Implementierung eines Treibers zur Anbindung von Mikrocontroller-basierten Maschinen über OPC-UA an das Konfigurations- und Inbetriebnahme-Tool von SSI Schäfer

Julian Malovanij, B.Sc.

Product Development

SSI Schäfer IT Solutions GmbH
Friedenstraße 30
93053 Regensburg
E-Mail: julian.malovanij@ssi-schaefer.com

Professor Dr. Frank Herrmann

Innovationszentrum für Produktionslogistik und Fabrikplanung

Ostbayerische Technische Hochschule Regensburg
Galgenbergstraße 32
93053 Regensburg
E-Mail: frank.herrmann@oth-regensburg.de

ABSTRACT

Das Thema „Implementierung eines Treibers zur Anbindung von Mikrocontroller-basierten Maschinen über OPC-UA an das Konfigurations- und Inbetriebnahme-Tool von SSI Schäfer“ soll zu einem Gerätetreiber für das Konfigurations- und Inbetriebnahme-Tool CPM führen, welcher den Anschluss von neuen Maschinen an ebendieses ermöglicht.

Aktuell befinden sich diese noch in Entwicklung, jedoch sollen sie, von den Vorteilen des CPM-Tools profitieren, welches unter anderem den Inbetriebnahmeprozess einfacher und schneller gestalten kann.

Ziel ist somit die Anbindung dieser Maschinen durch die Spezifikation einer Schnittstelle und der Implementierung dieser auf Basis der Schnittstellen des CPM-Tools, in welches sich der Gerätetreiber nahtlos einfügen soll.

SCHLÜSSELWÖRTER

Gerätetreiber, Mikrokontroller-basierte Maschinen, SPS, OPC-UA

EINLEITUNG

Dieses Projekt befasst sich mit der Implementierung eines Gerätetreibers zur Anbindung von Maschinen mit einer neuen, eigenentwickelten SPS, dem SSI Controller eXtensible (SCX), an das Konfigurations- und Inbetriebnahme-Tool des Unternehmens SSI Schäfer IT Solutions GmbH. Dieses ist das CPM-Tool, wobei CPM für Configuration, Parametrization und Maintenance steht.

Weil das CPM bisher nur für Siemens S7 Steuerungen ausgelegt ist und die OPC-UA-Verbindungsaufrufe in der Geschäftslogik implementiert sind, ist ein einfacher Anschluss der SCX nicht möglich. Um diese Abhängigkeit von der S7-Schnittstelle und OPC-UA als Kommunikationsprotokoll zu entfernen, wurde das CPM-Tool um eine Treiberschicht erweitert, welche die Geräte für die Geschäftslogik abstrahiert. Dementsprechend ist ein Gerätetreiber über diese Schnittstelle für die neuen SCX-Steuerungen implementiert worden.

IST-SITUATION

CPM-Tool

Wie in Abbildung 1 zu sehen, basiert das CPM-Tool auf einer Client-Server Architektur, wobei der Client in Angular als Webanwendung implementiert ist und via REST sowie Websockets mit dem CPM-Server kommuniziert.

Im Rahmen der Erweiterung um eine Treiberschicht sollen alle Geräte darüber abgewickelt werden, um von spezifischen Kommunikationsschnittstellen unabhängig zu sein. Dementsprechend müssen für SCX und Mock-Implementierung jeweils ein Gerätetreiber erstellt werden.

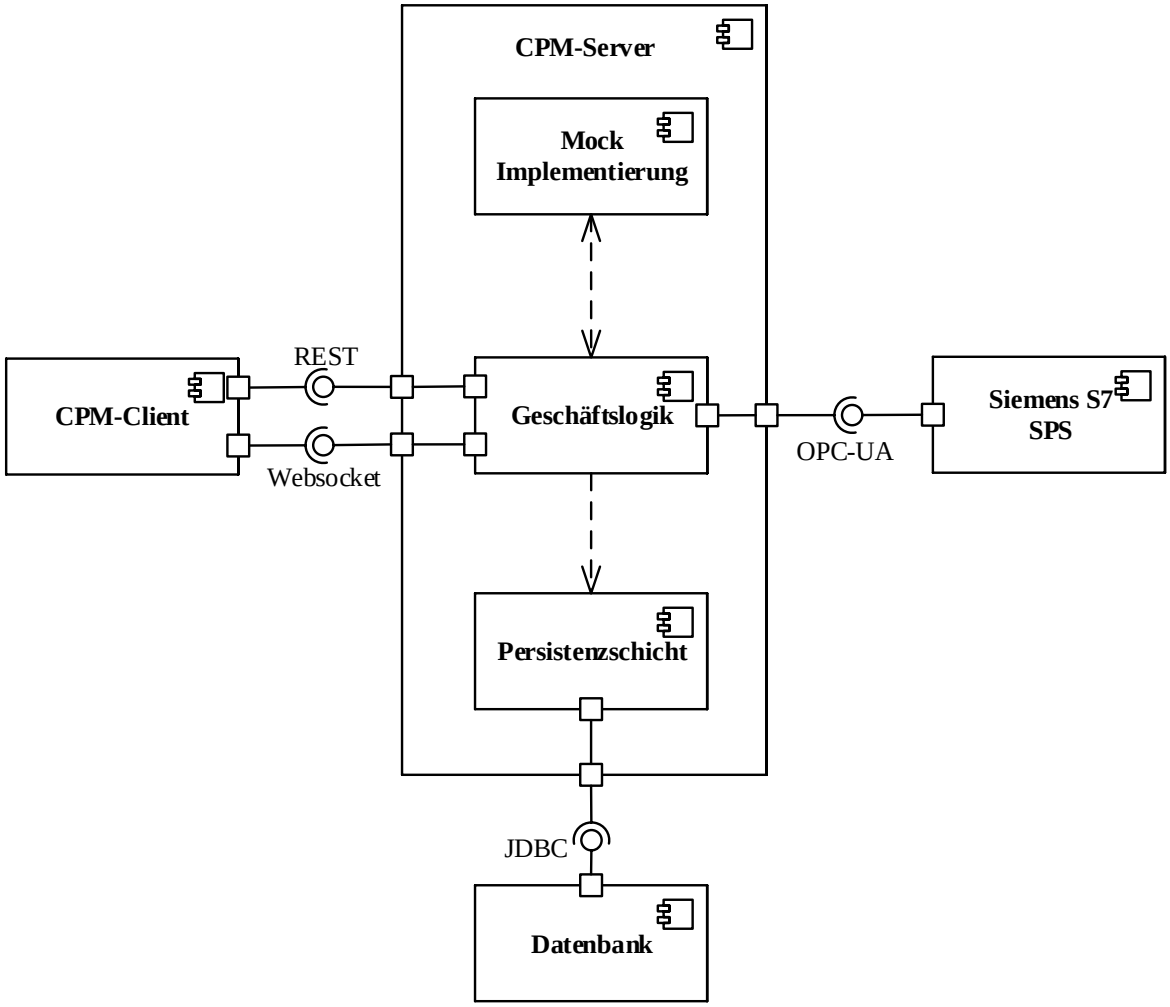

Abbildung 1: Komponentendiagramm CPM Gesamtübersicht (Ist-Situation vor Treiberschicht)

Element-Segment-Block

Das CPM-Tool legt für jede Steuerung bei der Durchsuchung des Baumes Elemente an. Diese sind die Funktionspunkte einer SPS. Da jeder Funktionspunkt aus mehreren Funktionsbausteinen bestehen kann, wie bspw. einem Standardbaustein und einem Baustein für projektspezifische Erweiterungen, sind Segmente eingeführt worden. Diese bilden die Bausteine ab und sind nur eine organisatorische Aufteilung, welche die Daten des Elementes enthält. Alle weiteren SPS-Komponenten werden als Blöcke abgebildet. Dies sind bspw. Fördertechnikelemente wie Förderer oder Plätze.

SCHNITTSTELLENSPEZIFIKATION

Da zum Zeitpunkt der Bearbeitung noch keine konkrete Schnittstellenspezifikation zur Kommunikation zwischen SCX und CPM vorhanden war, ist diese im Rahmen dieses Projekts entstanden. Als Basis dient hierfür die Siemens S7 Schnittstelle. Sie ist durch zahlreiche Abstimmungen mit den SCX- und CPM-Entwicklern entstanden und stellt, neben der Treiberschichtschnittstelle, die Basis für die Implementierung dar.

