z. B. einfache Sensoren, Antriebe oder Kamerasysteme sein. In den Slave-Nodes werden die PnP-relevanten Daten flüchtig bzw. nicht-flüchtig gespeichert. Die Slave-Nodes stellen Dienste zur Verfügung um PnP-relevante Ereignisse zu registrieren und um entsprechende Aktionen zur Konfiguration und Diagnose zu veranlassen.

8.3 Beziehung zwischen Anwendungs- und Netzwerksstruktur

Die Beziehung zwischen Anwendungs- und Netzwerksstruktur zeigt Abbildung 4. Die Slave-Anwendungen laufen auf Slave Nodes ab, die Master-Anwendungen auf dem Master Node. Die Konfiguration eines Slave Node kann auf dem Slave Node selbst gespeichert sein oder anderweitig für den Master Node verfügbar sein, falls der Slave Node über keinen nicht-flüchtigen Speicher verfügt. Das schließt die Verbindung zu einer Datenbank ein.Die Konfigurations-Anwendung kann Bestandteil des Master Nodes oder eines speziellen Slave Nodes sein. Die Konfigurations-Anwendung kann aber auch auf einem Gerät laufen, das nicht Bestandteil des Netzwerkes ist. In diesem Fall wird die Verbindung zu den Netzwerk-Knoten über andere Schnittstellen (z. B. RS232) realisiert.
Die Diagnose-Anwendung kann Bestandteil des Master Nodes oder eines speziellen Slave Nodes sein. Die Diagnose-Anwendung kann aber auch als auf die Netzwerk-Knoten verteilte Anwendung realisiert werden.

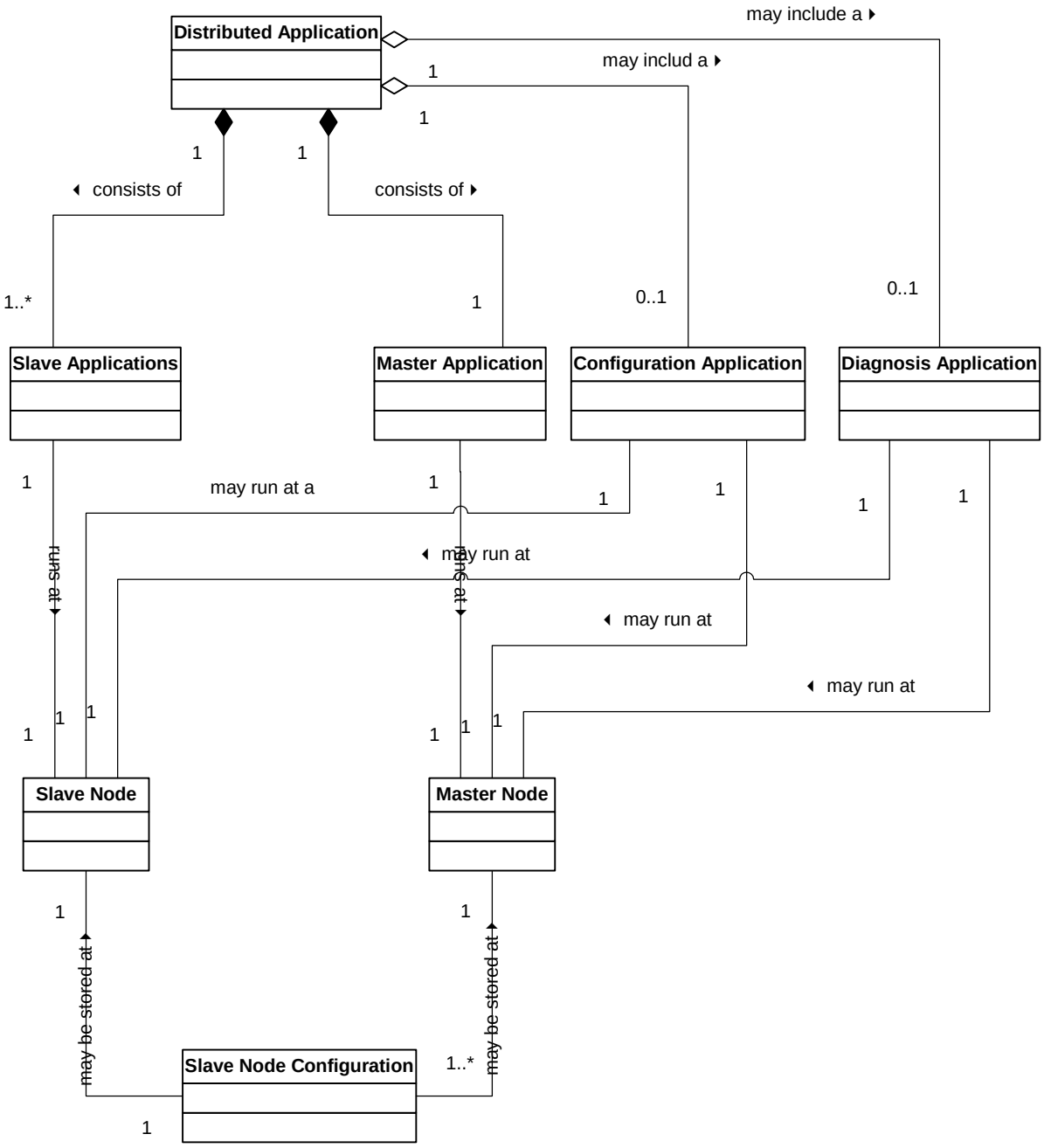


Abbildung 4: Beziehung zwischen Anwendungs- und Netzwerksstruktur

8.4 Gerätestruktur

8.4.1 Struktur der Master Node-Spezifikation

In Abbildung 5 ist die Struktur der Master Node-Spezifikation dargestellt. Sie berücksichtigt die Kommunikations-, Geräte- und Anwendungsprofile. Der Gerätetyp wird im Geräteprofil definiert. Daraus leiten sich auch die Geräte-Parametersätze für den Master Node und die Slave Nodes ab. Die Geräte-Parameterwerte ergeben sich daraus unter Berücksichtigung der Anwendungs-Parameterwerte.
Ein Master Node ist ein Gerät eines bestimmten Typs einschließlich der Beschreibung der Eigenschaften entsprechend den Festlegungen in den Profilen. Der konfigurierte Master Node hat Zugriff auf die Parameterwerte aller Slave Nodes und beinhaltet die eigenen Parameterwerte.

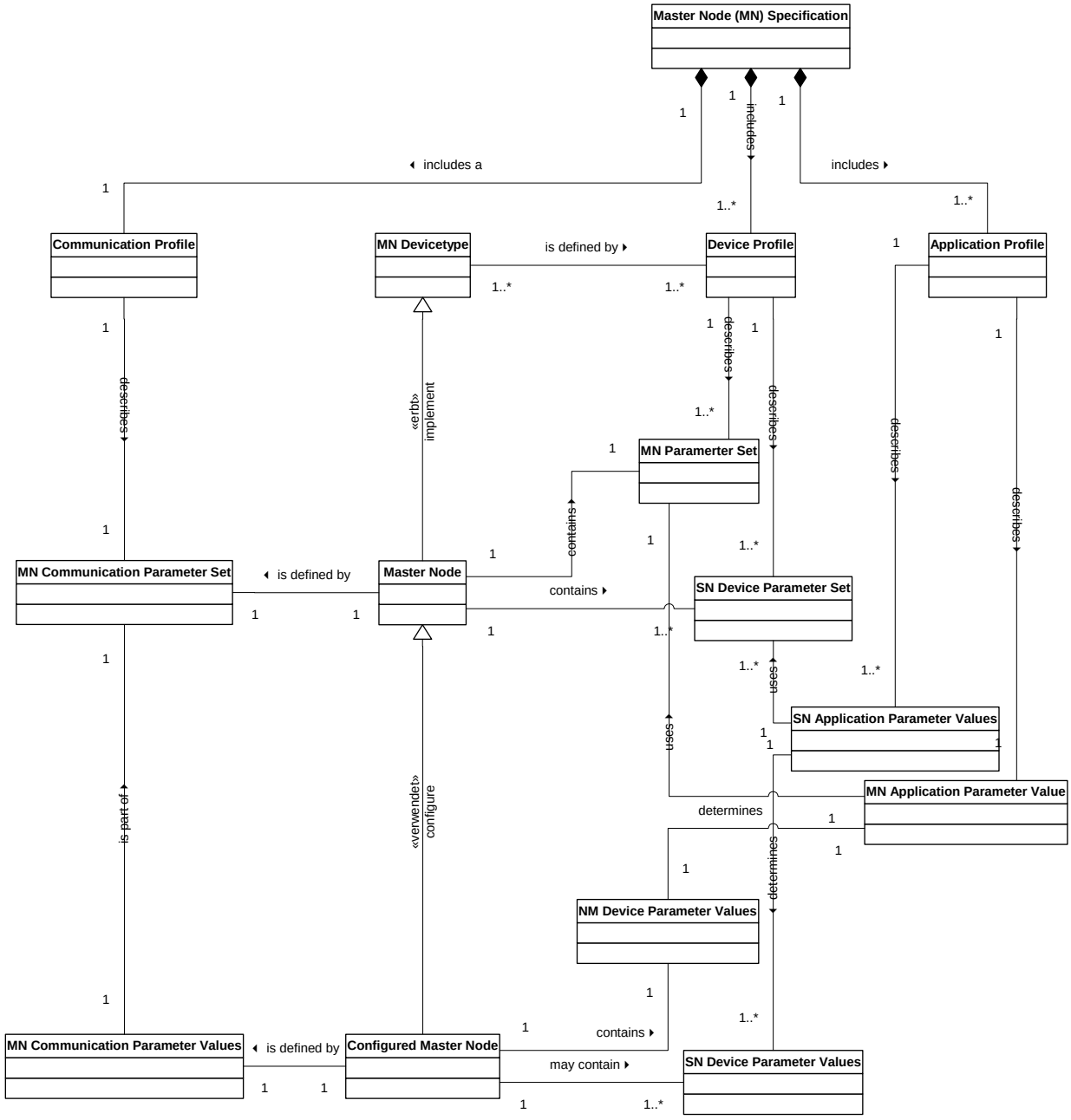


Abbildung 5: Struktur der Master Node-Spezifikation

Die Objekte, Dienste, Verhaltensweisen und Kodierungen der PAPAS-PnP-Spezifikation werden in den Geräte- und Anwendungsprofilen definiert. Das Kommunikationsprofil ist für das Plug-and-Play bzgl. der Kommunikations-Parameter verantwortlich. Die PAPAS-PnP-Spezifikation setzt voraus, dass diese Funktionalität vorhanden ist. Die Anwendungs-Parameterwerte werden im Entwurfsprozess unabhängig von konkreten Geräten erstellt. Es ist lediglich eine Verteilung der Anwendungs-Parameterwerte auf Gerätetypen erforderlich. Die Mechanismen zur Generierung der entsprechenden Geräte-Parameterwerte ist Bestandteil der Anwendungsprofile. Diese Abbildungsvorschriften können sowohl im Master Node als auch in den Slave Nodes ablaufen.
Es ist auch möglich, dass die Anwendungs-Parameterwerte erst beim Systemanlauf ermittelt werden (z. B. Lastdatenermittlung). In diesem Fall ist die automatische Abbildung der Anwendungs-Parameterwerte auf die Geräte-Parameterwerte besonders von Bedeutung.

Im Einzelfall kann die Geräte-Konfiguration auch mithilfe von Konfigurations-Anwendungen erstellt werden. Die PAPAS-PnP-Spezifikation stellt die Mechanismen zur Verfügung, um den

Anwendungen in den Slave-Nodes bzw. im Master-Node die Geräte-Konfiguration bekannt zu machen.

8.4.2 Struktur der Slave Node-Spezifikation

In Abbildung 6 ist die Struktur der Slave Node-Spezifikation dargestellt. Sie berücksichtigt die Kommunikations-, Geräte- und Anwendungsprofile. Der Gerätetyp wird im Geräteprofil definiert. Daraus leiten sich auch die Geräte-Parametersätze für den Slave Node ab. Die Geräte-Parameterwerte ergeben sich daraus unter Berücksichtigung der Anwendungs-Parameterwerte.

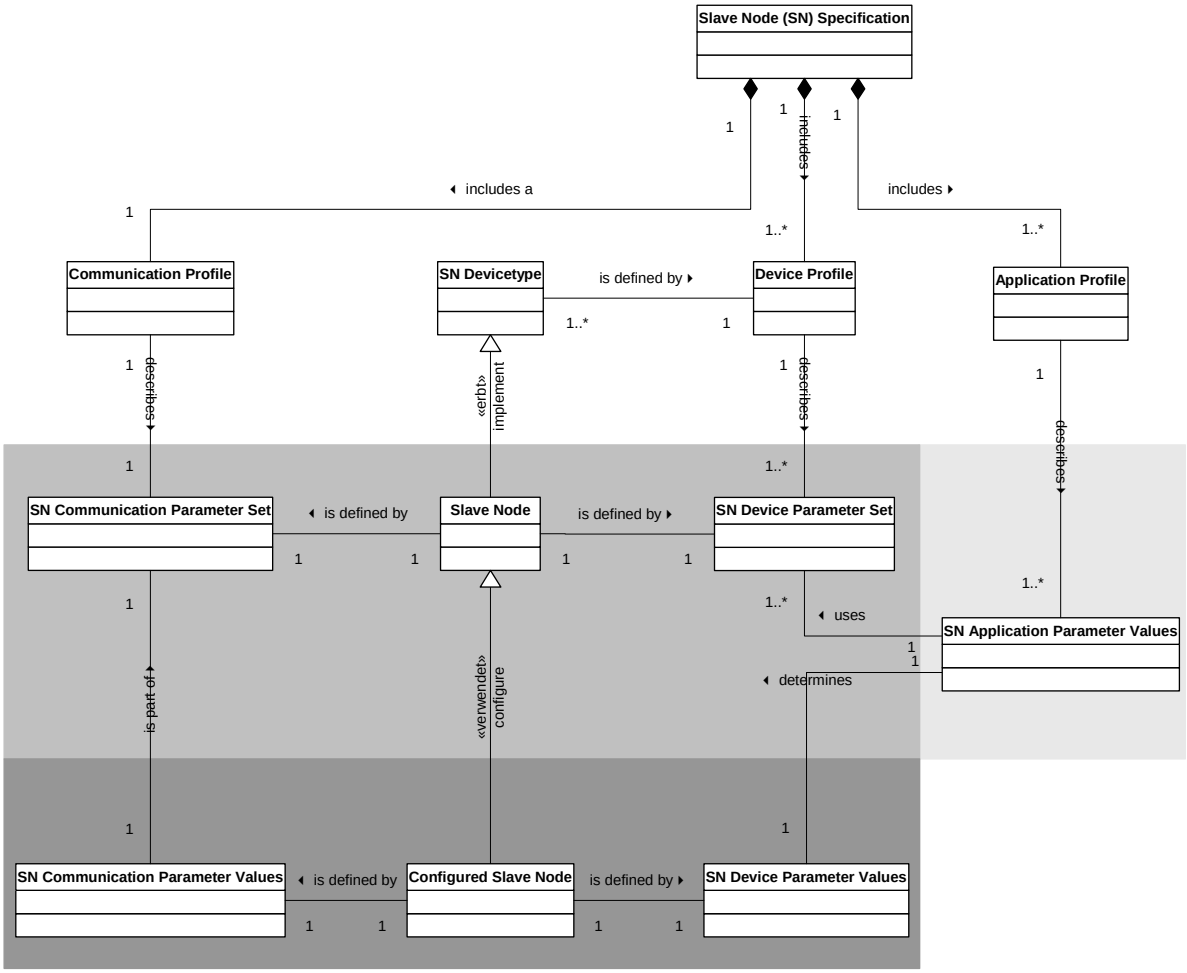


Abbildung 6: Struktur der Slave Node-Spezifikation

Ein Slave Node ist ein Gerät eines bestimmten Typs einschließlich der Beschreibung der Eigenschaften entsprechend den Festlegungen in den Profilen. Der Ersatz eines Slave Nodes durch einen anderen gleichen Typs wird durch die PAPAS-PnP-Spezifikation ermöglicht. Gerätetyp und Geräte-Parametersatz müssen in diesem Fall identifizierbar sein. Dann können daraus unter Berücksichtigung der Anwendungs-Parameterwerte die Geräte-Parameterwerte generiert werden.
Sollte das Austauschgerät mit einem anderen Anwendungsparametersatz verbunden sein muss dieser ebenfalls identifiziert und besonders berücksichtigt werden.

