Quelle: Everett Historical/Shutterstock

Ein Protokollwandler für die Automatisierungsindustrie mit Apache PLC4X

Kommunikativer Aufbruch

Christofer Dutz

In der Automatisierungstechnik haben sich vorwiegend proprietäre Protokolle etabliert. Einheitliche Schnittstellen für den Zugriff auf diese Geräte fehlen momentan allerdings noch. Apache PLC4X soll dies ändern. Ein Projektbericht.

Vor allem in den Bereichen Big Data, Machine Learning und Virtualisierung haben sich in den letzten Jahren große Veränderungen abgespielt. Mittlerweile sind diese Techniken in allen Bereichen der Industrie unverzichtbar.

Doch während etwa die Automobilindustrie mehr und mehr Telemetriedaten aus den Fahrzeugen auswertet und in Services und die Gestaltung neuer Modelle einfließen lässt, bleibt die eigentliche Produktion davon größtenteils unberührt.

Die hier eingesetzten Lösungen sind vor allem geschlossene Systeme zur Produktionssteuerung. Sie kommunizieren in der Regel mit industriellen speicherprogrammierbaren Steuerungen (SPS) oder Programmable Logic Controllers (PLC) und koordinieren und überwachen das Zusammenspiel mehrerer Anlagen im Produktionsablauf.

Diese Systeme bauen üblicherweise die Firmen, die auch die im Unternehmen eingesetzten Steuerungen herstellen. Denn diese Steuerungen sind teilweise schon seit mehr als 30 Jahren im Einsatz. Zwar haben sie sich über die Jahre hinweg durchaus weiterentwickelt, allerdings war dieser Markt immer dominiert von proprietären Lösungen einiger großer Hersteller. Und die haben sich große Mühe gegeben, ein Unternehmen, das sich einmal für ihre Systeme entschieden hatte, auch in Zukunft an sich zu binden. So sind es die Kunden seit jeher gewohnt, dass sie sämtliche Produkte bei „ihrem" Steuerungshersteller beziehen müssen.

Niemand hat die Absicht, eine Mauer zu errichten …

Ältere Leser kennen dieses Vendor-Lock-in noch aus den Zeiten der IBM- und Siemens-Mainframes. Während aber in der heutigen Softwarewelt eine derart proprietäre Herangehensweise nicht mehr durchsetzbar ist, haben sich Begriffe wie „offene Schnittstellen" noch nicht in der Welt der Industrieproduktion herumgesprochen. Zwar versucht die Industrie gerade mit OPC-UA einen neuen Standard zu implementieren, allerdings erfordert dieser eine Modifikation der bestehenden Anlagen. Außerdem erhöht ein zusätz-

licher OPC-UA Server die Rechenlast auf der Steuerung und eventuell muss auf größere Steuerungsmodelle ausgewichen werden.

Vermutlich ist die mangelhafte fachliche Diffusion zwischen Automatisierungsindustrie und Softwareherstellern dafür verantwortlich. Denn bei allen Gesprächen mit Vertretern der Industrie fiel auf, dass Automatisierungstechniker und Softwareentwickler in komplett unterschiedlichen Welten zu leben scheinen und unterschiedliche Sprachen sprechen.

Insbesondere gilt dies für Diskussionen über den Begriff „Sicherheit“. Für beide Berufsgruppen ist Sicherheit von größter Bedeutung. Leider verstehen sie darunter etwas komplett Unterschiedliches. Während der typische Entwickler damit Datenkonsistenz, Schutz vor Manipulation, Schutz vor unberechtigtem Zugriff und Ähnliches assoziiert, denkt der Mitarbeiter aus dem Automatisierungsumfeld vor allen an Prozesssicherheit. Für ihn bedeutet „Sicherheit“, dass Motor X auch wirklich garantiert zur richtigen Zeit stehen bleibt und Ventil Y genau dann öffnet, wenn Sensor Z einen Druckanstieg über einen bestimmten Wert meldet. Das Englische kennt dafür die unterschiedlichen Begriffe Security und Safety, im Deutschen gibt es diese Unterscheidung nicht. Sicherheit kann sowohl Sicherheit vor Angriffen bedeuten als auch Betriebssicherheit.

Kein Wunder, dass sich auch in beiden Richtungen technisches Know-how nicht gleichmäßig verteilt. Im IoT-Umfeld hat sich in den letzten Jahren beispielsweise eine Vielzahl von Protokollen für den Datenaustausch entwickelt. Höchstens MQTT wird einem Mitarbeiter der Automatisierungstechnik bekannt vorkommen.

Genauso sind Softwerkern die industriellen Protokolle ein Buch mit sieben Siegeln. Möglicherweise sind einige der Industrieprotokolle für die eine oder andere IoT-Lösung gut geeignet, genau wie bestimmte Security-Features moderner IoT-Protokolle für die Industrie nicht uninteressant wären. Aber da rührt sich wenig.

Das alte Lied vom Vendor-Lock-in

Die klassischen Hersteller im Automatisierungsmarkt sind über diesen Mangel an fachlichem Austausch keineswegs unglücklich. Wer ständig hört, dass man als „Industrie-4.0-Unternehmen“ in „die Cloud“ muss, aber eventuell die damit einhergehenden Konsequenzen und Abhängigkeiten noch nicht zu Ende durchdacht hat, dürfte eine proprietäre Lösung seines Steuerungslieferanten ohne Zögern adaptieren. Auch dann, wenn diese für das spezifische Problem gar nicht die bestmögliche Wahl ist.