Device Discovery

Die Device Discovery hat die Aufgabe, einem zu verbindenden Gerät automatisch und ohne weitere Eingaben durch den Benutzer den passenden Gerätetreiber zuzuordnen. Hierfür gibt es zwei relevante Schnittstellen.

Die erste Schnittstelle ist zwischen der Geschäftslogik des CPMs und dem Gerätetreiber. Hierbei müssen Methoden zum Abruf der Treibereigenschaften, sowie eine Methode zur Durchführung der Discovery, angeboten werden. Die Geschäftslogik wiederum bietet eine Methode zur Registrierung des Gerätetreibers an.

Die zweite Schnittstelle ist zwischen dem Gerätetreiber und der SCX-Steuerung. Hierbei stellt das SCX seine Eigenschaften, wie bspw. die Interfaceversion, über einen OPC-UA-Knoten auf dem OPC-UA-Server der Steuerung bereit.

Session

Das CPM ist das führende System für die Datenspeicherung für alle verbundenen SPS-Steuerungen. Aus diesem Grund darf jeweils nur ein CPM-Tool zeitgleich mit einer Steuerung verbunden

sein. Dementsprechend muss das SCX die Attribute „Hostname“ und „Last Hostname“ anbieten, damit die Geschäftslogik des CPMs sich nur verbindet, wenn diese entweder leer oder mit dem gleichen CPM-Hostnamen belegt sind.

Parametrierung

Die Parametrierung ist der Kern des CPMs. Hier können die Parameter der SPS im laufenden Betrieb verändert werden. Damit das CPM-Tool erkennen kann, welche Parameter relevant sind, wird auch hierfür eine Konvention benötigt.

Control Sync
Der Control Sync ist die Durchsuchung des Informationsmodells des OPC-UA-Servers des SCX nach relevanten Parametern. Dabei wird in Elemente, welche die Funktionspunkte einer SPS abbilden, Segmente, welche die Funktionsbausteine abbilden, und Blöcke, welche alle untergeordneten Knoten darstellen, gegliedert. Bei der Durchsuchung werden Elemente gesucht. Im Anschluss werden die gefundenen Elemente vollständig aus der Steuerung ausgelesen. Die Suche bricht ab, sobald ein Knoten keinen Info-Knoten mehr hat. Die Sub-Knoten von diesem werden dann nicht mehr beachtet und der nächste Kind-Knoten des jeweiligen Eltern-Knotens wird durchsucht. Im Info-Knoten sind die relevanten Attribute enthalten. Diese spezifizieren über das Vorhandensein einer NodeID, dass ein Knoten ein Segment eines Elementes ist und enthalten auch den mechanischen Namen, welcher im CPM verwendet wird.

Die Parameter sind als Knoten mit eigenem Wert sowie den Attributen „DetailGroup“, Permission und ID angelegt. Diese Parameter befinden sich jeweils in einem ParameterSet-Knoten, welcher selbst ein Kind-Knoten eines Blocks ist. Im Rahmen des Control Syncs werden aufgefundene Parameter in die Struktur übernommen, jedoch werden die Parameterwerte nicht gespeichert. Die Werte für das CPM kommen bspw. aus einem Import von Standardwerten beim Projekt-Start.

Control Import
Der Control Import entspricht dem Control Sync mit dem Zusatz, dass bei diesem die aktuellen Parameterwerte der SPS in die CPM-Datenbank übernommen werden.

Anforderungen zwischen CPM und SCX
Damit das CPM über Änderungen der Element- und Parameterstrukturen informiert wird, stellt das SCX die Timestamps „ElementModDate“ und „ParameterModDate“ zur Verfügung. Beide lösen einen Control Sync aus.

Zusätzlich hat jedes Element einen Node-Knoten, welcher über die NodeID erreichbar ist. Dieser enthält drei Request-Done-Laufnummernsysteme. Eines zur CPM-seitigen Anforderung einer Parameterübernahme, eines für die SCX-seitige Auslösung eines Control Syncs für ein einzelnes Element, sowie eines für die SCX-seitige Anforderung aller Parameter, welche das CPM dann auf die SCX schreibt.

IO-View und IO-Check

Der IO-View zeigt den Status der Ein- und Ausgänge der SPS in Form einer „Lampe“ an. Hierfür werden vom SCX in den Ordnern „Input“ und „Output“ jeweils Boolean-Variablen bereitgestellt. Für den IO-Check bietet die Steuerung eine Request-Boolean-Variable sowie einen IO-Check-State an. Da die SCX im Falle des IO-Checks in einen Simulationsmodus geht, müssen die Ausgänge auf einen „SimulatedState“ Knoten anstelle des Wurzelknotens gesetzt werden.

Protokollierung

Bei der Protokollierung werden die Log-Einträge von der Steuerung ausgelesen und auf dem Dateisystem des CPMs gespeichert. Um dies zu ermöglichen, stellt das SCX für jedes Protokoll einen Knoten nach Benennungskonvention „Logging*“ bereit, welcher die Eigenschaften des Protokolls enthält.

Device Capabilities

Das CPM-Tool unterstützt verschiedene Funktionalitäten, welche dynamisch im Betrieb ein- und ausgeschaltet werden können. Hierfür bietet das SCX eine Capabilities-Struktur an, welche je eine Boolean-Variable für jede unterstützte CPM-Funktion enthält. Eine Übersicht darüber ist in Tabelle 1 zu sehen.

Capability:	Vom SCX unterstützt:
Parametrierung	Ja
Handbetrieb	Nein
Operationen	Nein
IO-Check	Ja
Protokollierung	Ja
Metriken	Ja
Datenmanipulation	Nein

Tabelle 1: Im SCX unterstützte Capabilities

IMPLEMENTIERUNG

Die Implementierung eines SCX-Gerätetreibers ist das eigentliche Ziel des Projekts.

Mock-Gerätetreiber

Der CPM-Server benutzt, im Rahmen von automatischen Tests, Mock-Implementierungen für OPC-UA-Steuerungen. Da diese vor der Durchführung des Projekts direkt in die Geschäftslogik eingebunden waren, wurden sie in einen eigenen Gerätetreiber migriert. Hierbei wurde die Eigenschaft, dass alle Mock-Implementierungen das OPC-UA-Verbindungs-Interface implementieren, ausgenutzt. Dabei wurde ein S7-Gerätetreiber derartig modifiziert, sodass anstelle von OPC-UA-Verbindungen jeweils eine passende Instanz einer Mock-Implementierung verwendet wird. Diese

Instanzen werden dabei einmalig angelegt und im Arbeitsspeicher gehalten, um deren eigene, transiente Datenhaltung zu ermöglichen.

Um nur einen Gerätetreiber für die fünf Mock-Implementierungen umsetzen zu müssen, erfolgt die Instanzverwaltung mit Hilfe eines Enums. Dieses enthält die URL, den Steuerungsnamen und eine Java Reflection Referenz auf die jeweilige Implementierungsklasse. Zudem enthält das Enum eine statische Methode, welche anhand der URL die korrekte Implementierung zurückgibt. Dieser Enum-Wert ist dann der Key für die Map, welche die Instanzen der Mock-Implementierungen enthält. Über eine weitere, statische Methode kann diese Logik gegenüber dem Gerätetreiber versteckt werden, sodass nur noch ein Aufruf mit der URL als Übergabeparameter notwendig ist um die Implementierung, welche das OPC-UA-Verbindungsinterface implementiert, zu erhalten. Das Verhalten entspricht daher einem OPC-UA-Verbindungsaufruf.

SCX-Gerätetreiber

Das SCX besitzt an einigen Stellen numerische Knoten-IDs anstelle der geforderten Knoten-IDs, welche einen Pfad durch den SCX-Baum beschreiben. Aus diesem Grund sind einige Optimierungen des S7-Treibers nicht möglich und es musste eine Helfer-Klasse zum Handling der potentiell numerischen Knoten-IDs angelegt werden, da das CPM selbst alle Knoten-IDs als Strings speichert.

Der Gerätetreiber des SCX hat, wie in Abbildung 2 ersichtlich, für jede implementierte CPM-Funktion einen eigenen, unabhängigen Service. Dies ermöglicht einen modularen Aufbau, da über die Device Capabilities das Vorhandensein oder Fehlen eines Service für eine CPM-Funktion indiziert werden kann.