8.4.3 Implementierungs-Varianten

Die PAPAS-PnP-Spezifikation lässt eine Reihe von Implementierungsvarianten zu, die bei der Definition der Objekte, Dienste, Verhaltensweisen und Kodierungen zu berücksichtigen sind. Für Master Nodes sind folgende Varianten möglich:

- Master Nodes mit Konfigurations-Anwendung
- Master Nodes ohne Konfigurations-Anwendung
- Master Nodes mit Diagnose-Anwendung

- Master Nodes ohne Diagnose-Anwendung
- Master Nodes die Parameterwerte eines Slave Nodes speichern
- Master Nodes die Parameterwerte eines Slave Nodes aus einer Datenbank beziehen
- Master Nodes die Parameterwerte eines Slave Nodes vom Slave Node beziehen
- Master Nodes die Geräte-Parameterwerte eines Slave Nodes aus den Anwendungs-Parameterwerten bestimmen

Darüber hinaus können alle Kombinationen der genannten Varianten in einem Master Node implementiert sein.
Für Slave Nodes sind folgende Varianten möglich:

- Slave Nodes die Parameterwerte flüchtig speichern
- Slave Nodes die Parameterwerte nicht-flüchtig speichern
- Slave Nodes die Geräte-Parameterwerte aus den Anwendungs-Parameterwerten bestimmen
- Spezielle Slave Nodes mit Konfigurations-Anwendung
- Spezielle Slave Nodes mit Diagnose-Anwendung

Auch bei den Slave-Nodes können Kombinationen der genannten Varianten implementiert sein.

9 Beschreibung der Objekte der PAPAS-PnP-Spezifikation

9.1 Gerätespezifische Objekte

Gerätespezifische Objekte beschreiben die charakteristischen Parameter eines Gerätes.
Die Struktur der gerätespezifischen Objekte wird in den Geräteprofilen definiert.

9.1.1 Gerätetyp

Dieses Objekt gibt Auskunft über den Gerätetyp. Zum einen wird Bezug auf eine Geräteprofil genommen, zum anderen wird angegeben, ob ein Gerät bezogen auf die PAPAS-PnP-Spezifikation ein Master-Node oder ein Slave-Node ist. Mit diesen Informationen kann bei einem Gerätetausch entschieden werden, ob eine adäquates Gerät vorliegt.

9.1.2 Konfigurations-Zustand

Dieses Objekt gibt Auskunft ob die Geräte- bzw. Anwendungs-Parameterwerte verfügbar ist und wenn ja ob er einen aktuellen Wert besitzt oder nicht. Mit diesem Objekt wird der Ablauf während eines Anlaufes bzw. nach einem Gerätetausch gesteuert.

9.1.3 Geräte-Datenblatt

In diesem Objekt sind die Leistungsdaten des Gerätes gespeichert.
Der Master-Node liest ggf. diese Daten um festzustellen, welche Aufgaben dem Gerät zugeordnet werden können.
Die Struktur des Objektes "Geräte-Datenblatt" sollte in den Geräteprofilen definiert werden und kann sich somit von Gerätetyp zu Gerätetyp unterscheiden.

9.1.4 Unterstützte Betriebsmodi

In diesem Objekt werden für den Gerätetyp relevante Betriebsmodi gespeichert, die vom Gerät unterstützt werden.
Der Master-Node liest ggf. dieses Objekt um festzustellen, welche Aufgaben dem Gerät zugeordnet werden können.
Die Struktur des Objektes "Unterstützte Modi" sollte in den Geräteprofilen definiert werden und kann sich somit von Gerätetyp zu Gerätetyp unterscheiden.

9.1.5 Geräte-Parametersatz-Beschreibung (DDF)

In diesem Objekt wird die Beschreibung der Geräteobjekte abgelegt. Der Inhalt ist mit dem Device Description File (XML-Schema) des Gerätes identisch. Die Geräte-Parametersatz-Beschreibung wird vorzugsweise im Slave-Node gespeichert, alternativ aber auch im Master-Node.

9.1.6 Geräte-Parametersatz (DCF)

In diesem Objekt wird die konkrete Wertebelegung der Geräteobjekte abgelegt. Der Inhalt ist mit der Device Configuration File (XML-Instanzdatei) des Gerätes identisch. Der Geräte-Parametersatz wird im Master-Node abgelegt und hat seine Entsprechung als Parametersatz im Slave-Node.

9.1.7 Geräte-Parametersatz-Identifikation

In dieses Objekt wird die Identifikation eines Geräte-Parametersatzes abgelegt. Die Identifikation könnte sich aus Datum und Uhrzeit der Konfiguration ergeben. Dies ist insbesondere interessant, wenn die Konfiguration Off-line durch ein Engineering-Tool erstellt wurde.
Der Master-Node übernimmt einen vorhandenen Geräte-Parametersatz und stellt die Werte der Anwendung zur Verfügung. Sollten Änderungen an den Parameterwerten während des Betriebes vorgenommen werden, so ist der Wert für die Geräte-Parametersatz-Identifikation zu aktualisieren.
Das Objekt "Geräte-Parametersatz-Identifikation" sowie die Objekte die zum Geräte-Parametersatz gehören, werden in den Geräte-Profilen definiert. Die Zusammenstellung des Geräte-Parametersatzes könnte dabei sowohl als statisch als auch als dynamisch definiert werden. Im letzteren Fall ist ein weiteres Objekt erforderlich, in dem die Zusammensetzung des Geräte-Parametersatzes hinterlegt wird.

9.1.8 Unterstützte Geräte-Treiber

Mit diesem Objekt kann auf Anwendungssoftware referenziert werden, die im Master-Node abläuft und die Anwendungsfunktionalität des Slave-Nodes nutzt. Für einen Slave-Node kann es mehrere Treiber geben. Bei komplexen Slave-Nodes können auch mehrere Treiber (Funktionsbausteine) im Master-Node aktiv sein. Das Objekt sollte Informationen wie Typ, Version, Name und eine eindeutige Nummer beinhalten. Die Definition des Objektes "Unterstützte Geräte-Treiber " sollte Bestandteil eine Geräteprofils sein.
Das Objekt nutzt der Master-Node, um den bzw. die passenden Geräte-Treiber zu laden und zu aktivieren. Damit kann die Anwendung im Master-Node auf die erforderlichen Funktionen zugreifen.
Sind mehrere Geräte-Treiber möglich, sind die Werte des Objektes der Nutzer-Interaktion zur Auswahl zur Verfügung zu stellen.
Ob und wie das Objekt "Unterstützte Geräte-Treiber" definiert wird sollte in den Geräte-Profilen festgelegt werden. Die Wertebereiche werden herstellerspezifisch sein.
Geräte-Treiber-Identifikation
In diesem Objekt wird hinterlegt, welche(r) Geräte-Treiber für die relevante Anwendung gewählt wurde(n).
Im Falle eines Wiederanlaufs ist eine sonst erforderliche Nutzerinteraktion nicht nötig, wenn der Master-Node auf diese Information zurückgreifen kann.
Das Objekt "Geräte-Treiber-Identifikation" sollte in den Geräte-Profilen definiert werden. Die Wertebereiche werden herstellerspezifisch sein.

9.1.9 Geräte-Treiber-Identifikation

In diesem Objekt wird hinterlegt, welche(r) Geräte-Treiber für die relevante Anwendung gewählt wurde(n).
Im Falle eines Wiederanlaufs ist eine sonst erforderliche Nutzerinteraktion nicht nötig, wenn der Master-Node auf diese Information zurückgreifen kann.
Das Objekt "Geräte-Treiber-Identifikation" sollte in den Geräte-Profilen definiert werden. Die Wertebereiche werden herstellerspezifisch sein.

9.1.10Zusammenfassung

Ein Auszug aus der gerätespezifischen Objektbibliothek für Slave-Nodes ist in der nachfolgenden Tabelle zusammengefasst.

Index (hex)	Objekt	Name	Access
0x1000	VAR	Node Type (Device Type)	ro
0x1002	VAR	Konfigurations-Zustand (Manufacturer Status Register)	ro
0xA000	RECORD	Geräte-Datenblatt	ro
0xA001	ARRAY	Unterstützte Betriebsmodi	ro
0xA002	RECORD	Geräte-Parametersatz-Beschreibung	ro
0xA003	RECORD	Geräte-Parametersatz	rw
0xA004	VAR	Geräte-Parametersatz-Identifikation	rw
0xA005	ARRAY	Unterstützte Geräte-Treiber	ro
0xA006	ARRAY	Geräte-Treiber-Identifikation	rw

9.2 Anwendungsspezifische Objekte

Die Anwendung wird durch eine Reihe von Parametern charakterisiert, die nicht zwangsläufig mit Geräteparametern identisch sein müssen. Die Anwendungsparameter werden in einer verteilten Steuerungsarchitektur den verschiedenen Knoten im Netzwerk zugeordnet. Die Abbildung der Anwendungsparameter auf einen oder mehrere Geräteparameter kann sowohl in den Slave-Nodes als auch in den Master-Nodes erfolgen. Dementsprechend werden die anwendungsspezifischen Objekte auch entweder in den Slave-Nodes oder in den Master-Nodes gespeichert. Durch dieses Prinzip ist es möglich, dass die Anwendungsparameter durch verschiedene Geräte auf unterschiedliche Art und Weise umgesetzt werden. Ein Beispiel dafür ist Szenario 5, bei dem ein Gegenstand durch unterschiedliche Greifer (mit verschiedenen Geräte-Parameterwerten) adäquat (entsprechend den Anwendungsparametern) gegriffen werden kann.
Die Struktur der anwendungsspezifischen Objekte wird in den Anwendungsprofilen definiert.
Master-Node-Priorität
Dieses Objekt wird dazu genutzt, um in einem System mit mehreren Geräten, die Master-Node-Funktionalität implementiert haben, den Master-Node festzulegen. So könnte in einem Rapid-Prototyping-PC die Master-Node-Priorität in der Entwicklungsphase herauf gesetzt und anschließend wieder herab gesetzt werden.

9.2.1 Anwendungs-Parametersatz-Beschreibung (ADF)

In diesem Objekt wird die Beschreibung der Anwendungsobjekte abgelegt. Der Inhalt ist mit dem Application Description File (XML-Schema) der Anwendungsbeschreibung für dieses Geräte identisch. Die Anwendungs-Parametersatz-Beschreibung wird vorzugsweise im Slave-Node gespeichert, alternativ aber auch im Master-Node.

9.2.2 Anwendungs-Parametersatz (ACF)

In diesem Objekt wird die konkrete Wertebelegung der Anwendungsobjekte abgelegt. Der Inhalt ist mit dem Application Configuration Files (XML-Instanzdatei) der Anwendungsbeschreibung für dieses Gerät identisch. Der Anwendungs-Parametersatz wird im Master-Node abgelegt und kann seine Entsprechung als Parametersatz im Slave-Node haben.

9.2.3 Anwendungs-Parametersatz-Identifikation

Die Anwendungs-Parametersatz-Identifikation adressiert einen Anwendungs-Parametersatz für einen bestimmten Slave-Node im verteilten Netzwerk. Wird das entsprechende Gerät getauscht oder wird es nach Ausfall ersetzt, kann der Master-Node dem neuen Slave-Node die anwendungsspezifischen Objektwerte zuordnen. Daraus werden dann die Werte der gerätespezifischen Objekte abgeleitet.

9.2.4 Parameterwerte-Beziehung

In diesem Objekt ist die Beziehung zwischen Anwendungs-Parameterwerten und Geräte-Parameterwerten abgelegt. Damit ist es dem Master-Node möglich einem Slave-Node die Geräte-Parameterwerte anhand der Anwendungs-Parameterwerte zuzuordnen.

9.2.5 Lastdaten

Diese Struktur beinhaltet Parameter wie Masse, Schwerpunkt und Trägheitstensor. Diese Werte werden nicht konfiguriert, sondern werden automatisch ermittelt und dann der Anwendung im Master-Node zur Verfügung gestellt. Die Lastdaten sind Teil des Anwendungs-Parametersatzes.

9.2.6 Zusammenfassung

Index (hex)	Objekt	Name	Access
0xA100	VAR	Master-Node-Priorität	ro
0xA101	RECORD	Anwendungs-Parametersatz-Beschreibung	ro
0xA102	RECORD	Anwendungs-Parametersatz	rw
0xA103	VAR	Anwendungs-Parametersatz-Identifikation	rw
0xA104	RECORD	Parameterwerte-Beziehung	rw
0xA105	RECORD	Lastdaten	rw

10 Beschreibung der Dienste und Protokolle der PAPAS-PnP-Spezifikation

10.1 Dienste der Geräte- und Anwendungsprofile

10.1.1 Geräteklassenidentifikation durch den Master-Node

Jeder Slave-Node muss dem Master-Node seine Geräteklassenidentifikation zur Verfügung stellen können. Der Master-Node ließt die Geräteklassenidentifikation aus, wenn er den Slave-Node als neues Gerät erkannt hat.

10.1.2Gerätekonfigurationszustandsidentifikation durch den Master-Node

Der Gerätekonfigurationszustand wird vom Master-Node ausgelesen. Je nach Zustand (keine Konfiguration vorhanden, nicht konfiguriert, konfiguriert) muss die Konfiguration komplett oder teilweise durchgeführt werden.

10.1.3 Automatische Treiberzuordnung im Master-Node

Eine automatische Treiberzuordnung im Master-Node kann erfolgen, wenn die Zuordnung eindeutig ist. Ist dies nicht der Fall muss über eine manuelle Aktion am Mastern-Node eine Auswahl vorgenommen werden.
Die automatische Zuordnung des Treibers erfolgt bei Eindeutigkeit über die Geräteidentifikation des Slave-Nodes. Dazu wird die Geräteidentifikation vom Master-Node ausgelesen.

10.1.4 Nichtflüchtige Speicherung von Geräte-/Anwendungsdaten im Slave-Node

Die Geräte- und Anwendungsdaten können je nach Gerät flüchtig oder nichtflüchtig in der Management-Information-Base (MIB) des Gerätes abgelegt werden. Besteht die Möglichkeit der nichtflüchtigen Speicherung ist ein Dienst bereitzustellen, der die Parameterwerte in diesen Speicherbereich übernehmen kann.

10.1.5Nichtflüchtige Speicherung von Geräte-/Anwendungsdaten im MasterNode

Können die Geräte- und Anwendungsdaten nur flüchtig in einem Gerät gespeichert werden, ist ein Dienst bereitzustellen, der die Parameterwerte in die System-Management-Information-Base (System-MIB) übernimmt.

10.1.6 Slave-Node-Erkennung im Netzwerk durch Master-Node

Durch diese Dienst ist anzuzeigen, dass ein Slave-Node zum Netzwerk hinzugekommen ist bzw. das Netzwerk verlassen hat. Daraufhin können entsprechende Maßnahmen beim Anlauf bzw. beim Gerätetausch eingeleitet werden.

10.1.7 Nichtflüchtige Speicherung der Anwendungs-Parametersatz-Identifikation im Slave-Node

Die nichtflüchtige Speicherung der Anwendungs-Parametersatz-Identifikation im Slave-Node ermöglicht einen schnellen Wiederanlauf, da zur Zuordnung zwischen Slave-Node und Anwendungs-Parametersatz keine erneute Nutzerinteraktion erforderlich ist.

10.1.8 Nutzerinteraktion zur Treiberauswahl

Stehen zur Bedienung des Gerätes mehrere Treiber zur Auswahl, ist ein Dienst erforderlich, der die Auswahl des geeigneten Treibers über eine Nutzerinteraktion unterstützt.

10.1.9 Nutzerinteraktion zur Konfiguration des Slave-Node

Für den Fall das ein Geräte zum Netzwerk hinzugefügt werden soll, das im Projekt nicht berücksichtigt werden muss, ist ein Dienst erforderlich, der die Konfiguration über eine Nutzerinteraktion unterstützt.

10.1.10 Nutzerinformation über defektes Gerät

Dem Nutzer ist für Diagnosezwecke ein Dienst zur Verfügung zu stellen, der Informationen über Fehlerursachen bereitstellt.