Device Discovery

Die Device Discovery ist der Einstiegspunkt jeder Gerätetreiber-Implementierung, da sie dafür verantwortlich ist, ihre Zuständigkeit für die Steuerung zu erkennen und in diesem Fall eine Geräteinstanz an die Geschäftslogik des CPMs zurückzuliefern.

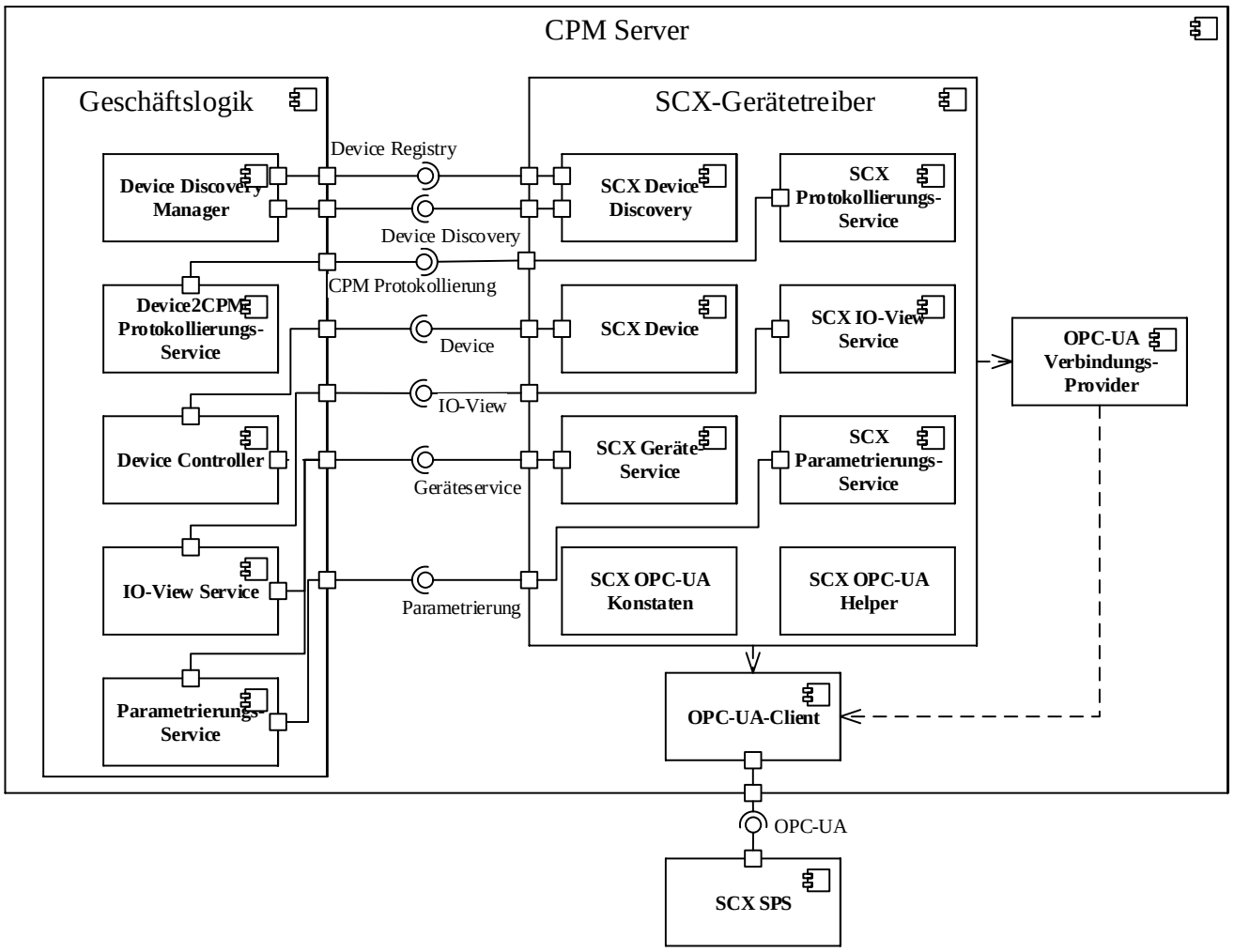

Abbildung 2: Komponentendiagramm des SCX-Gerätetreibers im Systemumfeld

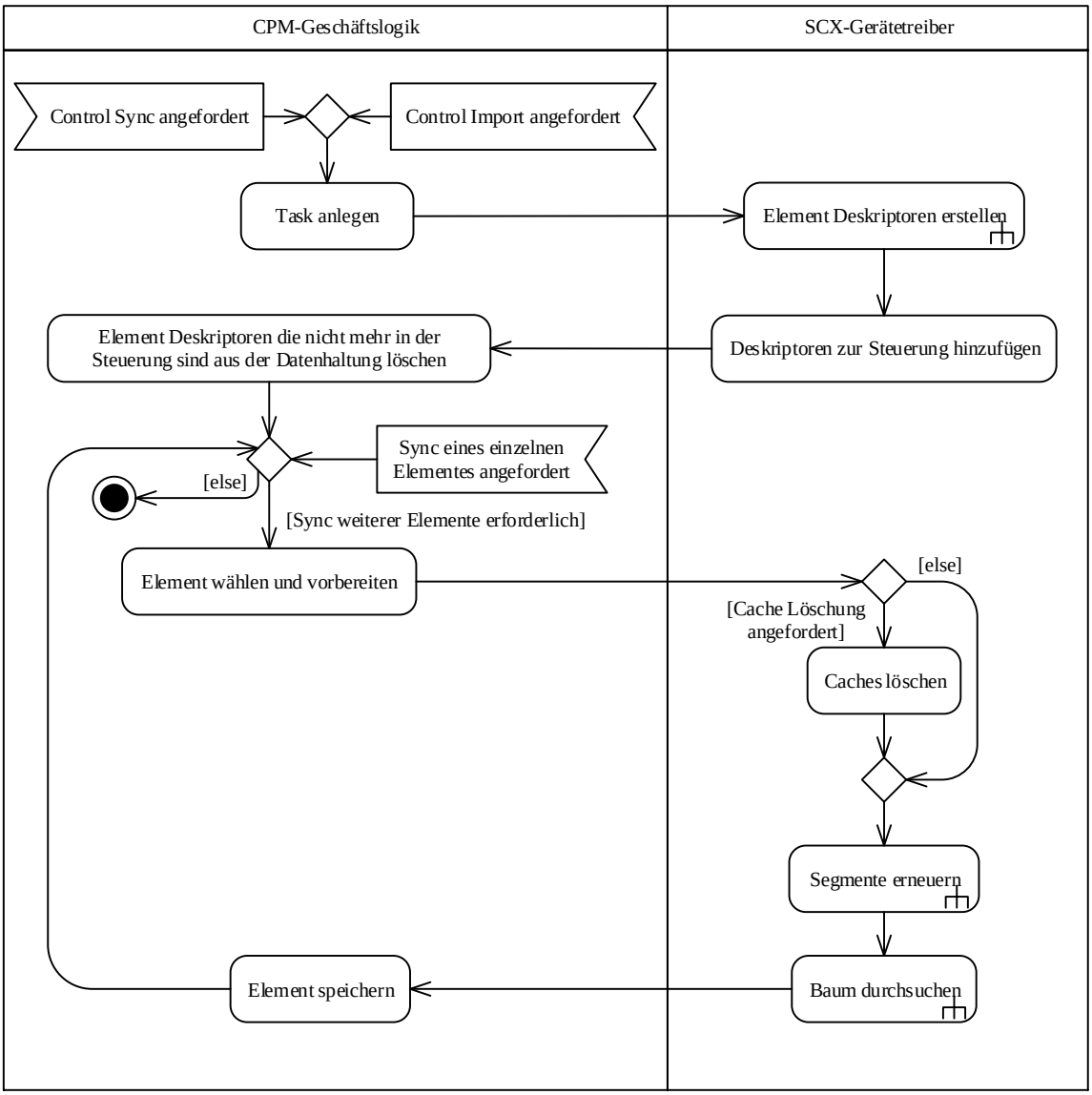

Abbildung 3: Aktivitätsdiagramm als Übersicht für den Ablauf eines Control Syncs/Imports

Um vom CPM-Tool als Gerätetreiber erkannt zu werden, muss sich die Device Discovery, neben der Implementierung des entsprechenden Interfaces, beim Device Discovery Manager der Geschäftslogik des CPMs registrieren.

Mock-Implementierung
Die Erkennung der Zuständigkeit erfolgt auf Basis der übergebenen URL.

SCX-Implementierung
Im SCX-Gerätetreiber ist die Erkennung der Zuständigkeit komplizierter als im Mock-Gerätetreiber. Hier ist ausschlaggebend, dass die DeviceHardwarePlattform, der DeviceType und die InterfaceVersion vom implementierten Treiber mit denen der Steuerung übereinstimmen.

Die Device Discovery verbindet sich hierfür mittels einer OPC-UA-Verbindung direkt mit dem SCX und versucht die benötigten Werte auszulesen. Schlägt dies fehl, wird null zurückgegeben, oder stimmen die gelesenen Werte nicht mit denen des Treibers überein, so ist dieser Gerätetreiber nicht zuständig. Andernfalls wird eine neue Geräteinstanz an die Geschäftslogik zurückgegeben.

Im Rahmen dieser Rückgabe werden weitere benötigte Informationen aus dem SCX ausgelesen. Diese sind die Device Capabilities, der Gerätename, -typ und -version.

Session

Gerät (DeviceImpl)
Die Geräteklasse ist für den Verbindungsauf- und -abbau zuständig. Dies wird größtenteils an einen OPC-UA-Connection-Provider weitergegeben, jedoch müssen alle OPC-UA-spezifischen Implementierungen wie Verbindungs-Listener in CPM-Äquivalente gekapselt werden.

Diese Klasse enthält außerdem Referenzen auf alle Services des Treibers und stellt diese dem CPM-Tool zur Verfügung.

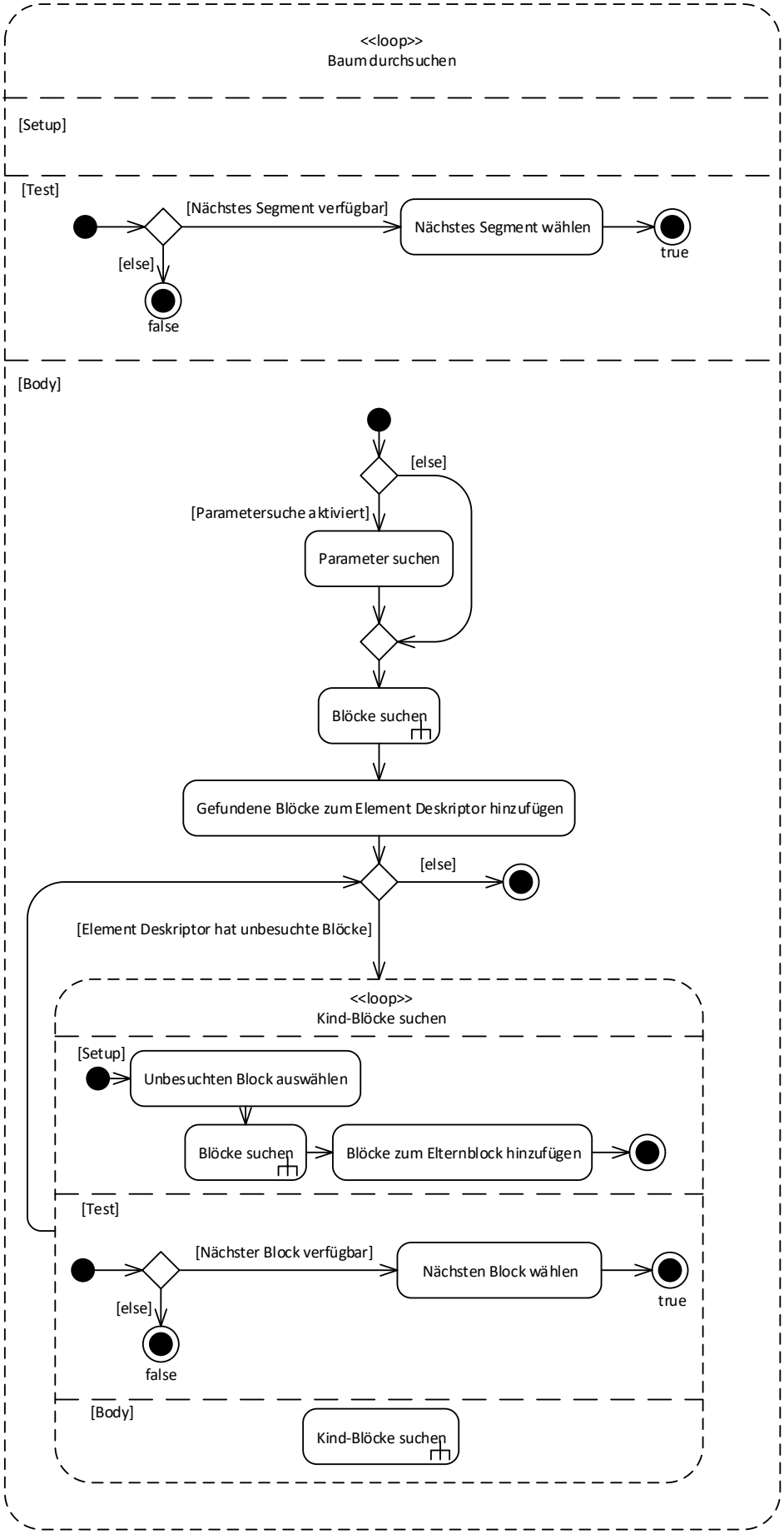

Abbildung 4: Subaktivität Baum durchsuchen

Geräteservice (DeviceServiceImpl)
Der Geräteservice bietet das Anlegen von Monitored Items für übergebene Notification-Listener an. Dies erfolgt im Rahmen der Trennung der Geschäftslogik von Kommunikationsprotokollen. Außerdem stellt der Geräteservice den Status der Steuerung sowie die Lese- und Schreibfunktionen bereit.

Parametrierung

Die Parametrierung ist die wichtigste Funktion des CPM-Tools.

Control Sync
Wie in Abbildung 3 zu sehen, erfolgt der Control Sync in zwei Schritten. . Im ersten werden alle Knoten mit einer NodeID in ihrem Info-Knoten gesucht. Dabei wird die rekursive Suche nach dem Fund des ersten entsprechenden Knoten sowie bei Fehlen eines Info-Knotens abgebrochen, ansonsten werden alle weiteren Subknoten durchsucht. Diese Knoten werden als Segment bezeichnet und besitzen neben dem mechanischen Namen und der NodeID optional einen String Control-Block. Diese Segmente werden dann auf Basis ihres mechanischen Namens und der NodeID zu Elementen zusammengefasst. Eine Liste dieser Elemente wird dann an die Geschäftslogik des CPM zurückgegeben.

Wie ebenfalls aus Abbildung 3 hervorgeht, erfolgt der zweite Schritt, nach dem Aktualisieren der CPM-internen gespeicherten Elemente, für jedes Element einzeln. Dementsprechend werden die Elemente von der Geschäftslogik an den Gerätetreiber übergeben. Dies ist nötig, da ansonsten für die Anforderung eines Control Syncs eines einzelnen Elementes eine weitere Implementierung notwendig wäre. In diesem zweiten Schritt werden die Segmente des Elements nochmals von der Steuerung gelesen. Danach wird der Baum rekursiv nach Blöcken durchsucht. Diese Suche ist in Abbildung 4 zu sehen. Die Rekursion bricht ab, sobald ein Knoten keinen Info-Knoten enthält. In einem solchen Fall wird der nächste Kind-Knoten des Eltern-Knotens betrachtet. Dabei wird ein Cache verwendet, um mehrfaches Lesen eines Knotens zu verhindern. Diese in Abbildung 5 dargestellte Blocksuche wird für alle Kind-Knoten des aktuellen Blocks rekursiv durchgeführt, sodass alle Blöcke im Baum mit einem Pfad, in welchem jeder Knoten einen Info-Knoten hat, gefunden werden können. Im Rahmen der Suche nach Blöcken wird für jeden Block auch nach Parametern gesucht. Diese sind im ParameterSet-Knoten enthalten und können von dort ausgelesen und in eine entsprechende CPM-Repräsentation umgeformt werden. Da auch hier numerische Knoten-IDs zum Einsatz kommen, müssen die ParameterSet-Knoten durch einen OPC-UA-Browse-Aufruf durchsucht werden. Um hier ebenfalls eine Beschleunigung zu erzielen, wird ein weiterer Cache verwendet.
Für jeden Parameter in einem ParameterSet-Knoten wird daraufhin zunächst der Datentyp ausgelesen. Ist dieser nicht vorhanden oder unbekannt, so wird angenommen, dass dies kein Parameter ist. Andernfalls wird der Parameterwert gelesen und im Anschluss ein Browse des Parameterknotens durchgeführt, um die benötigten Variablen DetailGroup, Permission und ID sowie ggf. das Init-Flag zu finden und daraufhin auszulesen. Der DetailGroup-String beinhaltet dabei den Detail- sowie den Group-String jeweils getrennt durch ein „@"-Zeichen. Die entsprechende Auflösung findet im Rahmen der Parametererstellung direkt im Anschluss an das Auslesen der Werte statt.