10.1.11 Reorganisation des isochronen Datenaustausches

Für den Fall das ein Geräte zum Netzwerk hinzugefügt werden soll, das im Projekt nicht berücksichtigt werden muss, ist ein Dienst erforderlich, der die Reorganisation des isochronen Datenaustausches realisiert.

10.1.12 Konfiguration von Slave-Nodes

Ein Slave-Node kann die Konfiguration durch den Master-Node anfordern (z. B. Gerätetausch). Die Konfigurationsanforderung kann sich auf Geräteparameter oder Anwendungsparameter beziehen. Im letzteren Fall ist der Slave-Node in der Lage, aus den Anwendungsparameter die Werte für die Geräteparameter zu generieren.

10.1.13 Speicherung der Parametersatz-Beschreibungen

Die Speicherung der Parametersatz-Beschreibungen (DDF, ADF) in die Slave-Nodes bzw. im Master-Node sichert die Konsistenz zwischen Firmware und Parametersatz-Beschreibungen und bietet zusätzlich die Möglichkeit, für die Anwendung auf der Grundlage dieser Informationen eine optimale Konfiguration der Geräteparameter zu ermitteln.

10.1.14 Zusammenfassung

In **Fehler! Verweisquelle konnte nicht gefunden werden.** sind die Funktionen zusammengefasst, die sich speziell aus der verbalen Beschreibung der Szenarien ergeben.

Funktion	Szenario								
	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
Geräteklassenidentifikation durch den Master-Node	+	+	+	+	+	+	+	+	x
Gerätekonfigurationszustandsidentifikation durch den Master-Node	+	-	+	+	x	x	-	-	x
Automatische Treiberzuordnung im Master-Node	+	+	-	+	+	+	+	+	-
Nichtflüchtige Speicherung von Geräte- und Anwendungsdaten im Slave-Node	x	+	x	x	+	+	+	+	+
Nichtflüchtige Speicherung von Geräte- und Anwendungsdaten im Master-Node	x	-	x	+	+	+	x	x	-
Master-Node erkennt neues Gerät (Slave-Node) im Netzwerk	-	+	+	+	x	x	+	+	+
Nichtflüchtige Speicherung der Anwendungsidentifikation im Slave-Node	-	+	x	x	+	+	+	+	+
Nutzerinteraktion zur Treiberauswahl	-	-	+	x	x	x	-	x	+
Nutzerinteraktion zur Konfiguration des Slave-Node	-	-	+	x	x	x	-	-	+
Nutzerinformation über defektes Gerät	-	-	-	+	-	-	-	-	-
Reorganisation des isochronen Datenaustausches (Hinzufügen bzw. Entfernen von Prozessdaten)	+	+	+	x	x	-	x	+	x

Szenario 1: Lastdatenermittlung
Szenario 2: Vorkonfigurierte Geräte
Szenario 3: Geräte mit Funktionsbausteinen
Szenario 4: Gerätetausch
Szenario 5: Greiferwechsel
Szenario 6: Alternative Geräte
Szenario 7: Diagnose
Szenario 8: Rapid-Prototyping
Szenario 9: Nichtprojektierte Geräte

+: Funktion unbedingt erforderlich
x: Funktion bedingt erforderlich
-: Funktion nicht erforderlich

10.2 Protokolle der Geräte- und Anwendungsprofile

- Systeme mit hohen und mit geringeren Determinismusanforderungen
- Geräte mit und ohne nicht-flüchtigen Speicher
- Geräte mit fester und variabler Treiberzuordnung

10.2.1Systemanlauf

Der Systemanlauf wird vom Master-Node gesteuert. Hierzu werden folgende Schritte abgearbeitet:

- lokale Initialisierung des Gerätes
- lokale Initialisierung der Anwendung: Während dieser Initialisierung bekommt der Master-Node ein mehr oder weniger komplettes Bild der verteilten Anwendung. Die Quelle der verteilten Anwendungskonfiguration wird nicht weiter spezifiziert. Am Kommunikationsnetzwerk angeschlossene Teilnehmer sind in dieser Phase als Quelle ausgeschlossen.

- lokale Initialisierung der Kommunikation: Eine Option ist, dass nach der Initialisierung der Kommunikation das Abbild der verteilten Anwendung über das Netzwerk von einem Konfigurationswerkzeug übertragen wird.
- Festlegung des Master-Nodes: Geräte die Master-Node-Funktionalität implementiert haben (z. B. Steuerung und Rapid-Prototyping-PC) handeln aus, wer die Master-Node-Funktionalität ausübt. Ein Wechsel der Master-Node-Funktionalität während des Regelbetriebes ist nicht vorgesehen.
- Feststellung der kommunikationsbereiten Slave-Nodes: Es wird nacheinander festgestellt, welche der im Master-Node konfigurierten Slave-Nodes nach einer Boot-up-Zeit auf eine Anfrage antworten. Anschließend wird das Hinzufügen weiterer Teilnehmer überwacht. Je nach Determinismusanforderung erstreckt sich die Überwachung auf Slave-Nodes, die nur am asynchronen bzw. am asynchronen und isochronen Datenaustausch teilnehmen.
- Identifizierung und Prüfung der Geräte- und Anwendungsparameterwerte: Für Slave-Nodes deren Konfiguration bereits Teil des dem Master-Node bekannten Abbildes der verteilten Anwendung ist, werden die Geräte- und Anwendungs-Parameterwerte des Slave-Nodes identifiziert und mit dem im Master-Node abgelegten Abbild verglichen. Bei Übereinstimmung können die Slave-Nodes sofort aktiviert werden. Insbesondere bei einem Wiederanlauf ohne Spannungsausfall (Warmstart) können die nachfolgend beschriebenen Schritte entfallen.
- Identifizierung und Prüfung des Gerätetyps: Der Master-Node erfragt den Gerätetyp eines Slave-Nodes. Aus dem Abbild der verteilten Anwendung ist dem Master-Node die Zuordnung zwischen Geräteadresse und Gerätetyp bekannt und kann demzufolge geprüft werden. Dieser Schritt ist nur erforderlich, wenn verschiedene Geräte eines Typs unter der Adresse zum Einsatz kommen können (wie z. B. bei den Szenarien Lastdatenermittlung, Gerätetausch, Greiferwechsel).
- Identifizierung des Geräte-Parametersatzes des Slave-Nodes: Können an einem Ort der verteilten Anwendung verschiedene Geräte eines Typs eingesetzt werden, wird in diesem Schritt der Geräte-Parametersatz des Slave-Nodes ermittelt (z. B. Szenario Gerätetausch). Das kann direkt durch Auslesen aus dem Slave-Node geschehen oder indirekt durch Ermittlung einer Identifikation aus dem Slave-Node und anschließendem Lesen des Device-Configuration-Files.
- Ermittlung und Einstellung der Anwendungs-Parameterwerte des Slave-Nodes: Zur Motivation der PAPAS-PnP-Spezifikation gehört, dass Anwendungs-Parameterwerte zu Beginn dem Master-Node nicht vollständig vorliegen müssen. In diesem Schritt werden diese Werte ermittelt. Das kann durch Lesen aus dem Slave-Node geschehen oder durch die Abarbeitung eines definierten Algorithmus. Es wird die Anwendung im Slave-Node identifiziert und ermittelt ob die Anwendungs-Parameterwerte im Slave-Node gespeichert sind (z. B. Szenario Vorkonfigurierte Geräte), in den Slave-Node geladen werden müssen (z. B. Szenario Gerätetausch) oder durch Abarbeitung eines Algorithmus zu ermitteln sind (z. B. Szenario Lastdatenermittlung). Für die Abarbeitung des Algorithmus ist der passende Treiber im Master-Node auszuwählen und zu laden. Es kann außerdem erforderlich sein, spezielle Geräte-Parameterwerte in den Slave-Node zu laden bzw. dort zu aktivieren. Weiterhin ist es denkbar, dass anderen Slave-Nodes (Sensoren) in die Abarbeitung involviert sind und demzufolge vorher in den Mode "Operational" (vollständig konfiguriert und aktiviert) versetzt werden müssen. Der Treiber wird aktiviert und die Anwendungs-Parameterwerte durch den Master-Node bzw. durch den Slave-Node erfasst und dann entsprechend zum Master-Node bzw. zum Slave-Node übertragen sowie im Slave-Node nicht-flüchtig gespeichert.
- Ermittlung und Einstellung der Geräte-Parameterwerte und des erforderlichen Treibers des Slave-Nodes: Die Geräte-Parameterwerte bzw. die erforderlichen Treiber (z. B. Szenario "Geräte mit Funktionsbausteinen") können auf zwei Arten ermittelt werden. Entweder automatisch anhand des Parametersatzes und der Anwendungs-Parameterwerte des Slave-Nodes oder durch Nutzerinteraktion am Master-Node. Im ersten Fall kann die Ermittlung im Master-Node z. B. auf der Basis einer Datenbank oder im Slave-Node z. B. anhand von Berechnungsvorschriften erfolgen. Wurden die Geräte-Parameterwerte im Master-Node ermittelt, werden sie anschließend in den

Slave-Node geladen und ggf. nicht-flüchtig gespeichert. Die Geräte-Parameterwerte können auch im Master-Node vorliegen, weil der Slave-Node Daten nur flüchtig speichert. In diesem Fall sind die Geräte-Parameterwerte ebenfalls an dieser Stellen in den Slave-Node zu laden.

- Bestimmung der Geräte-Parameterwerte des Master-Nodes: Auf der Grundlage der gegebenen und ermittelten Geräte- und Anwendungs-Parameterwerte der Slave-Nodes bestimmt der Master-Node seine Geräteparameterwerte. Dazu zählen in erster Linie die Bandbreiten für den isochronen und asynchronen Datenaustausch.
- Aktivierung der Slave-Nodes: Nach dem die Treiber im Master-Node geladen und alle Geräte- und Anwendungs-Parameterwerte vorhanden sind, werden die Slave-Nodes aktiviert und der isochrone Datenaustausch wird aufgenommen. Die Aktivierung kann für jeden Slave-Node einzeln oder für alle Slave-Nodes insgesamt vorgenommen werden.

10.2.2Diagnose

- passive Diagnose durch Auswertung und Interpretation des gesamten Datenverkehrs (Monitoring auf Anwendungsebene)
- aktive Diagnose durch Lesen und Interpretation von Objekten (insbesondere Fehlerreport- und Fehlerzähl-Objekte)

10.2.3Gerätetausch

- Feststellen eines Gerätefehlers durch den Master-Node: Im Master-Node wird festgestellt, ob ein bereits bekannter Slave-Node irgendwann ausfällt. Wenn dieser Slave-Node ein Teil des bekannten Abbild der verteilten Anwendung im Master-Node ist, müssen die nachfolgenden Schritte durchgeführt werden. Ist der Slave-Node nicht Teil des bekannten Abbilds der verteilten Anwendung, wird lediglich vermerkt, das dieser Slave-Node nicht mehr ansprechbar bzw. vorhanden ist.
- Identifizieren der fehlenden Anwendungsfunktionalität und des Gerätetyps durch den Master-Node: Wenn der Ausfall eines Slave-Nodes festgestellt wird, der Teil des bekannten Abbild der verteilten Anwendung im Master-Node ist, wird die fehlende Anwendungsfunktionalität identifiziert und gekennzeichnet.
- Unterstützung des Gerätewechsels durch Nutzerinteraktion am Master-Node: Falls eine eindeutige Identifizierung der fehlenden Anwendungsfunktionalität durch den Ausfall eines Slave-Nodes nicht möglich ist, muss die Zuordnung durch eine Nutzeraktion erfolgen.
- lokale Initialisierung des Gerätes
- lokale Initialisierung der Anwendung im Slave-Node: Die lokale Initialisierung im Slave-Node ist abhängig von den Informationen, die dem Slave-Node über seinen Teil der verteilten Anwendung zur Verfügung stehen. Im einfachsten Fall ist der Slave-Node kommunikationsbereit. Der Master-Node kann dann die Konfiguration auslesen, die Anwendungsfunktionalität zuordnen und den Slave-Node parametrieren. Je nach Komplexität des Gerätes und Stand der Vorkonfiguration können einige der folgenden Schritte entfallen.
- lokale Initialisierung der Kommunikation: Nach der lokalen Initialisierung der Kommunikation ist der Slave-Node über das Netz ansprechbar.
- Feststellung der kommunikationsbereiten Slave-Nodes (Austauschgerät): Im Master-Node erfolgt eine kontinuierliche Überwachung aller im Netz befindlichen Slave-Nodes. Ein Austauschgerät wird vom Master-Node aufgenommen und bearbeitet, sowie der neue Teilnehmer im Netz kommunikationsbereit ist.
- Identifizierung und Prüfung des Gerätetyps: Dieser Schritt ist ebenfalls im Systemanlauf enthalten. Der Inhalt ist identisch.
- Identifizierung und Prüfung der Geräte- und Anwendungs-Parameterwerte (geforderte Funktionalität): Dieser Schritt ist ebenfalls im Systemanlauf enthalten. Der Inhalt ist identisch.
- Identifizierung des Geräte-Parametersatzes des Slave-Nodes: Dieser Schritt ist ebenfalls im Systemanlauf enthalten. Der Inhalt ist identisch.

- Einstellung der Anwendungs-Parameterwerte des Slave-Nodes: Dieser Schritt ist ebenfalls im Systemanlauf enthalten. Der Inhalt ist identisch.
- Ermittlung und Einstellung der Geräte-Parameterwerte und des erforderlichen Treibers des Slave-Nodes: Dieser Schritt ist ebenfalls im Systemanlauf enthalten. Der Inhalt ist identisch.
- Aktivierung der Slave-Nodes: Dieser Schritt ist ebenfalls im Systemanlauf enthalten. Der Inhalt ist identisch.

10.3 Anforderungen an die Kommunikationsschichten

Die Spezifikation des Kommunikationssystems muss

- zulassen, dass Geräte im Regelbetrieb gesteckt und entfernt werden können, ohne dass die Kommunikation (insbesondere die isochrone Kommunikation) beeinflusst wird.
- gewährleisten, das die Geräte auf IP-Ebene adressierbar und über asynchrone Dienste bzw. Netzwerk-Management-Dienste erreichbar sind.
- gewährleisten, dass der Anschluss eines Gerätes erkannt wird, unabhängig davon ob es am isochronen oder nur am asynchronen Datenaustausch teilnimmt.
- Dienste und Protokolle bereitstellen, die die PAPAS-PnP-Spezifikation nutzen kann, ohne dass die isochrone Kommunikation beeinträchtigt wird.
- zulassen, dass der gesamte Datenverkehr für Diagnosezwecke mitgeschnitten werden kann und nicht Teile durch Switches oder "intelligente Hubs" gefiltert werden.

10.4 Objekte

10.4.1Geräteklassenidentifikation

Der Gerätetyp beschreibt eine Klasse, der ein Gerät zugeordnet wird, wie z. B. Sensor, Umrichter, Antrieb, Kamerasystem oder IO-Baugruppe. Spezifische Objekte, Dienste und Protokolle eines Gerätetyps werden in einem Geräteprofil spezifiziert. Die Definition des Objektes "Geräteklassenidentifikation" sollte Bestandteil der Spezifikation der Kommunikationsschichten sein.

Anhand dieses Objektes kann der Master-Node erkennen, welche Parameter zu konfigurieren sind, welche Parameter zur Steuerung und welche zur Diagnose zur Verfügung stehen. Es kann auf mögliche Plug-and-Play-Varianten und Plug-and-Play-Stufen geschlossen werden und entsprechende Aktionen können eingeleitet werden.

10.4.2Geräteidentifikation

Dieses Objekt mit Bestandteilen wie Herstellerkennung, Revisionsnummer, Seriennummer ermöglicht eine eindeutige Identifikation des Gerätes. Diese eindeutige Identifikation ist insbesondere für Diagnosezwecke erforderlich.

10.4.3Dienste und Protokolle

Der Master-Node kann veranlassen, dass der Slave-Node die konfigurierten Parameterwerte in einem nicht-flüchtigen Speicher ablegt. Von dort werden die Werte bei Power on oder auf Anweisung des Master-Nodes in den aktiven Parametersatz übernommen. Diese Funktionalität bezieht sich auf Kommunikationsparameter, Geräteparameter bzw. Anwendungsparameter separat oder gemeinsam. Verfügt ein Slave-Node nicht über einen nicht-flüchtigen Speicher werden die Parameterwerte vom Master-Node in den Slave-Node geladen.