Falls ein Block keinen Namen im Info-Knoten hat, so ist dieser für das CPM „unsichtbar", sodass all seine Parameter und Kind-Blöcke an den Eltern-Block angehangen werden.

Control Import
Der Control Import ist grundsätzlich dasselbe wie der Control Sync, jedoch werden in diesem Modus auch die ausgelesenen Parameterwerte von der Geschäftslogik des CPM-Tools in die interne Datenbank übernommen. Da diese auch im Zuge des Control Syncs ausgelesen und übergeben werden, sind somit für diesen Anwendungsfall keine weiteren Implementierungs-arbeiten von Seiten des SCX-Gerätetreibers notwendig.

Anforderungen zwischen SCX und CPM
Die Request-Laufnummer wird für eine Anforderung jeweils um eins erhöht, ebenso wird die Done-Laufnummer für eine Bestätigung jeweils um eins inkrementiert. Zu beachten sind Überläufe des uInt16, da im Gerätetreiber ein Int32 verwendet wird und das Verhalten somit anders ist.

Die Init-Anforderung des CPMs erfolgt nach dem Schreiben eines Parameters mit gesetztem Init-Flag. Nach der Übernahme der Parameterwerte bestätigt die SCX die Durchführung, dies wird vom CPM jedoch nicht ausgewertet.

Die Refresh- und Update-Anforderung erfolgen SCX-seitig und werden vom CPM über Monitored Items erkannt. Nach der Ausführung des Control Syncs bzw. des Schreibens aller Parameter für dieses Element bestätigt das CPM die Ausführung.

Die Timestamps „ElementModDate" und „ParameterModDate" enthalten den letzten Änderungszeitpunkt der jeweiligen Struktur. Sie werden auch bei einem Neustart des SCX gesetzt. Falls dieser Timestamp null ist oder nach dem im CPM gespeicherten Zeitpunkt liegt, wird ein Control Sync durchgeführt. Auch hier erfolgt die Erkennung über Monitored Items.

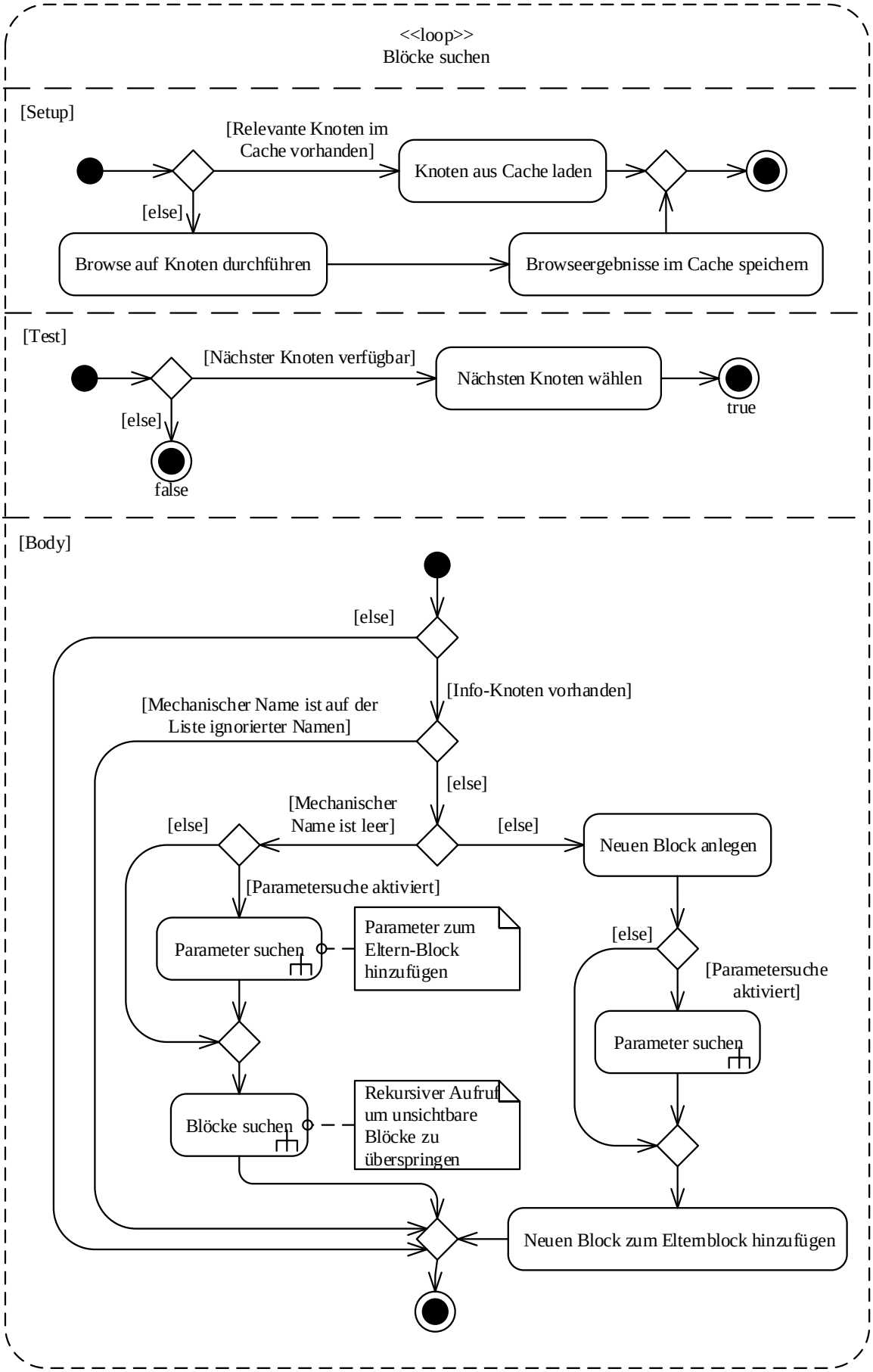

Abbildung 5: Subaktivität Blöcke suchen

IO-View

Der IO-View stellt die Ein- und Ausgänge des SCX als „Lampe“ im Frontend dar. Dafür müssen diese entweder über einen IO-Import oder einen PTC-Import eingelesen werden. Eine Statusänderung wird über Monitored Items erkannt.

IO-Import
Für den IO-Import werden die beiden Ordner „Input“ und „Output“ gebrowst. Hierbei werden alle Einträge vom Datentyp Boolean übernommen. Für Ausgänge gilt hierbei die Sonderregel, dass jeweils der „SimulatedState“-Knoten geprüft und als Ausgang registriert werden muss. Dieser ist durch eine Pfadkonkatination erreichbar.

IO-Check
Der IO-Check ist lediglich eine graphische Möglichkeit um Ein- und Ausgänge während der Inbetriebnahme zu überprüfen und deren Status zu ändern. Hierbei wird die Boolean-Variable „RequestIoCheck“ verwendet, welche während der gesamten Dauer vom CPM auf true gesetzt ist. Eine Änderung von false auf true erwirkt die SCX-seitige Initialisierung des IO-Checks und eine Änderung von true auf false bewirkt die SCX-seitige Beendigung des IO-Checks. Diese Statusänderungen kann das CPM aus der Variablen „IoCheckState“ lesen.

Der Status wird dabei durch dieselbe Methode zurückgemeldet, welche auch die Request-Variable setzt. Dies erfolgt dort im Rahmen der Prüfung, ob eine Anfrage eines IO-Checks nötig oder möglich ist und dient der beschleunigten Rückmeldung an den CPM-Client. Eine Rückmeldung einer Statusänderung erfolgt ebenfalls über ein Monitored Item.

Wie aus Abbildung 6 hervorgeht, kann ein IO-Check nur vom Zustand „Allowed“ aus gestartet werden. Der Zustand „Not-Allowed“ ist hierbei ein Trap-Zustand und alle anderen Zustände werden nach Anforderung durch das SCX gesteuert. Der Zustand „Active“ ist die SCX-Bezeichnung für den CPM-Zustand „Released“.

Protokollierung

Bei der Protokollierung wird die Speicherung des Protokolls von der Geschäftslogik des CPMs übernommen, während der Gerätetreiber für die Abholung der Log-Einträge verantwortlich ist.

Wie aus Abbildung 7 ersichtlich wird, wird bei einer Aktivierung der Protokollierung für eine Steuerung ein Browse durchgeführt, um alle Knoten zu erhalten, welche der Benennungskonvention „Logging*“ folgen. Diese werden als Protokolldefinitionen bezeichnet. Von diesen Knoten wird dann unter Ausnutzung der Pfad-Knoten-ID durch Pfadkonkatination die Knoten für den Datenbanknamen, den Protokollnamen, den Header und die Größe des Ringpuffers mit einem Aufruf gelesen.

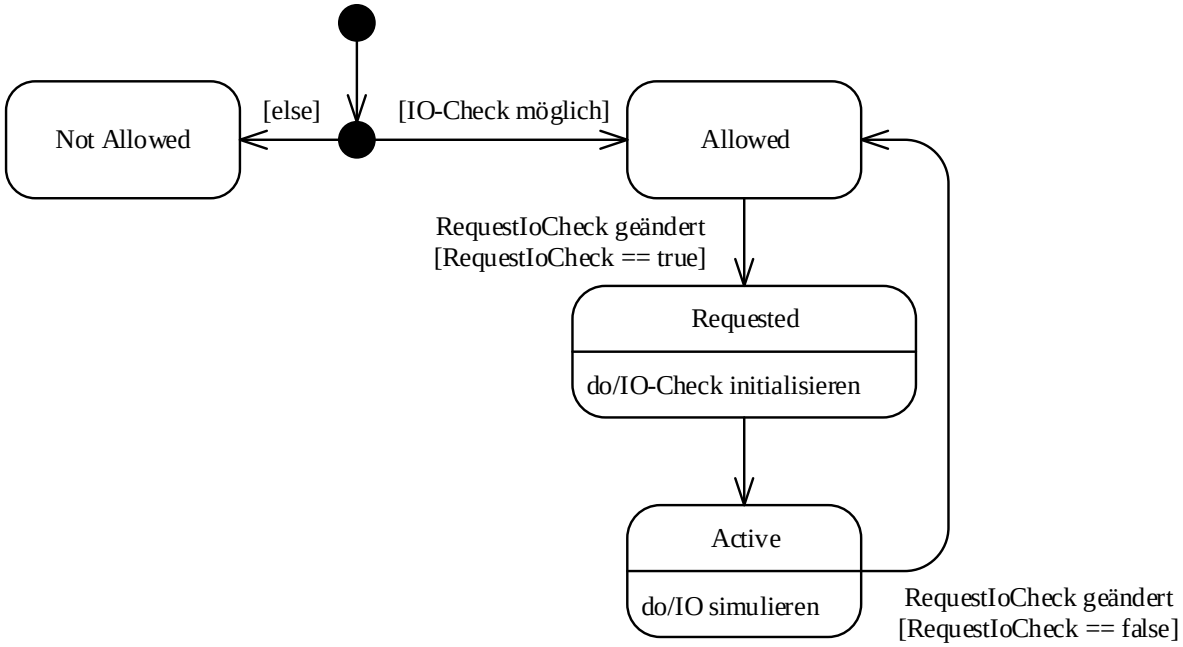

Abbildung 6: Zustandsdiagramm IO-Check

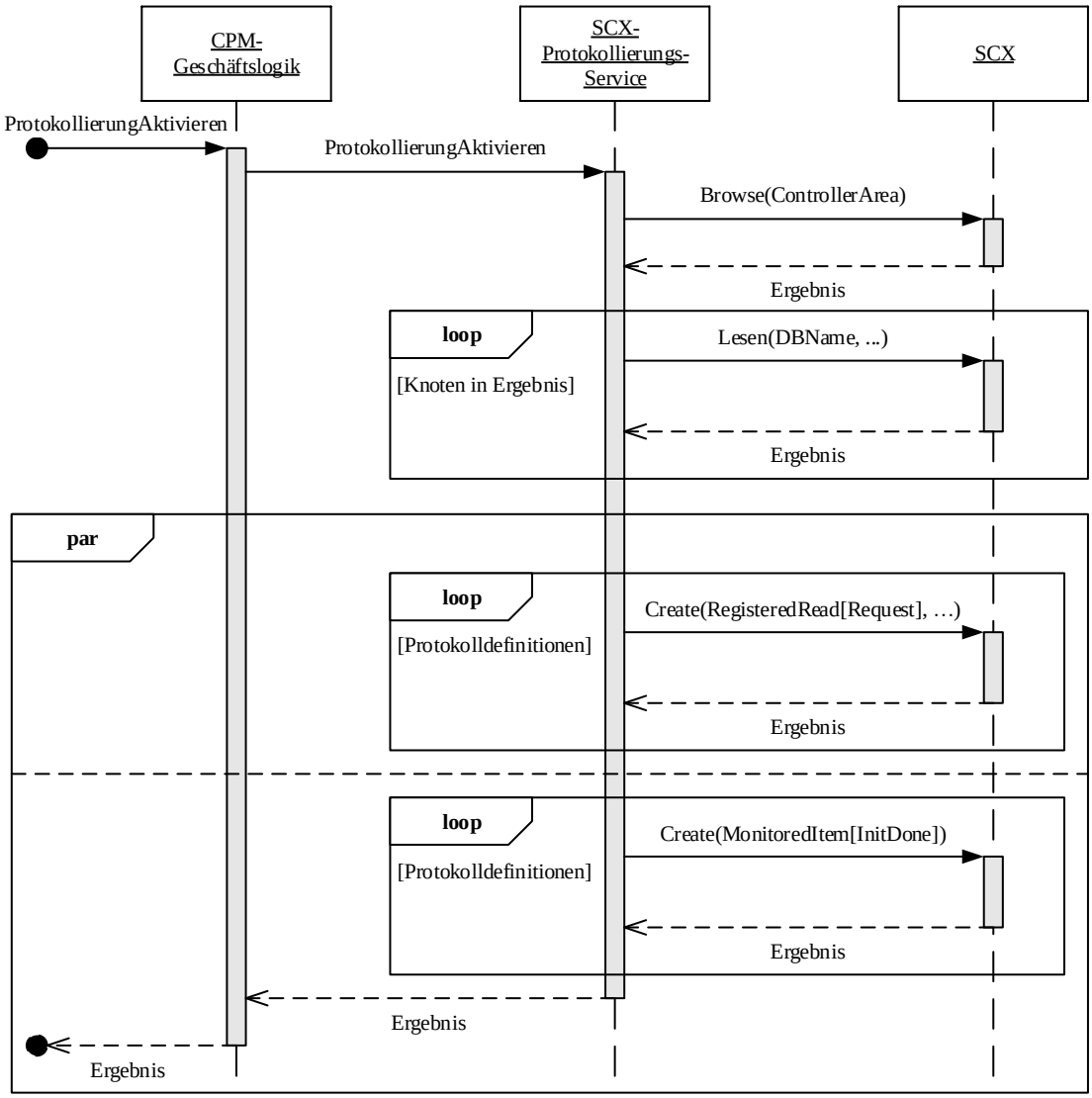

Abbildung 7: Sequenzdiagramm Protokollierungsaktivierung

Außerdem wird ein neuer Thread gestartet, welcher für jede Protokolldefinition das Request-Done-Laufnummernsystem, die Start- und Stoppindizes für den Ringpuffer, sowie die Eintragsknoten selbst als Registered Reads im SCX-OPC-UA-Server anlegt und somit veranlasst, dass diese im Arbeitsspeicher vorgehalten werden. Dies ist nötig, da bei einem Ringpuffer die Einträge schnellstmöglich abgeholt werden müssen, weil es ansonsten zu Verlusten aufgrund von Überschreibungen (Überlauf) kommen kann.

Die Abholung geschieht durch ein Monitored Item auf der Request-Laufnummer, sodass diese bei jeder Änderung ausgelöst wird. Wie in Abbildung 8 zu sehen, werden daraufhin, da das SCX einen Ringspeicher implementiert, die Knoten-IDs der abzuholenden Einträge erstellt. Nachdem die entsprechenden Log-Einträge gelesen wurden, werden sie an die Geschäftslogik übergeben. Es erfolgt eine Bestätigung über die Done-Laufnummer.