10.4.4Parametersynchronisation

Zur Wahrung der Konsistenz der Parameterwerte zwischen Master-Node und Slave-Node ist eine Synchronisation vorzusehen. Eine Werteänderung im Parametersatz des Slave-Nodes nach einer Synchronisation ist anzuzeigen (z. B. mittels Parametersatz-Identifikation), und eine erneute Synchronisation ist zu veranlassen (z. B. Lastdatenermittlung).

Diese Funktionalität bezieht sich auf Kommunikationsparameter, Geräteparameter bzw. Anwendungsparameter separat oder gemeinsam.

10.4.5Kommunikations-Dienste

Für die Realisierung der PnP-Funktionalität werden die in der Tabelle 5 spezifizierten Dienste von der Kommunikationsschicht vorausgesetzt.

Dienst	Slave-Node	Master-Node	Beschreibung
COM_Initialize	.req/.con	.req/.con	Initialisierung der PnP-Funktionalität
COM_Reset		.req/.con	Zurücksetzen der PnP-Funktionalität
COM_Activate_Node_Live_List		.req/.con	Aktivieren einer Live-List-Funktion für alle Teilnehmer im Netzwerk
COM_Deactivate_Node_Live_List		.req/.con	Deaktivieren einer Live-List-Funktion für alle Teilnehmer im Netzwerk
COM_Node_Live_List		.ind	Anzeigen von Teilnehmern im Netzwerk, wenn sie sich das erste Mal melden oder nicht mehr ansprechbar sind
COM_Read	.ind/.res	.req/.con	Lesen von Objekten
COM_Write	.ind/.res	.req/.con	Schreiben von Objekten
COM_Activate	.ind/.res	.req/.con	Aktivieren von Slave-Nodes

Tabelle 5 Dienste der Kommunikationsschicht

10.5 Beschreibung der Geräte- und Anwendungsobjekte

Zur Beschreibung der Objektstruktur der Geräte- und Anwendungsprofile werden XML-Schemata erarbeitet (Device Description Files). Aus den Schemata lassen sich sowohl Dokumente für die Spezifikation als auch Datenstrukturen und Parserfunktionalitäten für die Implementierung generieren. Außerdem sind die XML-Schemata die Basis für Editoren zur Erstellung der Objektbeschreibungen konkreter Geräte und Anwendungen (XML-Instanzen, Device Configuration Files).

11 Formale Beschreibung der PAPAS-PnP-Spezifikation

Im Rahmen der PAPAS-PnP-Spezifikation sind Kommunikationsobjekte, Kommunikationsdienste und Kommunikationsprotokolle (Kodierungen, Verhalten) zur Realisierung der PnP-Aufgaben zu definieren, die bei der Spezifikation der Geräte- und Anwendungsprofile zu berücksichtigen sind.
Die PAPAS-PnP-Spezifikation ist unabhängig von einem konkreten Kommunikationssystem zu erarbeiten. Objekte, Dienste, Verhaltensweisen und Kodierungen sollten, wo möglich, entsprechend der Zugehörigkeit zu Kommunikationsprofil, Geräteprofil bzw. Anwendungsprofil klassifiziert werden. Bei einigen Kommunikationsobjekten wird dies nicht möglich sein. Hier hängt die Klassifizierung ggf. davon ab, was durch ein konkretes Kommunikationssystem bereits unterstützt wird.
Letztendlich kann die Zuordnung der Bestandteile der PAPAS-PnP-Spezifikation zu den einzelnen Profil-Teilen erst bei der Abbildung auf ein konkretes Kommunikationssystem erfolgen.

11.1 Modell

11.1.1Systemarchitektur

Der Modellansatz berücksichtigt bezüglich der Speicherung der für das Plug-And-Play erforderlichen Objekte verschiedene Varianten. Die Daten werden entweder in den Slave-Nodes nicht-flüchtig abgelegt oder sie können im Master-Node (Abbildung 7) oder in einem zentraler Server (Abbildung 8) gespeichert werden.

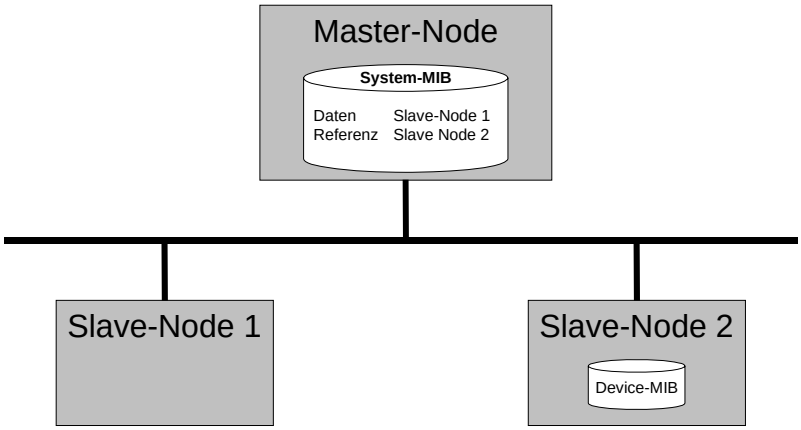


Abbildung 7: Plug-and-Play-Systemarchitektur mit nicht-flüchtiger Speicherung der Objekte im Master-Node und optional in Slave-Nodes

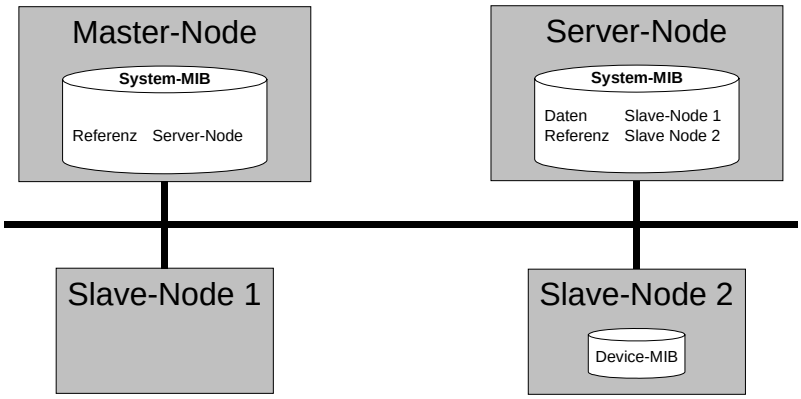


Abbildung 8: Plug-and-Play-Systemarchitektur mit nicht-flüchtiger Speicherung der Objekte in einem zentralen PnP-Server-Node und optional in Slave-Nodes

11.2 Master-Node

In der Abbildung 9 ist die Architektur des Master-Nodes dargestellt. Die PnP-Funktionalität im Master-Node wird mit den Zustandsmaschinen Scheduler, Device Exchange, Slave-Node Handler und Diagnosis realisiert. Die Zustandsmaschinen agieren untereinander und mit ihrer Umgebung über eine asynchrone Dienstschnittstelle.

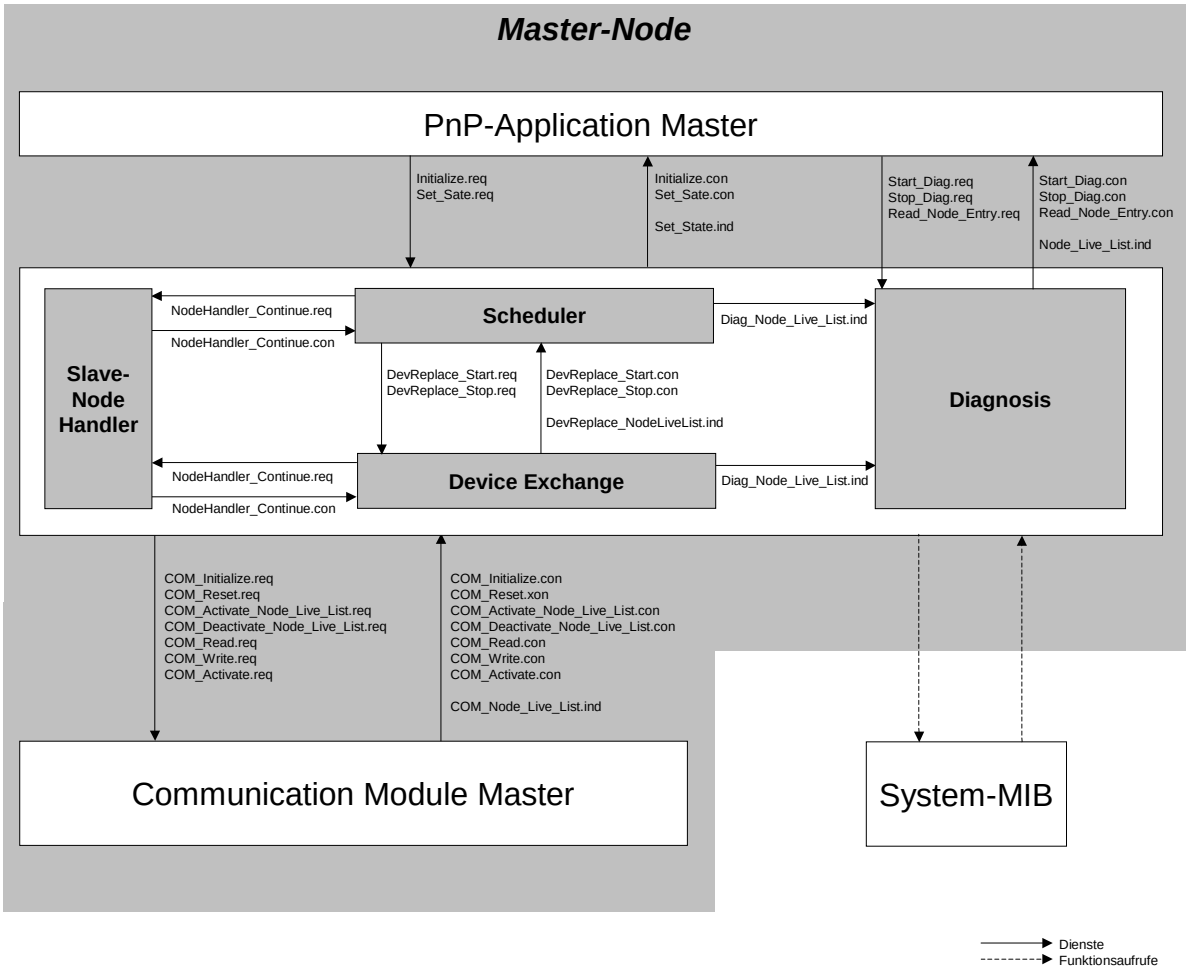


Abbildung 9: Architektur Master-Node

11.2.1Scheduler

In der Zustandsmaschine Scheduler ist die zentrale Verwaltung der PnP-Funktionalität definiert. Der Scheduler hat die drei Hauptzustände Offline, Setup und Operation. Diese Zustände können vom User gesetzt werden (Set_State). Im Zustand Offline ist die PnP-Funktionalität abgeschaltet. Wenn der Master-Node Gerätefunktionalität besitzt, arbeitet er wie ein Slave-Node.
Nach erfolgreicher Initialisierung des Master-Nodes (Initialize) wird automatisch der Zustand Setup erreicht. Im Zustand Setup ist die PnP-Funktionalität eingeschaltet. Der Master-Node wird vom Kommunikationsmodul ständig über Änderungen bezügliche der ansprechbaren Teilnehmer informiert (COM_Node_Live_List). Diese Teilnehmer werden durch den Node Handler (Node_Handler_Continue) verwaltet.
Die zu bearbeiteten Slave-Nodes können durch den User (Initialize) eingeschränkt werden. Nur die angegebenen Slave-Nodes werden auch tatsächlich vom Kommunikationsmodule angezeigt. Die Adressierung der Slave-Nodes erfolgt über eine Identifikation, die in einem hier nicht definierten Zusammenhang mit der physikalischen Adresse steht. Dieser Zusammenhang wird in dem für ein konkretes Kommunikationssystem zu spezifizierendes Anpassungsmodul definiert.
Der Zustand Setup kann mit dem Dienst Set_State verlassen werden, um in den Zustand Operation zu wechseln. Der Zustand Operation steht für den Produktivdatenverkehr. Die Steuerung der PnP_Funktionalität wird dann von der Zustandsmaschine Device Exchange übernommen.

11.2.2Device Echange

Im Zustand Operation der Zustandsmaschine Scheduler wird die Zustandsmaschine Device Echange gestartet (DevReplace_Start). Im Device Exchange wird im wesentlichen zwischen den Unterzuständen Operation_All und Operation_Partly gewechselt. Operation_All bedeutet, das alle in der verteilten Anwendung parametrierten Geräte vorhanden sind und fehlerfrei arbeiten. Operation_Partly ist der komplementäre Zustand. Unter Verwendung des Slave Node Handlers wird, angestoßen durch eine Änderung der ansprechbaren Slave-Nodes, versucht, immer wieder den Zustand Operation_All zu erreichen.
Slave-Nodes, die nicht in der verteilten Anwendung parametriert sind, werden ebenfalls bearbeitet und über Diagnosis dem User angezeigt.
Prinzipiell arbeitet die Zustandsmaschine Device Exchange ähnlich wie die des Schedulers. Sie benutzen beide den Slave-Node Handler in der gleichen Art und Weise. Im Scheduler steht die Inbetriebnahme im Vordergrund. Ständige Wechsel bei den ansprechbaren Slave-Nodes und hauptsächlich asynchroner Busverkehr zur Parametrierung werden hier erwartet. Beim Devices Exchange befindet sich die Anlage im isochronen (zyklischen) Datenverkehr. Asynchroner Datenverkehr zur Parametrierung ist hier die Ausnahme. Eine der typischen Ausnahmen ist der Gerätetausch.

11.2.3Slave-Node Handler

Der Node Handler arbeitet jeweils einen ansprechbaren Teilnehmer (Slave-Node) ab, initiiert durch den Systemanlauf oder durch den Gerätetausch. In einer Folge von Lese- und Schreibzugriffen (COM_Read, COM_Write) werden gerätespezifische Objekte gelesen und beschrieben. Letztendlich erfolgt eine Zuordnung des Slave-Nodes zum Anwendungs-Parametersatz des Master-Nodes. Wenn die Zuordnung erfolgreich hergestellt werden konnte, wird der Slave-Node aktiviert (COM_Activate). Alle für diesen Vorgang benötigten Daten (Objekte) werden der System-MIB entnommen bzw. dort aktualisiert.
Diagnosis
Die Zustandmaschine Diagnosis ist optional. Sie kann durch den User zugeschaltet werden. Läuft die Diagnose wird der User kontinuierlich über den Zustand der im Netz befindlichen Nodes informiert. Über Read-Dienste können alle vorhandenen Informationen über einen Slave-Node ausgelesen werden.

11.2.4PnP-Application Master

Die PnP-Application Master repräsentiert die spezielle Plug-and-Play-Anwendung des Master-Nodes. Die Anwendung definiert sich durch die Funktion des eigentlichen Gerätes kann aber auch Anwendungsfunktionen zur Plug-and-Play-Funktionalität haben wie z. B. eine erforderliche Nutzerinteraktion.

11.2.5Communication Module Master

Das Communication Module Master fasst das verwendete Kommunikationsprotokoll zusammen. Da das Kommunikationsprotokoll offen ist, werden in Abschnitt 1.1.5.6 Anforderungen definiert, die das Protokoll für die Plug-and-Play-Funktionalität zur Verfügung stellen muss.

11.2.6System-MIB

Die Kommunikationsobjekte des Master-Nodes befinden sich in der dargestellten System-MIB (Management Information Base). Für den Zugriff auf die System-MIB wird eine synchrone Funktionsschnittstelle definiert. Der Inhalt der System-MIB bleibt damit transparent. Die Objekte werden im folgenden Abschnitt beschrieben.

11.2.7Geräte-Parameter-Beschreibung

Object 0x1000: Node-Type

Der Node-Typ wird als Bestandteil des Objektes Device-Type kodiert. Dazu wird das Bit 24 verwendet.

Object Description

Index	**0x1000**
Name	Node Type
Object Code	VAR
Data Type	UNSIGNED1
Category	Mandatory

Entry Description

Access	ro
Value Range	UNSIGNED1
Default Value	1

Coding

Code	Node Type
0	Slave-Node
1	Master-Node

11.2.8Anwendungs-Parameter-Beschreibung

Object 0xA100: Master-Node-Priorität

Die Master-Node-Priorität wird dazu genutzt, um in einem System mit mehreren Geräten, die Master-Node-Funktionalität implementiert haben, den Master-Node festzulegen. So könnte in einem Rapid-Prototyping-PC die Master-Node-Priorität in der Entwicklungsphase herauf gesetzt und anschließend wieder herab gesetzt werden. Die höchste Priorität hat der Master-Node mit dem Wert 0x00 die niedrigste der Master-Node mit dem Wert 0xFF.

Object Description

Index	**0xA100**
Name	Master-Node-Priorität
Object Code	VAR
Data Type	UNSIGNED8
Category	Mandatory

Entry Description

Access	ro
Value Range	UNSIGNED8
Default Value	0xFF

Object 0xA101: Application Parameter Set Description

In diesem Objekt wird die Beschreibung der Anwendungsobjekte abgelegt.

Object Description

Index	**0xA101**
Name	Application Parameter Set Description
Object Code	RECORD
Data Type	Application Profile Specific
Category	Optional

Entry Description

Access	ro
Value Range	Application Profile Specific
Default Value	Application Profile Specific

Object 0xA102: Application Parameter Set

In diesem Objekt wird die konkrete Wertebelegung der Anwendungsobjekte abgelegt.

Object Description

Index	**0xA102**
Name	Application Parameter Set
Object Code	RECORD
Data Type	Application Profile Specific
Category	Optional

Entry Description

Access	rw
Value Range	Application Profile Specific
Default Value	Application Profile Specific

Object 0xA103: Application Parameter Set Identification

Die Anwendungs-Parametersatz-Identifikation adressiert einen Anwendungs-Parametersatz für einen bestimmten Slave-Node im verteilten Netzwerk.

Object Description

Index	**0xA103**
Name	Application Parameter Set Identification
Object Code	VAR
Data Type	UNSIGNED16
Category	Optional

Entry Description

Access	rw
Value Range	UNSIGNED16
Default Value	0x0000

Object 0xA104: Parameter Value Relationchip

In diesem Objekt ist die Beziehung zwischen Anwendungs-Parameterwerten und Geräte-Parameterwerten abgelegt.

Object Description

Index	**0xA104**
Name	Parameter Value Relationchip
Object Code	RECORD
Data Type	Application Profile Specific
Category	Optional

Entry Description

Access	rw
Value Range	Application Profile Specific
Default Value	Application Profile Specific

Object 0xA105: Lastdaten

Diese Struktur beinhaltet Parameter wie Masse, Schwerpunkt und Trägheitstensor.

Object Description

Index	**0xA105**
Name	Lastdaten
Object Code	RECORD
Data Type	Application Profile Specific
Category	Optional

Entry Description

Access	rw
Value Range	Application Profile Specific
Default Value	Application Profile Specific

11.2.9Funktionen

In der folgenden Tabelle sind die Funktionen aufgeführt, die von der System-MIB bereitgestellt werden. Die System-MIB kann lokal im Master-Node verfügbar sein oder auf einem anderen Gerät, so dass Remote-Zugriffe erforderlich sind.

Rückgabe	Funktion	Übergabe	Beschreibung
	SysMib_Connect	Init_Data: Initialisierungsdaten (nicht näher spezifiziert)	Verbinden mit der SysMib, SysMib enthält ein Daten-Objekt für den Master-Node und eine Liste mit Daten-Objekten für alle Slave-Nodes mit flüchtigen Speicher
	SysMib_Disconnect		Trennen von der SysMib
	SysMib_SetSystemState	Systemzustand(Setup, Operation_Partly, Operation_All)	Setzen der Systemzustände

Rückgabe	Funktion	Übergabe	Beschreibung
Systemzustand(Setup, Operation_Partly, Operation_All)	SysMib_GetSystemState		Lesen der Systemzustände
	SysMib_SetNodeSate	Node_Entry_Id: aktueller Slave-Node; State(Offline, Configured, Active)	Setzen der Zustände der Slave-Nodes Offline, Configured, Active
	SysMib_CreateDynNodeEntry	Node_Entry_Id	Anlegen eines dynamischen Eintrages für einen Slave-Node
	SysMib_DestructDynNodeEntry	Node_Entry_Id	Freigeben eines dynamischen Eintrages für einen Slave-Node
	SysMib_SetDevSheet	Node_Entry_Id; Geräte-Datenblatt	Schreiben des Geräte-Datenblattes für einen Slave-Node
	SysMib_SetModes	Node_Entry_Id; Unterstützte Betriebsmodi	Schreiben der unterstützten Betriebsmodi für einen Slave-Node
	SysMib_SetPrmDescr	Node_Entry_Id; Geräte-Parametersatz-Beschreibung	Schreiben der Geräte-Parametersatz-Beschreibung für einen Slave-Node
	SysMib_SetPrm	Node_Entry_Id; Geräte-Parametersatz	Schreiben des Geräte-Parametersatzes für einen Slave-Node
	SysMib_SetPrmId	Node_Entry_Id; Geräte-Parametersatz-Identifikation	Schreiben der Geräte-Parametersatz-Identifikation für einen Slave-Node
	SysMib_SetDevDrv	Node_Entry_Id; Unterstützte Geräte-Treiber	Schreiben der unterstützten Geräte-Treiber für einen Slave-Node
	SysMib_SetDevDrvId	Node_Entry_Id; Unterstützte Geräte-Treiber-Identifikation	Schreiben der unterstützte Geräte-Treiber-Identifikation für einen Slave-Node
	SysMib_Reconfig	Node_Entry_Id	Rekonfiguration der Anwendungs-Parameter anhand der Geräte-Parameter für einen Slave-Node
	SysMib_DetermineDevParam		Bestimmen der Geräte-Parameter für den Master-Node (Bandbreiten)
TRUE, FALSE	SysMib_ExpectedConfiguration Complete		Ermitteln, ob die Ist-Konfiguration dem Anwendungsparametersatz entspricht

11.2.10 Scheduler

Dienst-Beschreibung

Initialisierung des Master-Nodes

Mit dem Dienst *Initialize* wird die PnP-Funktionalität des Master-Nodes initialisiert. Während der Initialisierung wird eine Verbindung zur System-MIB (Management Information Base) aufgebaut, in der alle anwendungsspezifischen Objekte gespeichert sind. Der Zugriff auf die System-MIB erfolgt transparent über eine Funktionsschnittstelle.

Parametername	.req	.con
Argument	M	
Init_Data	M	
Result(+)		S
Status		M
Result(-)		S
Status		M

Tabelle 1 Dienst Initialize

Argument

Die Argumente übermitteln die dienstspezifischen Parameter einer Dienstanforderung.

Init_Data

Dieser Parameter enthält eine Struktur mit Initialisierungsdaten für die PnP-Funktionalität des Master-Node.

Init_Data.Node_Live_List

Mit diesem Struktur-Element werden Slave-Nodes konfiguriert, die in der PnP-Funktionalität des Master-Nodes berücksichtigt werden. Nur die konfigurierten Slave-Nodes werden vom Master-Node erfasst und bearbeitet. Die Identifizierung der Slave-Nodes erfolgt über eine eindeutige Identifikation, die beispielsweise den physikalischen Adressen entsprechen können.

Result (+)

Diese Parameter zeigen die positive Anwort auf die Dienstanforderung an.

Status

Der Parameter *Status* zeigt an, das der Dienst erfolgreich ausgeführt wurde.

Status	Bedeutung
OK	Dienst wurde erfolgreich ausgeführt

Result (-):

Diese Parameter zeigen die negative Anwort auf die Dienstanforderung an.

Status:

Dieser Parameter zeigt einen Fehlercode an.

Status	Bedeutung
NO	Dienst konnte nicht ausgeführt werden

Setzen des Zustandes im Master-Node

Mit dem Dienst *Set_State* wird der Zustand im Master-Node geschaltet bzw. angezeigt.

Parametername	.req	.ind	.con
Argument	M	M	
State	M	M	
Result(+)			S
Status			M
Result(-)			S
Status			M

Tabelle 2 Dienst Set_State

Argument

Die Argumente übermitteln die dienstspezifischen Parameter einer Dienstanforderung.

State

Dieser Parameter enthält den Zustand des Master-Nodes.

State	Bedeutung	Gültigkeit
Offline	PnP-Funktionalität ist ausgeschaltet	Set-State.req Set-State.ind
Setup	PnP-Funktionalität ist eingeschaltet	Set-State.req Set-State.ind
Operation	PnP-Funktionalität eingeschaltet;	Set-State.req

State	Bedeutung	Gültigkeit
	Nutzdatenbetrieb läuft	
Operation_All	Nutzdatenbetrieb läuft; Automatisierungsfunktion vollständig	Set-State.ind
Operation_Partly	Nutzdatenbetrieb läuft; Automatisierungsfunktion unvollständig	Set-State.ind

Result (+)

Diese Parameter zeigen die positive Anwort auf die Dienstanforderung an.

Status

Der Parameter *Status* zeigt an, das der Dienst erfolgreich ausgeführt wurde.

Status	Bedeutung
OK	Dienst wurde erfolgreich ausgeführt

Result (-):

Diese Parameter zeigen die negative Anwort auf die Dienstanforderung an.

Status:

Dieser Parameter zeigt einen Fehlercode an.

Status	Bedeutung
NO	Dienst konnte nicht ausgeführt werden

Protokoll-Beschreibung

#	Current State	Event /Condition =>Action	Next State
1	OFFLINE	**Initialize.req(Init_Data)** => COM_Initialize.req	COM_INIT
2	COM_INIT	**COM_Initialize.con(Status)** /Status = OK => COM_Activate_Node_Live_List.req(Node_Live_List = Init_Data.NodeLiveList)	ACT_NODE_LIVE_LIST
3	COM_INIT	**COM_Initialize.con(Status)** /Status <> OK => Initialize.con(Status)	OFFLINE
4	ACT_NODE_LIVE_LIST	**COM_Activate_Node_Live_List.con(Status)** /Status = OK => SysMib_Connect(Init_Data); SysMib_SetSystemState(Setup); Initialize.con(Status = OK) Set_State.ind(State = Setup)	SETUP
5	ACT_NODE_LIVE_LIST	**COM_Activate_Node_Live_List.con(Status)** /Status <> OK => SysMib_Disconnect(); Initialize.con(Status)	OFFLINE
6	DEACT_NODE_LIVE_LIST	**COM_Deactivate_Node_Live_List.con** => COM_Reset.req	COM_RESET

#	Current State	Event /Condition =>Action	Next State
7	COM_RESET	**COM_Reset.con** => SysMib_Disconnect(); SysMib_SetSystemState(Offline); Set_State.ind(State = Offline)	OFFLINE
8	any-state	**Set_State.req(State)** /State = Offline => Set_State.con(Status = OK) DevReplace_Stop.req	STOP_DEV_REPLACE
9	STOP_DEV_REPLACE	**DevReplace_Stop.con** => COM_Deactivate_Node_Live_List.req	DEACT_NODE_LIVE_LIST
10	any-state	**Set_State.req(State)** /State = Setup => Set_State.con(Status = OK) DevReplace_Stop.req	SE_STOP_DEV_REPLACE
11	SE_STOP_DEV_REPLACE	**DevReplace_Stop.con** => SysMib_SetSystemState(Setup); Set_State.ind(State = Setup)	SETUP
12	any-state	**Set_State.req(State)** /State = Operation && SysMib_ExpectedConfigurationComplete() = TRUE => SysMib_DetermineDevParam(); Set_State.con(Status = OK) Set_State.ind(State = Operation_All) DevReplace_Start.req(State = Operation_All)	START_DEV_REPLACE
13	any-state	**Set_State.req(State)**/State = Operation && SysMib_ExpectedConfigurationComplete() = FALSE=>SysMib_DetermineDevParam();Set_State.con(Status = OK)Set_State.ind(State = Operation_Partly)DevReplace_Start.req(State = Operation_Partly)	START_DEV_REPLACE
14	START_DEV_REPLACE	**DevReplace_Start.con(State, Status)** /Status = OK => SysMib_SetSystemState(Operation); Set_State.con(Status) Set_State.ind(State)	OPERATION
15	START_DEV_REPLACE	**DevReplace_Start.con(State, Status)** /Status <> OK => SysMib_SetSystemState(Setup); Set_State.con(Status) Set_State.ind(State = Setup)	SETUP
16	SETUP	**COM_Node_Live_List.ind(Node_Entry_Id, Node_Entry_State)** /Node_Entry_State = Setup => COM_Read.req(Index = 0x1000)	CHK_NODE_TYPE
17	CHK_NODE_TYPE	**COM_Read.con(Index, Data, Status)** /Status = OK && Node-Entry ist Master-Node => COM_Read.req(Index = 0xA100)	READ_PRIO
18	READ_PRIO	**COM_Read.con(Index, Data, Status)** /Status = OK && Master-Node mit einer niedrigeren Master-Node-Priorität =>	SETUP

#	Current State	Event /Condition =>Action	Next State
19	READ_PRIO	**COM_Read.con(Index, Data, Status)** /Status = OK && Master-Node mit einer höherer Master-Node-Priorität => Master-Node zieht sich zurück oder verhält sich nur noch wie ein Slave-Node (siehe Zustandsmaschine für Slave-Node) COM_Deactivate_Node_Live_List.req	DEACT_NODE_LIVE_LIST
20	CHK_NODE_TYPE	**COM_Read.con(Index, Data, Status)** /Status = OK && Node-Entry ist Slave-Node && Node mit Node_Entry_Id existiert nicht in dynamischer System-MIB => SysMib_CreateDynNodeEntry(Node_Entry_Id); SysMib_SetNodeSate(Node_Entry_Id, Offline); NodeHandler_Continue.req(Node_Entry_Id)	SETUP
21	CHK_NODE_TYPE	**COM_Read.con(Index, Data, Status)** /Status = OK && Node-Entry ist Slave-Node && Slave-Node in dyn. SysMib enthalten => SysMib_SetNodeSate(Node_Entry_Id, Offline); NodeHandler_Continue.req(Node_Entry_Id)	SETUP
22	SETUP	**COM_Node_Live_List.ind(Node_Entry_Id, Node_Entry_State)** /Node_Entry_State = Free => SysMib_DestructDynNodeEntry(Node_Entry_Id); Diag_Node_Live_List.ind(Node_Entry_Id, Entry_State)	SETUP
23	SETUP	**NodeHandler_Continue.con(Node_Entry_Id, Status)** /Status = OK => Diag_Node_Live_List.ind(Node_Entry_Id, Entry_State = Activated)	SETUP
24	SETUP	**NodeHandler_Continue.con(Node_Entry_Id, Status)** /Status <> OK => SysMib_DestructDynNodeEntry(Node_Entry_Id); Diag.ind(Node_Entry_Id, Status)	SETUP
25	OPERATION	**COM_Node_Live_List.ind(Node_Entry_Id, Node_Entry_State)** /Node_Entry_State = Setup => COM_Read.req(Index = 0x1000)	OP_CHK_NODE_TYPE
26	OPERATION	**COM_Node_Live_List.ind(Node_Entry_Id, Node_Entry_State)** /Node_Entry_State = Free => DevReplace_Node_Live_List.ind(Node_Entry_Id, Node_Entry_State)	OPERATION
27	OP_CHK_NODE_TYPE	**COM_Read.con(Index, Data, Status)** /Status = OK && Node-Entry ist Master-Node => COM_Read.req(Index = 0xA100)	OP_READ_PRIO
28	OP_CHK_NODE_TYPE	**COM_Read.con(Index, Data, Status)** /Status = OK && Node-Entry ist Slave-Node && Slave-Node in dyn, SysMib enthalten => SysMib_CreateDynNodeEntry(Node_Entry_Id); SysMib_SetNodeSate(Node_Entry_Id, Offline); DevReplace_Node_Live_List.ind(Node_Entry_Id, Node_Entry_State)	OPERATION

#	Current State	Event /Condition =>Action	Next State
29	OP_CHK_NODE_TYPE	**COM_Read.con(Index, Data, Status)** /Status = OK && Node-Entry ist Slave-Node && Slave-Node in dyn. SysMib nicht enthalten => SysMib_SetNodeSate(Node_Entry_Id, Offline); DevReplace_Node_Live_List.ind(Node_Entry_Id, Node_Entry_State)	OPERATION
30	OP_READ_PRIO	**COM_Read.con(Index, Data, Status)** /Status = OK && Master-Node mit einer niedrigeren Master-Node-Priorität =>	OPERATION
31	OP_READ_PRIO	**COM_Read.con(Index, Data, Status)** /Status = OK && Master-Node mit einer höherer Master-Node-Priorität => Master-Node zieht sich zurück oder verhält sich nur noch wie ein Slave-Node (siehe Zustandsmaschine für Slave-Node) COM_Deactivate_Node_Live_List.req	DEACT_NODE_LIVE_LIST

11.2.11 Device Exchange

Dienst-Beschreibung

Starten des Gerätetausches

Mit dem Dienst *DevReplace_Start* wird die Zustandsmaschine für den Gerätetausch gestartet.