Das SCX aktiviert die Protokollierung je Protokoll durch einen Init-Flag. Treiberseitig wird dabei die Geschäftslogik mit dem Anlegen oder Abschließen einer Datei beauftragt und benötigte Monitored Items angelegt bzw. bei der Deaktivierung entfernt.

Device Capabilities

Die Device Capabilities legen die im Moment von der SCX unterstützten CPM-Funktionen fest. Da sie als Struktur angelegt sind, können sie in einem Lese-Aufruf ausgelesen werden. Für das CPM werden diese dann in ein Device Capabilities DTO übertragen. Dabei erfolgt zunächst für jede Capability die Prüfung, ob der Wert vorhanden, also ungleich null ist. Ist dies der Fall, so wird der Boolean-Wert übernommen, andernfalls wird false angenommen. Um auf Änderungen reagieren zu können wird ein Monitored Item auf die Struktur angelegt.

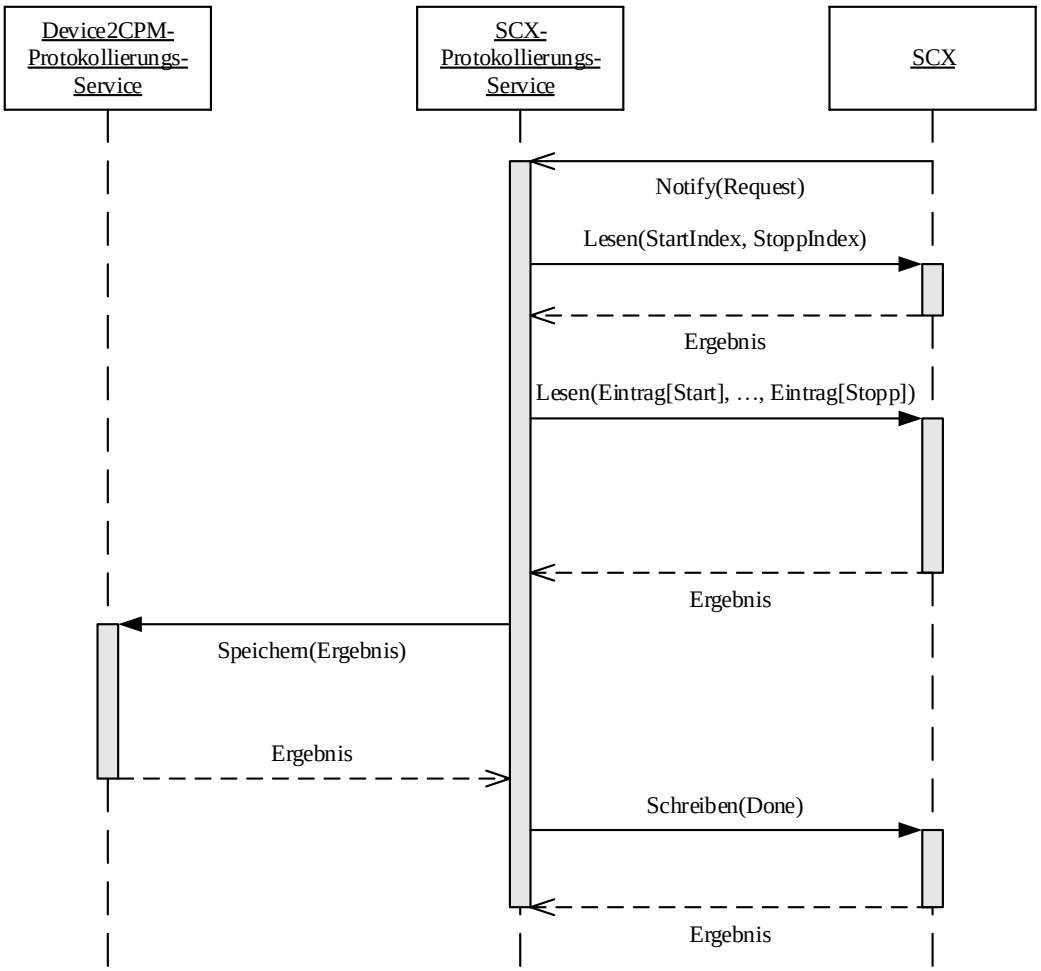

Abbildung 8: Sequenzdiagramm Protokolleintragsabholung

Bewertung der Lösung

Mock-Gerätetreiber
Diese Lösung erfüllt alle an sie gestellten Anforderungen. Zum einen sind die Mock-Implementierungen aus der Geschäftslogik entfernt worden, zum anderen sind alle OPC-UA-Aufrufe über die Treiberschicht und somit dem Mock-Gerätetreiber abstrahiert worden. Dadurch ist die Nutzung von anderen Kommunikationsprotokollen ermöglicht worden. Zudem ist durch die Package-Struktur des Treibers auch die Wiederverwendbarkeit bei Schnittstellenänderungen sichergestellt worden.

SCX-Gerätetreiber
Der SCX-Treiber erfüllt alle an ihn gestellten Anforderungen. Es ist die Schnittstelle der Treiberschicht vollständig implementiert worden, ebenso wie die Schnittstelle zum SCX selbst. Trotz der Änderungen an dieser durch die aktuell andauernde Entwicklung sind alle im Rahmen dieses Projekts angedachten Funktionen vollständig implementiert. Weil jedoch noch Zwischenlösungen in der SCX implementiert sind, welche diese Durchführung ermöglichen, ist noch viel Optimierungspotenzial für eine Weiterentwicklung der Gerätetreibers vorhanden. Ein Beispiel hierfür ist die Umstellung von numerischen auf Pfad-Knoten-IDs. Letztere ermöglichen einen direkten Zugriff auf Kind-Knoten mittels Pfadkonkatination, während bei ersteren ein Browse durchgeführt werden muss, um diese zu finden. Dabei werden auch viele weitere Knoten übertragen, was die Ausführung langsamer werden lässt und mehr Platz im Arbeitsspeicher einnimmt.

Dieses Problem besteht primär in der Parametrierung, da hier die SCX-Elemente durchsucht werden und diese in der Regel noch die numerischen Knoten-IDs nutzen. Da dies jedoch nur während eines Control Sync bzw. Control Imports passiert, ist im laufenden Betrieb davon in der Regel nichts zu merken. Jedoch kann, insbesondere bei großen Anlagen, die Dauer eines solchen Vorgangs dadurch im Vergleich zu einer Siemens S7-SPS mit dem entsprechenden, bereits optimierten Gerätetreiber, deutlich verlängert werden. Erste Messungen auf der für diese Entwicklungsarbeit zur Verfügung gestellten Test-SCX ergeben bspw. eine Control Sync Dauer von ca. 14 Sekunden, was im Vergleich zu einer größeren Test-S7-SPS mit einer Sync-Dauer von ca. fünf Sekunden einen deutlich langsameren Ablauf indiziert.

Zugriff nach oben (A)	Zugriff nach unten (B)	Zugriff nach oben erlaubt (C)	Zugriff nach unten erlaubt (D)	Ausgabe
0	0	X	X	1
0	1	X	0	1
0	1	X	1	0
1	0	0	X	1
1	0	1	X	0
1	1	0	X	1
1	1	1	0	1
1	1	1	1	0

Tabelle 2: Wahrheitswertetabelle

TEST

SCX-Test

Um den implementierten SCX-Gerätetreiber, insbesondere auf korrektes Verhalten, zu testen, wurde Mitte Dezember 2022 ein gemeinsamer Test mit dem SCX- und dem CPM-Server-Entwickler durchgeführt. Hierbei wurden, bis auf verschobene Pfade für einige Knoten, keine Fehler im Gerätetreiber gefunden.

Automatisierte Architekturtests

Das CPM verwendet automatisierte Unit- und Integrationstests, welche im Rahmen eines Continuous-Integration-Ansatzes bei jedem Build-Vorgang, und somit auch nach jeder Codeänderung, durchgeführt werden. In diese bestehende Testumgebung sollte ein neuer, ebenfalls automatisierter Architekturtest integriert werden. Dieser soll in Zukunft verhindern, dass neue Abhängigkeiten wie bspw. von Geräteschnittstellen in der Geschäftslogik entstehen.