Parametername	.req	.con
Argument	M	
State	M	
Result(+)		S
Status		M
Result(-)		S
Status		M

Tabelle 3 Dienst DevReplace_Start

Argument

Die Argumente übermitteln die dienstspezifischen Parameter einer Dienstanforderung.

State

Dieser Parameter enthält den Unterzustand des Master-Nodes im Zustand *Operation*.

State	Bedeutung	Gültigkeit
Operation_All	Nutzdatenbetrieb läuft; Automatisierungsfunktion vollständig	Set-State.ind
Operation_Partly	Nutzdatenbetrieb läuft; Automatisierungsfunktion unvollständig	Set-State.ind

Result (+)

Diese Parameter zeigen die positive Anwort auf die Dienstanforderung an.

Status

Der Parameter *Status* zeigt an, das der Dienst erfolgreich ausgeführt wurde.

Status	Bedeutung
OK	Dienst wurde erfolgreich ausgeführt

Result (-):

Diese Parameter zeigen die negative Anwort auf die Dienstanforderung an.

Status:

Dieser Parameter zeigt einen Fehlercode an.

Status	Bedeutung
NO	Dienst konnte nicht ausgeführt werden

Stoppen des Gerätetausches

Mit dem Dienst *DevReplace_Stop* wird die Zustandsmaschine für den Gerätetausch gestoppt.

Parametername	.req	.con
Argument	M	
Result(+)		S
Status		M
Result(-)		S
Status		M

Tabelle 4 Dienst DevReplace_Stop

Argument

Die Argumente übermitteln die dienstspezifischen Parameter einer Dienstanforderung.

Result (+)

Diese Parameter zeigen die positive Anwort auf die Dienstanforderung an.

Status

Der Parameter *Status* zeigt an, das der Dienst erfolgreich ausgeführt wurde.

Status	Bedeutung
OK	Dienst wurde erfolgreich ausgeführt

Result (-):

Diese Parameter zeigen die negative Anwort auf die Dienstanforderung an.

Status:

Dieser Parameter zeigt einen Fehlercode an.

Status	Bedeutung
NO	Dienst konnte nicht ausgeführt werden

Anzeigen einer Änderung in der Node Live List

Mit dem Dienst *DevReplace_Node_Live_List* wird eine Änderung in der Live List der im Netzwerk befindlichen Nodes angezeigt.

Parametername	.ind
Argument	M
Node_Entry_Id	M
Node_Entry_State	M

Tabelle 5 Dienst DevReplace_Node_Live_List

Argument

Die Argumente übermitteln die dienstspezifischen Parameter einer Dienstanforderung.

Node_Entry_Id

Dieser Parameter enthält die Identifikation eines Slave-Nodes.

Node_Entry_State

Dieser Parameter enthält den Zustand eines Slave-Nodes.

State	Bedeutung
Free	Identifiaktion frei; Slave-Node nicht mehr erreichbar
Setup	Identifiaktion besetzt; Slave-Node ansprechbar

Anzeigen des Zustandes im Master-Node

Mit dem Dienst *Set_State* wird der Zustand im Master-Node angezeigt.

Parametername	.ind
Argument	M
State	M

Tabelle 6 Dienst Set_State

Argument

Die Argumente übermitteln die dienstspezifischen Parameter einer Dienstanforderung.

State

Dieser Parameter enthält den Unterzustand des Master-Nodes im Zustand *Operation*.

State	Bedeutung	Gültigkeit
Operation_All	Nutzdatenbetrieb läuft; Automatisierungsfunktion vollständig	Set-State.ind
Operation_Partly	Nutzdatenbetrieb läuft; Automatisierungsfunktion unvollständig	Set-State.ind

Protokoll-Beschreibung

#	Current State	Event /Condition =>Action	Next State
1	STOP	**DevReplace_Start.req(State)** /State = Operation_All => DevReplace_Start.con(Status = OK)	OPERATION_ALL

#	Current State	Event /Condition =>Action	Next State
2	STOP	**DevReplace_Start.req(State)** /State = Operation_Partly => DevReplace_Start.con(Status = OK)	OPERATION_PARTLY
3	OPERATION_ALL	**DevReplace_Start.req(State)** => DevReplace_Start.con(Status = OK)	OPERATION_ALL
4	OPERATION_PARTLY	**DevReplace_Start.req(State)** => DevReplace_Start.con(Status = OK)	OPERATION_PARTLY
5	OPERATION_ALL	**DevReplace_Stop.req** => DevReplace_Stop.con(Status = OK)	STOP
6	OPERATION_PARTLY	**DevReplace_Stop.req** => DevReplace_Stop.con(Status = OK)	STOP
7	STOP	**DevReplace_Stop.req** => DevReplace_Stop.con(Status = OK)	STOP
8	OPERATION_ALL	**DevReplace_Node_Live_List.ind(Node_Entry_Id, Node_Entry_State)** /Node_Entry_State = Free && Slave-Node im Projekt nicht enthalten => SysMib_DestructDynNodeEntry(Node_Entry_Id); Diag_Node_Live_List.ind(Node_Entry_Id, Entry_State)	OPERATION_ALL
9	OPERATION_ALL	**DevReplace_Node_Live_List.ind(Node_Entry_Id, Node_Entry_State)** /Node_Entry_State = Free && Slave-Node im Projekt enthalten => SysMib_DestructDynNodeEntry(Node_Entry_Id); Diag_Node_Live_List.ind(Node_Entry_Id, Entry_State) Set_State.ind(State = Operation_Partly)	OPERATION_PARTLY
10	OPERATION_ALL	**DevReplace_Node_Live_List.ind(Node_Entry_Id, Node_Entry_State)** /Node_Entry_State = Setup => SysMib_CreateDynNodeEntry(Node_Entry_Id); NodeHandler_Continue.req(Node_Entry_Id)	OPERATION_ALL
12	OPERATION_PARTLY	**DevReplace_Node_Live_List.ind(Node_Entry_Id, Node_Entry_State)** /Node_Entry_State = Free => SysMib_DestructDynNodeEntry(Node_Entry_Id); Diag_Node_Live_List.ind(Node_Entry_Id, Entry_State)	OPERATION_PARTLY
14	OPERATION_PARTLY	**DevReplace_Node_Live_List.ind(Node_Entry_Id, Node_Entry_State)** /Node_Entry_State = Setup => SysMib_CreateDynNodeEntry(Node_Entry_Id); NodeHandler_Continue.req(Node_Entry_Id)	OPERATION_PARTLY
16	OPERATION_ALL	**NodeHandler_Continue.con(Node_Entry_Id, Status)** /Status = OK && SysMib_ExpectedConfigurationComplete() = TRUE => Diag_Node_Live_List.ind(Node_Entry_Id, Entry_State)	OPERATION_ALL

#	Current State	Event /Condition =>Action	Next State
17	OPERATION_ALL	**NodeHandler_Continue.con(Node_Entry_Id, Status)** /Status = OK && SysMib_ExpectedConfigurationComplete() = FALSE => Diag_Node_Live_List.ind(Node_Entry_Id, Entry_State) Set_State.ind(State = Operation_Partly)	OPERATION_PARTLY
18	OPERATION_PARTLY	**NodeHandler_Continue.con(Node_Entry_Id, Status)** /Status = OK && SysMib_ExpectedConfigurationComplete() = TRUE => Diag_Node_Live_List.ind(Node_Entry_Id, Entry_State)Set_State.ind(State = Operation_All)	OPERATION_ALL
19	OPERATION_PARTLY	**NodeHandler_Continue.con(Node_Entry_Id, Status)** /Status = OK && SysMib_ExpectedConfigurationComplete() = FALSE => Diag_Node_Live_List.ind(Node_Entry_Id, Entry_State)	OPERATION_PARTLY
20	OPERATION_ALL	**NodeHandler_Continue.con(Node_Entry_Id, Status)** /Status <> OK => SysMib_DestructDynNodeEntry(Node_Entry_Id); Diag_Node_Live_List.ind(Node_Entry_Id, Entry_State) Set_State.ind(State = Operation_All)	OPERATION_ALL
21	OPERATION_PARTLY	**NodeHandler_Continue.con(Node_Entry_Id, Status)** /Status <> OK => SysMib_DestructDynNodeEntry(Node_Entry_Id); Diag_Node_Live_List.ind(Node_Entry_Id, Entry_State) Set_State.ind(State = Operation_Partly)	OPERATION_PARTLY

11.2.12 Slave-Node Handler

Dienst-Beschreibung

Node Handler abarbeiten

Mit dem Dienst *NodeHandler_Continue* wird die Zustandsmaschine für den Node Handler abgearbeitet.

Parametername	.req	.con
Argument	M	M
Node_Entry_Id	M	M
Result(+)		S
Status		M
Result(-)		S
Status		M

Tabelle 7 Dienst NodeHandler_Continue

Argument

Die Argumente übermitteln die dienstspezifischen Parameter einer Dienstanforderung.

Node_Entry_Id

Dieser Parameter enthält die Identifikation eines Slave-Nodes.

Result (+)

Diese Parameter zeigen die positive Anwort auf die Dienstanforderung an.

Status

Der Parameter *Status* zeigt an, das der Dienst erfolgreich ausgeführt wurde.

Status	Bedeutung
OK	Dienst wurde erfolgreich ausgeführt

Result (-):

Diese Parameter zeigen die negative Anwort auf die Dienstanforderung an.

Status:

Dieser Parameter zeigt einen Fehlercode an.

Status	Bedeutung
NO	Dienst konnte nicht ausgeführt werden

Protokoll-Beschreibung

#	Current State	Event /Condition =>Action	Next State
1	IDLE	**NodeHandler_Continue.req(Node_Entry_Id)** => SysMib_SetNodeState(Node_Entry_Id, Offline); COM_Read.req(Index = 0x1002)	READ_CFG_STATE
2	READ_CFG_STATE	**COM_Read.con(Index, Data, Status)** /Status = OK && Werte des Slave-Node in dyn. SysMib enthalten =>	CHECK_CFG_DYN
3	READ_CFG_STATE	**COM_Read.con(Index, Data, Status)** /Status = OK && Werte des Slave-Node in dyn. SysMib nicht enthalten =>	CHECK_CFG
4	CHECK_CFG_DYN	/Konfigurations-Zustand = konfiguriert && Slave-Node im Projekt enthalten und identisch => SysMib_SetNodeState(Node_Entry_Id, Configured); COM_Activate.req(Slave-Node)	SET_ACTIVATE
5	CHECK_CFG_DYN	/Konfigurations-Zustand = konfiguriert && Slave-Node im Projekt nicht enthalten oder nicht identisch => SysMib_Reconfig(Node_Entry_Id);	WRITE_CFG
6	CHECK_CFG_DYN	/Konfigurations-Zustand = nicht konfiguriert && Slave-Node im Projekt enthalten und kompatibel =>	WRITE_CFG
7	CHECK_CFG_DYN	/Konfigurations-Zustand = nicht konfiguriert && Slave-Node im Projekt nicht enthalten oder nicht kompatible => SysMib_Reconfig(Node_Entry_Id);	WRITE_CFG
8	CHECK_CFG_DYN	/Konfigurations-Zustand = keine Konfiguration verfügbar && Slave-Node im Projekt enthalten und kompatible =>	WRITE_CFG

#	Current State	Event /Condition =>Action	Next State
9	CHECK_CFG_DYN	/Konfigurations-Zustand = keine Konfiguration verfügbar && Slave-Node im Projekt nicht enthalten oder nicht kompatible => SysMib_Reconfig(Node_Entry_Id);	WRITE_CFG
10	CHECK_CFG	/Konfigurations-Zustand <> keine Konfiguration verfügbar => COM_Read.req(Index = 0xA000)	READ_DATA_SHEET
11	CHECK_CFG	/Konfigurations-Zustand = keine Konfiguration verfügbar && Slave-Node im Projekt enthalten =>	WRITE_CFG
12	CHECK_CFG	/Konfigurations-Zustand = keine Konfiguration verfügbar && Slave-Node in Projekt nicht enthalten => NodeHandler_Continue.con(Node_Entry_Id, Status = NO)	IDLE
13	READ_DATA_SHEET	**COM_Read.con(Index, Data, Status)**/Status = OK=>SysMib_SetDevSheet(Node_Entry_Id, Geräte-Datenblatt);COM_Read.req(Index = 0xA001)	READ_MODES
14	READ_MODES	**COM_Read.con(Index, Data, Status)** /Status = OK => SysMib_SetModes(Node_Entry_Id, Unterstützte Betriebsmodi); COM_Read.req(Index = 0xA002)	READ_PRM_DESCR
15	READ_PRM_DESCR	**COM_Read.con(Index, Data, Status)** /Status = OK => SysMib_SetPrmDescr(Node_Entry_Id, Geräte-Parametersatz-Beschreibung); COM_Read.req(Index = 0xA003)	READ_PRM
16	READ_PRM	**COM_Read.con(Index, Data, Status)** /Status = OK => SysMib_SetPrm(Node_Entry_Id, Geräte-Parametersatz); COM_Read.req(Index = 0xA004)	READ_PRM_ID
17	READ_PRM_ID	**COM_Read.con(Index, Data, Status)** /Status = OK => SysMib_SetPrmId(Node_Entry_Id, Geräte-Parametersatz-Identifikation); COM_Read.req(Index = 0xA005)	READ_DEV_DRV
18	READ_DEV_DRV	**COM_Read.con(Index, Data, Status)** /Status = OK => SysMib_SetDevDrv(Node_Entry_Id, Unterstützte Geräte-Treiber); COM_Read.req(Index = 0xA006)	READ_DEV_DRV_ID
19	READ_DEV_DRV_ID	**COM_Read.con(Index, Data, Status)** /Status = OK => SysMib_SetDevDrvId(Node_Entry_Id, Unterstützte Geräte-Treiber-Identifikation); SysMib_Reconfig(Node_Entry_Id);	CHECK_CFG_DYN
20	WRITE_CFG	=> COM_Write.req(Index = 0xA003, SysMib_GetPrm(Node_Entry_Id)	WRITE_PRM

#	Current State	Event /Condition =>Action	Next State
21	WRITE_PRM	**COM_Write.con(Index, Status)** /Status = OK => COM_Write.req(Index = 0xA004, SysMib_GetPrmId(Node_Entry_Id)	WRITE_PRM_ID
22	WRITE_PRM_ID	**COM_Write.con(Index, Status)** /Status = OK => COM_Write.req(Index = 0xA006, SysMib_GetDevDrvId(Node_Entry_Id)	WRITE_DEV_DRV_ID
23	WRITE_DEV_DRV_ID	**COM_Write.con(Status)** /Status = OK => COM_Read.req(Index = 0x1002)	READ_CFG_STATE
24	SET_ACTIVATE	**COM_Activate.con(Status)** /Status = OK => SysMib_SetNodeState(Node_Entry_Id, Active); NodeHandler_Continue.con(Node_Entry_Id, Status = OK)	IDLE
25	any-state	**COM_Write.con(Index, Status)** /Status <> OK => SysMib_SetNodeState(Node_Entry_Id, Offline); (Bemerkung: Slave-Node muss diese Funktionalität unterstützen) NodeHandler_Continue.con(Node_Entry_Id, Status)	IDLE
26	any-state	**COM_Read.con(Index, Data, Status)** /Status <> OK => SysMib_SetNodeState(Node_Entry_Id, Offline); (Bemerkung: Slave-Node muss diese Funktionalität unterstützen) NodeHandler_Continue.con(Node_Entry_Id, Status)	IDLE
27	any-state	**COM_Activate.con(Status)** /Status <> OK => SysMib_SetNodeState(Node_Entry_Id, Offline); NodeHandler_Continue.con(Node_Entry_Id, Status)	IDLE

11.2.13 Diagnosis

Dienst-Beschreibung

Starten der Diagnose

Mit dem Dienst *Start_Diag* wird die Zustandsmaschine für die Diagnose gestartet.