Als Testframework wird ArchUnit verwendet. Dieses erlaubt die Definition von Architekturregeln auf Klassen- und Package-Ebene und besitzt vordefinierte Regeln für bestimmte Architekturtypen, wie bspw. für eine Schichtenarchitektur. Diese wurden im Rahmen der Regeldefinition für das CPM-Tool ebenfalls angewandt.

Ein großer Teil des Arbeitsaufwands für diese Architekturtests war die Erstellung einer neuen Kondition für ArchUnit, welche einen Verstoß verzeichnet, wenn Klassen Abhängigkeiten außerhalb der Super- oder Subpackages ihres eigenen Packages haben. Hierbei wird für jeden Zugriff geprüft, ob es ein Zugriff auf ein Superpackage ist, indem alle Subpackages des Ziels aufgelistet werden und das Package des Ursprungs darin gesucht wird. Ein Zugriff auf ein Subpackage wird genauso geprüft, nur Ursprung und Ziel sind vertauscht. Diese Erkenntnisse werden mit Hilfe eines logischen Ausdrucks in eine Aussage verwandelt, ob ein Zugriffsverstoß vorliegt oder nicht. Dieser ist in der Wahrheitswertetabelle in Tabelle 2 modelliert.

Zusätzlich zu der Wahrheitswertetabelle wird im ersten Schritt geprüft, ob sich sowohl Ursprung als auch Ziel des Zugriffs im selben Paket befindet. Wenn dies zutrifft, so ist ein Verstoß nicht gegeben.

ZUSAMMENFASSUNG

Das Projekt hat die Implementierung eines Gerätetreibers für die neue SCX-Steuerung, welche im Moment von SSI Schäfer entwickelt wird, zum Ziel. Ein solcher soll in Zukunft die Konfiguration und Inbetriebnahme der Maschinen erleichtern, indem die Funktionalität des CPM-Tools genutzt werden kann.

Der erste Schritt war die Absprache einer Schnittstelle mit dem SCX, da diese noch nicht existierte. Diese wurde dann im Verlauf der Implementierung auf Basis der Siemens S7-Schnittstelle immer wieder erweitert, sodass zum Schluss die hier beschriebene Schnittstelle entstanden ist.

Wie in Abbildung 9 zu sehen ist, wurde der SCX-Server im Laufe dieses Projekts um die Treiberschicht erweitert. Hierdurch konnte das grundlegende Designprinzip der Separation of Concerns, insbesondere im Vergleich mit der Situation in Abbildung 1, deutlich besser umgesetzt werden. Zudem ist die Umstellung der Mock-Implementierung auf die Treiberschicht sowie die Implementierung des neuen SCX-Gerätetreibers durchgeführt worden.

Der Gerätetreiber folgt wie die Geschäftslogik des CPMs dem dienstorientierten Modell verteilter Systeme. Hierbei sind die Dienste des Gerätetreibers unabhängig voneinander implementiert, was in Zukunft eine leichtere Änderung und Erweiterung ermöglicht. So können die zurzeit noch nicht im SCX enthaltenen CPM-Funktionen wie der Handbetrieb, die Datenmanipulation oder die Geräteoperationen in künftigen Versionen nachgereicht werden, sollten diese benötigt werden.

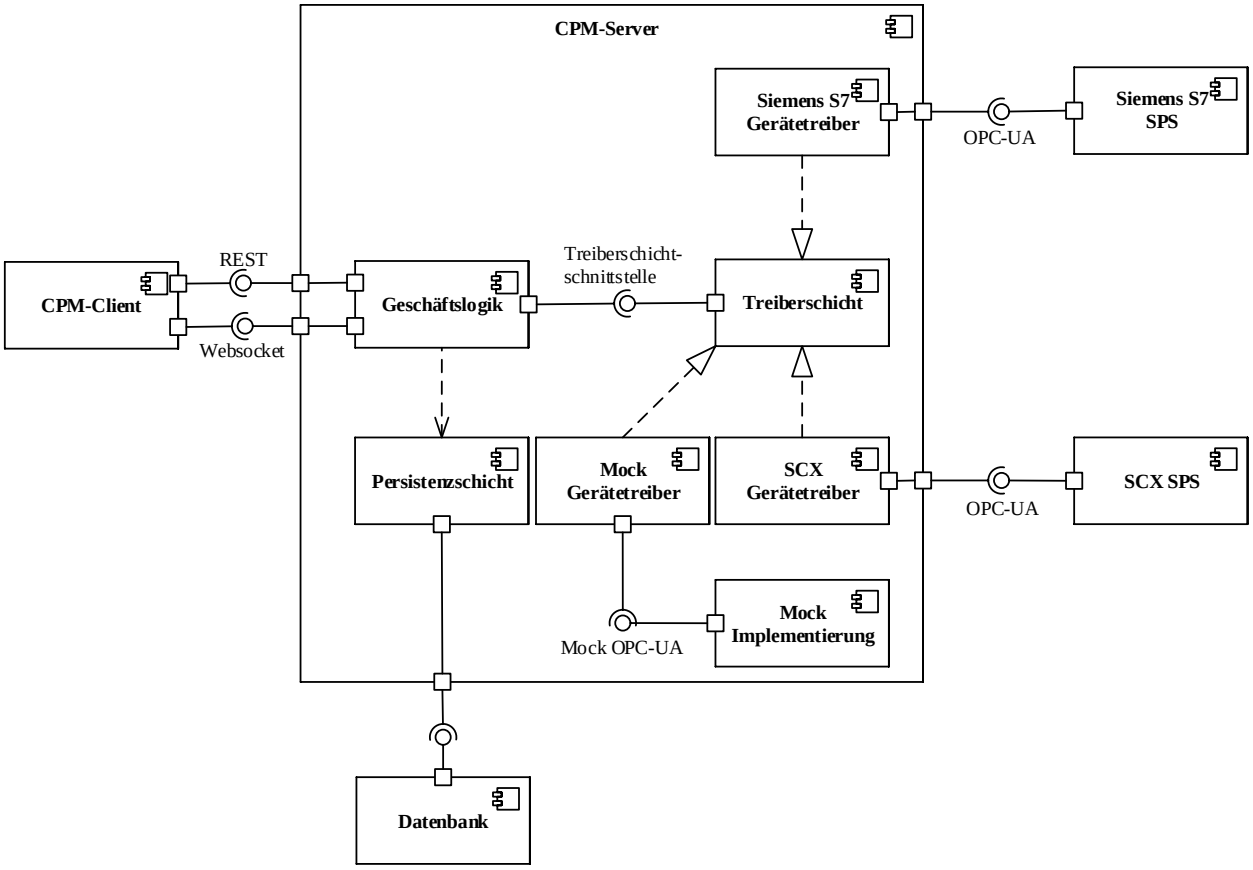

Abbildung 9: Komponentendiagramm des CPM-Tools im Überblick (Soll-Situation)

FAZIT

Die Implementierung des Gerätetreibers gestaltete sich schwieriger als zu Beginn angenommen, da sowohl in der Schnittstelle der Treiberschicht als auch in der Schnittstelle zum SCX während der Durchführung des Projekts immer wieder zu, teilweise auch konzeptionellen, Änderungen, wie bspw. bei der Protokollierung, gekommen ist. Entsprechend war eine andauernde Überarbeitung auch bereits als abgeschlossen betrachteter Dienste notwendig.

Zudem gab es Verzögerungen bei der Implementierung der vereinbarten Schnittstelle auf Seiten des SCX, welche sich aus den noch immer andauernden Lieferschwierigkeiten der Corona-Pandemie sowie der Energiekrise ergeben haben.

Trotz allem konnte die Implementierung auf Basis der aktuell gültigen Schnittstellenspezifikationen sowohl der Treiberschicht als auch des SCX bewerkstelligt werden. Dabei sind alle darin gestellten Anforderungen berücksichtigt worden und im gemeinsamen Test konnten keine funktionellen Fehler im SCX-Gerätetreiber gefunden werden.

Dementsprechend wäre dieser zur Veröffentlichung bereit, jedoch ist im Moment aufgrund der Verzögerungen noch kein Anwendungsfall in Sicht (die erste Anlage ist für 2024 anvisiert), sodass aufgrund der weiter andauernden Entwicklungen entschieden wurde, mit dieser zu warten bis sie benötigt wird. Dadurch wird verhindert, dass eine Version veröffentlicht wird, welche zum Zeitpunkt des ersten Einsatzes bereits veraltet wäre.

Dementsprechend ist die Zielsetzung des Projekts erreicht worden.