Parametername	.req	.con
Argument	M	
Result(+)		S
Status		M
Result(-)		S
Status		M

Tabelle 8 Dienst Start_Diag

Argument

Die Argumente übermitteln die dienstspezifischen Parameter einer Dienstanforderung.

Result (+)

Diese Parameter zeigen die positive Anwort auf die Dienstanforderung an.

Status

Der Parameter *Status* zeigt an, das der Dienst erfolgreich ausgeführt wurde.

Status	Bedeutung
OK	Dienst wurde erfolgreich ausgeführt

Result (-):

Diese Parameter zeigen die negative Anwort auf die Dienstanforderung an.

Status:

Dieser Parameter zeigt einen Fehlercode an.

Status	Bedeutung
NO	Dienst konnte nicht ausgeführt werden

Stoppen der Diagnose

Mit dem Dienst *Stop_Diag* wird die Zustandsmaschine für die Diagnose gestoppt.

Parametername	.req	.con
Argument	M	
Result(+)		S
Status		M
Result(-)		S
Status		M

Tabelle 9 Dienst Stop_Diag

Argument

Die Argumente übermitteln die dienstspezifischen Parameter einer Dienstanforderung.

Result (+)

Diese Parameter zeigen die positive Anwort auf die Dienstanforderung an.

Status

Der Parameter *Status* zeigt an, das der Dienst erfolgreich ausgeführt wurde.

Status	Bedeutung
OK	Dienst wurde erfolgreich ausgeführt

Result (-):

Diese Parameter zeigen die negative Anwort auf die Dienstanforderung an.

Status:

Dieser Parameter zeigt einen Fehlercode an.

Status	Bedeutung
NO	Dienst konnte nicht ausgeführt werden

Auslesen eines Nodes

Mit dem Dienst *Read_Node_Entry* können alle Informationen zu einem Node ausgelesen werden. Die Informationen befinden sich im wesentlichen in der System-MIB

Parametername	.req	.con
Argument	M	
Node_Entry_Id	M	
Result(+)		S
Data		M
Status		M
Result(-)		S
Status		M

Tabelle 10 Dienst Read_Node_Entry

Argument

Die Argumente übermitteln die dienstspezifischen Parameter einer Dienstanforderung.

Node_Entry_Id

Dieser Parameter enthält die Identifikation eines Nodes.

Result (+)

Diese Parameter zeigen die positive Anwort auf die Dienstanforderung an.

Data

Der Parameter *Data* enthält die ausgelesenen Informationen.

Status

Der Parameter *Status* zeigt an, das der Dienst erfolgreich ausgeführt wurde.

Status	Bedeutung
OK	Dienst wurde erfolgreich ausgeführt

Result (-):

Diese Parameter zeigen die negative Anwort auf die Dienstanforderung an.

Status:

Dieser Parameter zeigt einen Fehlercode an.

Status	Bedeutung
NO	Dienst konnte nicht ausgeführt werden

Anzeigen einer Änderung in der Node Live List

Mit dem Dienst *Diag_Node_Live_List* wird eine Änderung in der Live List der im Netzwerk befindlichen Nodes angezeigt. Dieser Dienst wird direkt auf den Dienst Node_Live_List.ind abgebildet.

Parametername	.ind
Argument	M
Node_Entry_Id	M
Node_Entry_State	M

Tabelle 11 Dienst Diag_Node_Live_List / Node_Live_List

Argument

Die Argumente übermitteln die dienstspezifischen Parameter einer Dienstanforderung.

Node_Entry_Id

Dieser Parameter enthält die Identifikation eines Slave-Nodes.

Node_Entry_State

Dieser Parameter enthält den Zustand eines Slave-Nodes.

State	Bedeutung
Free	Identifiaktion frei; Slave-Node nicht mehr erreichbar
Setup	Identifiaktion besetzt; Slave-Node ansprechbar

Protokoll-Beschreibung

#	Current State	Event /Condition =>Action	Next State
1	STOP	**Start_Diag.req** => Start_Diag.con	RUN
2	RUN	**Start_Diag.req** => Start_Diag.con	RUN
3	RUN	**Stop_Diag.req** => Stop_Diag.con	STOP
4	STOP	**Stop_Diag.req** => Stop_Diag.con	STOP
5	RUN	**Read_Node_Entry.req(Node_Entry_Id)** => Read_Node_Entry.req(Data, Status = OK)	RUN
6	STOP	**Read_Node_Entry.req(Node_Entry_Id)** => Read_Node_Entry.req(Status = NA)	STOP
7	RUN	**Diag_Node_Live_List.ind(Node_Entry_Id, Entry_State)** => Node_Live_List.ind(Node_Entry_Id, Entry_State)	RUN
8	STOP	**Diag_Node_Live_List.ind(Node_Entry_Id, Entry_State)** => ignore	STOP
9	RUN	**Diag.ind(Node_Entry_Id, Entry_State)** => Diagnose.ind(Node_Entry_Id, Entry_State)	RUN
10	STOP	**Diag.ind(Node_Entry_Id, Entry_State)** => ignore	STOP

11.3 Slave-Node

Überblick

In der Abbildung 3 sind die Komponenten des Slave-Nodes für die PnP-Funktionalität und deren Zusammenspiel über Dienste dargestellt.

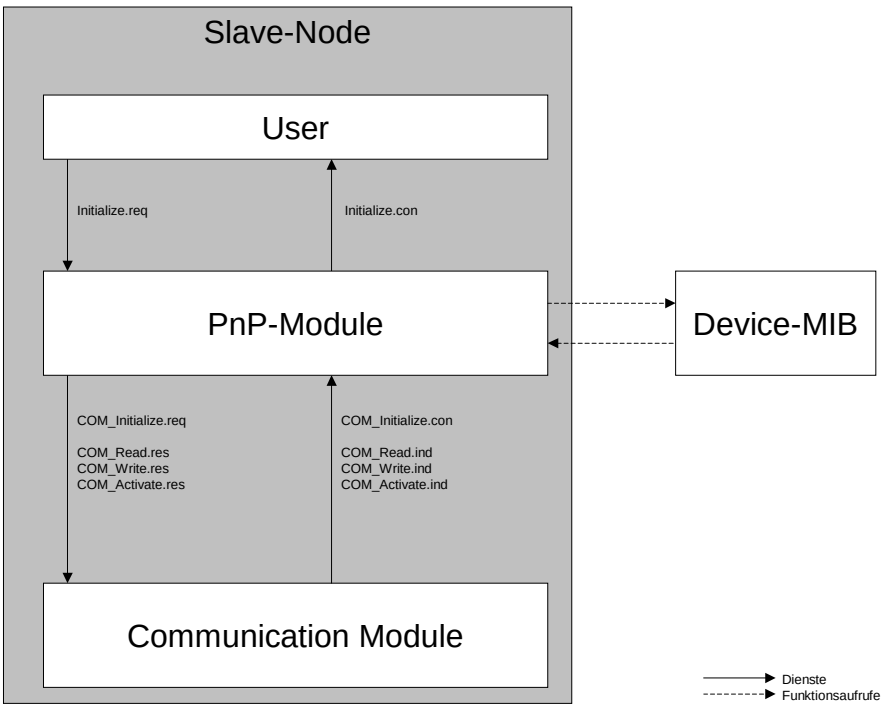


Abbildung 3 Dienste Slave-Node

Der Slave-Node agiert im Gegensatz zum Master-Node rein passiv. Nach der erfolgreichen Initialisierung (*Initialize*) reagiert der Slave-Node vorwiegend auf die Lese- und Schreibzugriffe des Master-Nodes, um die gerätespezifischen Objekt zu lesen und zu aktualisieren. Falls der Slave-Node vom Master-Node falsch parametriert wird, lässt er sich nicht aktivieren (*COM_Activate*).
Ein Master-Node kann ebenfalls über die Slave-Node-Funktionalität verfügen und sich im Netz dementsprechend verhalten.

Objekt-Beschreibung

Object 0x1000: Node-Type

Der Node-Typ wird als Bestandteil des Objektes Device-Type kodiert. Dazu wird das Bit 24 verwendet.

Object Description

Index	**0x1000**
Name	Node Type
Object Code	VAR
Data Type	UNSIGNED1
Category	Mandatory

Entry Description

Access	ro
Value Range	UNSIGNED1
Default Value	0

Coding

Code	Node Type
0	Slave-Node

Object 0x1002: Konfigurations-Zustand (Manufacturer Status Register)

Der Konfigurations-Zustand wird im Manufacturer Status Register abgelegt.

Object Description

Index	**0x1002**
Name	Konfigurations-Zustand
Object Code	VAR
Data Type	UNSIGNED8
Category	Mandatory

Entry Description

Access	ro
Value Range	UNSIGNED8
Default Value	0x00

Coding

Code	Konfigurations-Zustand
0x00	Keine Konfiguration verfügbar
0x01	nicht konfiguriert
0x02	konfiguriert
0x03 - 0xff	reserved

Object 0xA000: Device Data Sheet

In diesem Objekt sind die Leistungsdaten des Gerätes gespeichert.

Object Description

Index	**0xA000**
Name	Device Data Sheet
Object Code	RECORD
Data Type	Device Profile Specific
Category	Optional

Entry Description

Access	ro
Value Range	Device Profile Specific
Default Value	Device Profile Specific

Object 0xA001: Supported Operation Modes

In diesem Objekt werden für den Gerätetyp relevante Betriebsmodi gespeichert, die vom Gerät unterstützt werden.

Object Description

Index	**0xA001**
Name	Supported Operation Modes
Object Code	RECORD
Data Type	Device Profile Specific
Category	Optional

Entry Description

Access	ro
Value Range	Device Profile Specific
Default Value	Device Profile Specific

Object 0xA002: Device Parameter Set Description

In diesem Objekt wird die Beschreibung der Geräteobjekte abgelegt.

Object Description

Index	**0xA002**
Name	Device Parameter Set Description
Object Code	RECORD
Data Type	Device Profile Specific
Category	Mandatory

Entry Description

Access	ro
Value Range	Device Profile Specific
Default Value	Device Profile Specific

Object 0xA003: Device Parameter Set

In diesem Objekt wird die konkrete Wertebelegung der Geräteobjekte abgelegt.

Object Description

Index	**0xA003**
Name	Device Parameter Set
Object Code	RECORD
Data Type	Device Profile Specific
Category	Mandatory

Entry Description

Access	rw
Value Range	Device Profile Specific
Default Value	Device Profile Specific

Object 0xA004: Device Parameter Set Identification

In dieses Objekt wird die Identifikation eines Geräte-Parametersatzes abgelegt.

Object Description

Index	**0xA004**
Name	Device Parameter Set Identification
Object Code	VAR
Data Type	UNSIGNED16
Category	Mandatory

Entry Description

Access	rw
Value Range	UNSIGNED16
Default Value	0x00

Object 0xA005: Supported Device Driver

Mit diesem Objekt kann auf Anwendungssoftware referenziert werden, die im Master-Node abläuft und die Anwendungsfunktionalität des Slave-Nodes nutzt.

Object Description

Index	**0xA005**
Name	Supported Device Driver
Object Code	RECORD
Data Type	Device Profile Specific
Category	Optional

Entry Description

Access	rw
Value Range	Device Profile Specific
Default Value	Device Profile Specific

Object 0xA006: Device Driver Indentification

In diesem Objekt wird hinterlegt, welche(r) Geräte-Treiber für die relevante Anwendung gewählt wurde(n).

Object Description

Index	**0xA006**
Name	Device Driver Indentification
Object Code	ARRAY
Data Type	UNSIGNED16
Category	Optional

Entry Description

Access	rw
Value Range	Device Profile Specific
Default Value	Device Profile Specific

11.3.1Funktionen

In der folgenden Tabelle sind die Funktionen aufgeführt, die von der lokalen MIB des Slave-Node bereitgestellt werden.

Rückgabe	Funktion	Übergabe	Beschreibung
Konfigurations-Zustand	DevMib_GetCfgState		Lesen des Konfigurations-Zustand
Geräte-Datenblatt	DevMib_GetDevSheet		Lesen des Geräte-Datenblattes
Unterstützte Betriebsmodi	DevMib_GetModes		Lesen der unterstützten Betriebsmodi
Geräte-Parametersatz-Beschreibung	DevMib_GetPrmDescr		Lesen der Geräte-Parametersatz-Beschreibung
Geräte-Parametersatz	DevMib_GetPrm		Lesen des Geräte-Parametersatzes
Geräte-Parametersatz-Identifikation	DevMib_GetPrmId		Lesen der Geräte-Parametersatz-Identifikation
Unterstützte Geräte-Treiber	DevMib_GetDevDrv		Lesen der unterstützten Geräte-Treiber
Unterstützte Geräte-Treiber-Identifikation	DevMib_GetDevDrvId		Lesen der unterstützte Geräte-Treiber-Identifikation
	DevMib_SetCfgState	Konfigurations-Zustand	Schreiben des Konfigurations-Zustand
	DevMib_SetPrm	Geräte-Parametersatz	Schreiben des Geräte-Parametersatzes
	DevMib_SetPrmId	Geräte-Parametersatz-Identifikation	Schreiben der Geräte-Parametersatz-Identifikation
	DevMib_SetDevDrvId	Unterstützte Geräte-Treiber-Identifikation	Schreiben der unterstützte Geräte-Treiber-Identifikation
TRUE, FALSE	DevMib_Reconfig		Rekonfiguration der Geräte-Parameter aufgrund von neuen Daten vom Master-Node

11.3.1.1 Dienst-Beschreibung

Initialisierung des Slave-Nodes

Mit dem Dienst *Initialize* wird die PnP-Funktionalität des Slave-Nodes initialisiert. Während der Initialisierung wird eine Verbindung zur Device-MIB (Management Information Base) aufgebaut, in der alle gerätespezifischen Objekte gespeichert sind. Der Zugriff auf die Device-MIB erfolgt transparent über eine Funktionsschnittstelle.

Parametername	.req	.ind	.res	.con
Argument	M			
Init_Data	M			
Result(+)				S
Status				M
Result(-)				S
Status				M

Tabelle 12 Dienst Initialize

Argument

Die Argumente übermitteln die dienstspezifischen Parameter einer Dienstanforderung.

Init_Data

Dieser Parameter enthält die Initialisierungsdaten für die PnP-Funktionalität des Master-Node.

Result (+)

Diese Parameter zeigen die positive Anwort auf die Dienstanforderung an.

Status

Der Parameter *Status* zeigt an, das der Dienst erfolgreich ausgeführt wurde.

Status	Bedeutung
OK	Dienst wurde erfolgreich ausgeführt

Result (-):

Diese Parameter zeigen die negative Anwort auf die Dienstanforderung an.

Status:

Dieser Parameter zeigt einen Fehlercode an.

Status	Bedeutung
NO	Dienst konnte nicht ausgeführt werden

11.3.1.2 Protokoll-Beschreibung

#	Current State	Event /Condition =>Action	Next State
1	OFFLINE	**Initialize.req(Init_Data)** => COM_Initialize.req	COM_INIT
2	COM_INIT	**COM_Initialize.con** /Status = OK => Initialize.con(Status)	RUN

#	Current State	Event /Condition =>Action	Next State
3	COM_INIT	**COM_Initialize.con** /Status <> OK => Initialize.con(Status)	OFFLINE
3	RUN	**COM_Read.ind(Index)** /Index = 0x1000 => COM_Read.res(Index, DeviceMib_GetCfgState(), Status = OK)	same
4	RUN	**COM_Read.ind(Index)** /Index = 0x1002 => COM_Read.res(Index, DeviceMib_GetCfgState(), Status = OK)	same
5	RUN	**COM_Read.ind(Index)** /Index = 0xA000 => COM_Read.res(Index, DeviceMib_GetDevSheet(), Status = OK)	same
6	RUN	**COM_Read.ind(Index)** /Index = 0xA001 => COM_Read.res(Index, DeviceMib_GetModes(), Status = OK)	same
7	RUN	**COM_Read.ind(Index)** /Index = 0xA002 => COM_Read.res(Index, DeviceMib_GetPrmDescr(), Status = OK)	same
8	RUN	**COM_Read.ind(Index)** /Index = 0xA003 => COM_Read.res(Index, DeviceMib_GetPrm(), Status = OK)	same
9	RUN	**COM_Read.ind(Index)** /Index = 0xA004 => COM_Read.res(Index, DeviceMib_GetPrmId(), Status = OK)	same
10	RUN	**COM_Read.ind(Index)** /Index = 0xA005 => COM_Read.res(Index, DeviceMib_GetDevDrv(), Status = OK)	same
11	RUN	**COM_Read.ind(Index)** /Index = 0xA006 => COM_Read.res(Index, DeviceMib_GetDevDrvId(), Status = OK)	same
11	RUN	**COM_Read.ind(Index)** /Index = 0xA100 => COM_Read.res(Index, DeviceMib_GetDevDrvId(), Status = OK)	same
12	RUN	**COM_Read.ind(Index)** /Index <> gültiger Index => COM_Read.res(Index, 0, Status = NO)	same
13	RUN	**COM_Write.ind(Index, Data)**/Index = 0xA003=>DevMib_SetPrm(Geräte-Parametersatz); if (DevMib_Reconfig = TRUE) DevMib_SetCfgState(konfiguriert) else DevMib_SetCfgState(nicht konfiguriert)COM_Write.res(Status = OK)	RUN

#	Current State	Event /Condition =>Action	Next State
14	RUN	**COM_Write.ind(Index, Data)** /Index = 0xA004 => DevMib_SetPrmId(Geräte-Parametersatz-Identifikation); if (DevMib_Reconfig = TRUE) DevMib_SetCfgState(konfiguriert) else DevMib_SetCfgState(nicht konfiguriert) COM_Write.res(Status = OK)	RUN
15	RUN	**COM_Write.ind(Index, Data)** /Index = 0xA006 => DevMib_SetDevDrvId(Unterstützte Geräte-Treiber-Identifikation); if (DevMib_Reconfig = TRUE) DevMib_SetCfgState(konfiguriert) else DevMib_SetCfgState(nicht konfiguriert) COM_Write.res(Status = OK)	RUN
16	RUN	**COM_Write.ind(Index, Data)** /Index <> gültiger Index => COM_Write.res(Status = NO)	RUN
17	RUN	**COM_Activate.ind(Slave-Node)** /DevMib_GetCfgState() = konfiguriert => COM_Activate.res(Status = OK)	RUN
18	RUN	**COM_Activate.ind(Slave-Node)** /DevMib_GetCfgState() = nicht konfiguriert => COM_Activate.res(Status = NO)	RUN
19	RUN	**COM_Activate.ind(Slave-Node)** /Slave-Node <> eigene Slave-Adresse => COM_Activate.res(Statzus = NO)	RUN

11.3.2Validierung der Beschreibung an Plug-and-Play-Szenarien

Anhand von exemplarischen Plug-and-Play-Szenarien soll nachgewiesen werden, dass die formale Beschreibung der PAPAS-PnP-Spezifikation den gestellten Anforderungen entspricht. Es werden zwei Szenarien beschrieben, ein Systemanlauf und ein Gerätetausch. Die Randbedingungen der Szenarien sind so gewählt worden, dass mehrere Anforderungen je Szenario geprüft werden.
Die Beschreibung der Szenarien erfolgt in Tabellenform. In der ersten Spalte (Node) steht in welchem Gerätetyp die beschriebenen Aktionen ablaufen. Die zweite Spalte (SM) bezieht sich auf eine der im Abschnitt dargestellten Zustandsmaschinen. Die nächste Spalte (Zustand) kennzeichnet den Ausgangszustand für die beschriebenen Aktionen. In der vierten Spalte (Aktionen) wird beschrieben welche Aktionen von den Geräten ausgeführt werden. Um den Verlauf in den Zustandsmaschinen verfolgen zu können, werden in der letzten Spalte die Transitionen aufgelistet, die durchlaufen werden.

Systemanlauf

Für den Systemanlauf wird angenommen, dass ein Gerät (Slave-Node 1) die Werte des Parametersatzes nicht-flüchtig speichert, während ein zweites Gerät (Slave-Node 2) dazu nicht in der Lage ist. Es wird davon ausgegangen, dass das Projekt vollständig ist, das heißt die beiden Geräte sind in dem Projekt enthalten. Es werden also beim Slave-Node 1 die Werte des Parametersatzes aus dem Gerät gelesen, während die Werte des Parametersatzes von Slave-Node 2 mit einem Engineering-Tool dem System bekannt gegeben und vor der Aktivierung in das Gerät geladen werden müssen.

Node	SM	Zustand	Aktionen	Referenz
Master-Node	Scheduler	OFFLINE	Applikation initialisiert PnP-Scheduler-Zustandsmaschine; Anmeldung beim Kommunikationsmodul; Aktivierung der Node-Life-List-Funktion im Kommunikationsmodul; Verbindung zur System-MIB und Setzen des Zustandswertes "Setup"; Information der Applikation, dass Initialisierung erfolgreich war und der Systemanlauf gestartet ist	INIT, COM_INIT, COM_INIT_POS
Master-Node	Diagnosis	STOP	Applikation startet Diagnose-Zustandsmaschine	START
Slave-Node 1	Scheduler	OFFLINE	Applikation initialisiert PnP-Scheduler-Zustandsmaschine; Anmeldung beim Kommunikationsmodul	INIT, INIT_POS
Slave-Node 2	Scheduler	OFFLINE	Applikation initialisiert PnP-Scheduler-Zustandsmaschine; Anmeldung beim Kommunikationsmodul	INIT, INIT_POS
Master-Node	Scheduler	SETUP	Kommunikationsmodul meldet, dass Node 1 im Netz erkannt wurde; Node-Type von Node 1 wird abgefragt und als Slave-Node identifiziert, dessen Gerätebeschreibung nicht in der System-MIB vorhanden ist; Anlegen eines Eintrages in der System-MIB und Information des Slave-Node-Handlers	NODE_SETUP, SN_NOT_EXIST
Master-Node	Slave-Node-Handler	IDLE	Abfragen des Konfigurationszustand von Slave-Node 1	RD_CFG_STATE
Slave-Node 1	Scheduler	RUN	Antwort, dass Slave-Node 1 Gerätebeschreibung und Parametersatz gespeichert hat	RD_CFG_STATE
Master-Node	Slave-Node-Handler	READ_CFG_STATE	Gerätebeschreibung und Parametersatz des Slave-Node 1 nicht in System-MIB enthalten; Gerätebeschreibung und Parametersatz im Slave-Node 1 verfügbar	UNKNOWN, UNKNOWN_RD
Master-Node	Slave-Node-Handler	READ_DATA_SHEET	Abfragen der Gerätebeschreibung und des aktuellen Parametersatzes von Slave-Node 1; Eintrag in die System-MIB; Slave-Node 1 ist konfiguriert und im Projekt enthalten; Parametersatz ist mit Projekt identisch; Eintrag dieses Zustandes "Konfiguriert" in die System-MIB; Aktivierung von Slave-Node 1	RD_MODES, RD_PRM_DESCR, RD_PRM, RD_PRM_ID, RD_DEV_DRV, RD_DEV_DRV_ID, CHK_CFG_READ, CONFIGURED
Slave-Node 1	Scheduler	RUN	Aktivierung des Slave-Nodes	ACTIVATE
Master-Node	Slave-Node-Handler	SET_ACTIVATE	Slave-Node 1 ist aktiviert; Eintrag dieses Zustandes "Aktiviert" in die System-MIB; Information des Schedulers	ACTIVATE_POS
Master-Node	Scheduler	SETUP	Eintrag des Zustandes von Slave-Node 1 in Diagnose-Modul (steht für Applikation zur Verfügung)	SN_SETUP
Master-Node	Scheduler	SETUP	Kommunikationsmodul meldet, dass Node 2 im Netz erkannt wurde; Node-Type von Node 2 wird abgefragt und als Slave-Node identifiziert, dessen Gerätebeschreibung nicht in der System-MIB vorhanden ist; Anlegen eines Eintrages in der System-MIB und Information des Slave-Node-Handlers	NODE_SETUP, SN_NOT_EXIST
Master-Node	Slave-Node-Handler	IDLE	Abfragen des Konfigurationszustand von Slave-Node 2	RD_CFG_STATE
Slave-Node 2	Scheduler	RUN	Antwort, dass Slave-Node 2 keine Gerätebeschreibung und keinen Parametersatz gespeichert hat	RD_CFG_STATE

Master-Node	Slave-Node-Handler	READ_CFG_STATE	Gerätebeschreibung und Parametersatz des Slave-Node 2 in System-MIB enthalten (über Engineering-Tool); Im Slave-Node 2 ist keine Gerätebeschreibung und kein Parametersatz verfügbar; Slave-Node 2 ist im Projekt vorhanden und Gerätebeschreibung ist kompatibel	KNOWN, NOT_CONFIGURED
Master-Node	Slave-Node-Handler	WRITE_CFG	Schreiben des Parametersatzes in Slave-Node 2	WR_PRM, WR_PRM_ID, WR_DEV_DRV_ID
Slave-Node 2	Scheduler	RUN	Speicher des Parametersatz; Ändern des Konfigurationszustand in "Konfiguriert", wenn alle Werte übertragen wurden	WR_PRM, WR_PRM_ID, WR_DEV_DRV_ID
Master-Node	Slave-Node-Handler	WRITE_DEV_DRV_ID	Abfragen des Konfigurationszustand von Slave-Node 2	CHK_CFG_WRITE
Slave-Node 2	Scheduler	RUN	Antwort, dass Slave-Node 2 Gerätebeschreibung und Parametersatz gespeichert hat	RD_CFG_STATE
Master-Node	Slave-Node-Handler	READ_CFG_STATE	Gerätebeschreibung und Parametersatz des Slave-Node 2 in System-MIB enthalten (über Engineering-Tool); Slave-Node 2 ist konfiguriert und im Projekt enthalten; Parametersatz ist identisch; Eintrag dieses Zustandes "Konfiguriert" in die System-MIB; Aktivierung von Slave-Node 2	KNOWN, CONFIGURED
Slave-Node 2	Scheduler	RUN	Aktivierung von Slave-Node 2	ACTIVATE
Master-Node	Slave-Node-Handler	SET_ACTIVATE	Slave-Node 2 ist aktiviert; Eintrag dieses Zustandes "Aktiviert" in die System-MIB; Information des Schedulers	ACTIVATE_POS
Master-Node	Scheduler	SETUP	Eintrag des Zustandes von Slave-Node 2 in Diagnose-Modul (steht für Applikation zur Verfügung)	SN_SETUP
Master-Node	Scheduler	any_state (SETUP)	Applikation startet Produktivdatenübertragung; Konfiguration ist abgeschlossen (System-Parametrierung und System-Projekt identisch); Ermittlung von Systemdaten aus der aktuellen Konfiguration für den Master-Node (z. B. belegte Bandbreite); Benachrichtigung der Applikation, dass alle Geräte aktiv; Starten der Device-Exchange-Zustandsmaschine	OP_ALL
Master-Node	Device Exchange	STOP	Starten der Device Exchange-Zustandsmaschine	OP_ALL

Gerätetausch

Beim Gerätetausch wird angenommen, dass ein Gerät ohne nicht-flüchtigen Speicher gegen Gerät gleicher Klasse mit nicht-flüchtigem Speicher ausgetauscht wird. Allerdings sollen die Werte des Parametersatzes nicht mit denen des entfernten Gerätes identisch sein. Es müssen also nach dem Gerätetausch die projektierten Werte des Parametersatzes in das neue Gerät geschrieben werden, bevor es aktiviert wird.

Node	SM	Zustand	Aktionen	Referenz
Master-Node	Scheduler	OPERATION	Slave-Node wird entfernt; Kommunikationsmodul meldet, dass Slave-Node entfernt wurde; Information der Device-Exchange-Zustandsmaschine über die Entfernung des Slave-Nodes	OP_SN_FREE
Master-Node	Device Exchange	OPERATION_ALL	Slave-Node ist im Projekt enthalten; Entfernung des Eintrages aus der System-MIB; Eintrag des System-Zustandes "Operation Partly" in Diagnose-Modul (steht für Applikation zur Verfügung); Benachrichtigung der Applikation, dass System nicht vollständig ist	OP_PART2
Slave-Node	Scheduler	OFFLINE	Neuer Slave-Node wird hinzugefügt; Applikation initialisiert PnP-Scheduler-Zustandsmaschine; Anmeldung beim Kommunikationsmodul	INIT, INIT_POS

Node	SM	Zustand	Aktionen	Referenz
Master-Node	Scheduler	OPERATION	Kommunikationsmodul meldet, dass Node im Netz erkannt wurde; Node-Type von Node wird abgefragt und als Slave-Node identifiziert, dessen Gerätebeschreibung nicht in der System-MIB vorhanden ist; Information der Device-Exchange-Zustandsmaschine	OP_SN_SETUP, OP_SN_EXIST
Master-Node	Device Exchange	OPERATION_PARTLY	Anlegen eines Eintrages in der System-MIB und Information des Slave-Node-Handlers	OP_PART4
Master-Node	Slave-Node Handler	IDLE	Lesen des Konfigurationszustand des Slave-Nodes	RD_CFG_STATE
Slave-Node	Scheduler	RUN	Antwort, dass Slave-Node Gerätebeschreibung und Parametersatz gespeichert hat	RD_CFG_STATE
Master-Node	Slave-Node Handler	READ_CFG_STATE	Gerätebeschreibung und Parametersatz des Slave-Node nicht in System-MIB enthalten; Gerätebeschreibung und Parametersatz im Slave-Node verfügbar	UNKNOWN, UNKNOWN_RD
Master-Node	Slave-Node Handler	READ_CFG_STATE	Abfragen der Gerätebeschreibung und des aktuellen Parametersatzes von Slave-Node; Eintrag in die System-MIB; Slave-Node ist konfiguriert und im Projekt enthalten; Parametersatz ist mit Projekt nicht identisch; Rekonfiguriert die Werte der Geräteparameter für den Slave entsprechend der Anwendungsparameter	RD_MODES; RD_PRM_DESCR, RD_PRM, RD_PRM_ID, RD_DEV_DRV, RD_DEV_DRV_ID, CHK_CFG_READ, NOT_IDENTICAL
Master-Node	Slave-Node-Handler	WRITE_CFG	Schreiben des im Projekt festgelegten Parametersatzes in den Slave-Node	WR_PRM, WR_PRM_ID, WR_DEV_DRV_ID
Slave-Node	Scheduler	RUN	Speicher des Parametersatz; Ändern des Konfigurationszustand in "Konfiguriert", wenn alle Werte übertragen wurden	WR_PRM, WR_PRM_ID, WR_DEV_DRV_ID
Master-Node	Slave-Node-Handler	WRITE_DEV_DRV_ID	Abfragen des Konfigurationszustand des Slave-Node	CHK_CFG_WRITE
Slave-Node	Scheduler	RUN	Antwort, dass Slave-Node Gerätebeschreibung und Parametersatz gespeichert hat	RD_CFG_STATE
Master-Node	Slave-Node-Handler	READ_CFG_STATE	Gerätebeschreibung und Parametersatz des Slave-Node in System-MIB enthalten; Slave-Node ist konfiguriert und im Projekt enthalten; Parametersatz ist identisch; Eintrag dieses Zustandes "Konfiguriert" in die System-MIB; Aktivierung von Slave-Node	KNOWN, CONFIGURED
Slave-Node	Scheduler	RUN	Aktivierung des Slave-Nodes	ACTIVATE
Master-Node	Slave-Node-Handler	SET_ACTIVATE	Slave-Node ist aktiviert; Eintrag dieses Zustandes "Aktiviert" in die System-MIB; Information der Device-Exchange-Zustandsmaschine	ACTIVATE_POS
Master-Node	Device Exchange	OPERATION_PARTLY	Konfiguration ist abgeschlossen (System-Parametrierung und System-Projekt identisch); Eintrag des Zustandes von Slave-Node in Diagnose-Modul (steht für Applikation zur Verfügung); Benachrichtigung der Applikation, dass alle Geräte aktiv;	OP_ALL5