So stellten auf der SPS IPC Drives 2017 in Nürnberg, einer der größten Messen der Steuerungs- und Regelungstechnik, fast alle großen Anbieter proprietäre Produkte für Big Data und Cloud-Computing vor, die durchweg einen Umstieg von Plattform A auf Plattform B schwer bis unmöglich machen.

Wer sich dieser engen Bindung an eine Plattform nicht unterwerfen will, dem steht momentan nur eine Option zur Verfügung: der Einsatz eines kommerziellen Hard- oder Softwareprotokolladapters, der Steuerungsdaten ausliest und in ein anderes Protokoll umsetzt; meist solche, die Maschinendaten per MQTT verfügbar machen.

Einige Unternehmen bieten Hardware-Gateways an, die alle Sensordaten an die eigene Cloud weiterleiten. „Eigene Cloud“ meint hierbei allerdings nicht eine Cloud in der Hand des Kunden, sondern Systeme des Gateway-Anbieters. Der Kunde darf dann dort auf seine Daten zugreifen.

Das ist natürlich etwas anderes als ein direkter Zugang zu den Daten der Steuerung. In der Regel müssen diese Systeme vorab konfiguriert werden, um bestimmte Daten in einer bestimmten Frequenz abzurufen und weiterzuleiten. Dieses Vorgehen reduziert die Flexibilität und erhöht die Latenz erheblich. Mir ist auch kein Anbieter bekannt, der eine „Cloud-Lösung“ im Portfolio hat, bei der die Daten nicht das Unternehmen verlassen. Die Kunden müssen sich also auf die Sicherheit und die Verfügbarkeit ihrer Daten auf den Systemen ihres Anbieters verlassen.

Früher war nicht alles besser

Dass nicht direkt mit den Steuerungen kommuniziert wird, liegt vor allem an der mangelnden Verfügbarkeit von Treibern und Softwareadaptern für den Zugriff darauf. Zwar gibt es eine Vielzahl von Bibliotheken, die die einzelnen Protokolle der SPS (teilweise) implementieren und somit theoretisch direkten Zugang zu den Steuerungen ermöglichen. Leider schließen deren Lizenzen aber eine kommerzielle Nutzung de facto aus, etwa die GPL.

Ein weiteres Problem besteht darin, dass diese Bibliotheken oftmals veraltet

Incubating: In der Probephase

Enthält der Projektname eines Apache-Projekts den Zusatz „incubating“, wie bei Apache PLC4X und Apache Edgent, bedeutet dies, dass es noch unter der Aufsicht des Apache Incubator Project Management Committee steht. Dies verlangt die ASF von allen neuen Projekten, bis diese unter Beweis gestellt haben, dass sich ihre Infrastruktur, Kommunikationskultur und Entscheidungsfindungsprozesse auf eine Art stabilisiert haben, die konsistent ist mit der anderer erfolgreicher ASF-Projekte. Der Inkubationsstatus ist folglich nicht zwangsläufig ein Indikator für Vollständigkeit und Stabilität des Codes, sondern besagt vielmehr, dass sich das Projekt als Ganzes noch die volle Unterstützung der ASF verdienen muss.

Hilfe erwünscht

Auf dem weiten Weg zu einer universellen Schnittstelle für programmierbare Steuerungen sind wir für jede Hilfe dankbar; sei es beim Implementieren weiterer Treiber für neue Steuerungen und Protokolle oder beim Bereitstellen von Detailinformationen. So sind einige wichtige Protokolldetails noch nicht spezifiziert – hier ist Wissen für uns das höchste Gut. Auch die Verwendung von PLC4X in eigenen Projekten nebst Feedback ist für uns sehr wichtig. Letztendlich wollen wir eine Lösung schaffen, mit der viele Entwickler auf der ganzen Welt spannende Projekte realisieren können.

Der wohl beste Weg, informiert zu werden und am Projekt teilzuhaben, ist es, sich auf unseren Mailinglisten anzumelden – siehe ix.de/ix1807092.

iX-TRACT

- In der Automatisierungsindustrie sind nahezu sämtliche Protokolle proprietär.
- Das erschwert den Softwarezugriff auf die Geräte.
- Bei Codecentric wurde mit Apache PLC4X eine einheitliche Treiberschicht implementiert.

sind. Meist handelt es sich um Eins-zu-eins-Portierungen in die Jahre gekommener C- und C++-Treiber. Viele stammen noch aus Zeiten, in denen Multi-Core-Prozessoren und Parallelität kein Thema waren. Ein gleichzeitiges Versenden mehrerer Requests würde zu unvorhersehbaren Effekten führen, da sich alle Request den exakt gleichen Speicherbereich teilen.

Was also fehlt, ist die Verfügbarkeit von Treibern zur direkten Kommunikation mit industriellen Steuerungen, mit Lizenzen, die den kommerziellen Einsatz ermöglichen und dabei die Vorteile und Fähigkeiten moderner Sprachen und Computersysteme nutzen.

Genau diese Lücke soll Apache PLC4X (incubating) (siehe Kasten „Incubating … “) schließen. Denn uns fiel bei codecentric immer wieder auf, dass Unternehmen aus dem produzierenden Gewerbe nach Unterstützung in den Bereichen Virtualisierung und Big Data fragten. Zwar waren die vorhandenen Lösungen aus den erwähnten Gründen suboptimal, doch mangels Alternativen waren wir immer wieder gezwungen, auf die erwähnten Protokolladapter zurückzugreifen.

Darum wurde ein „Innovationsprojekt“ initiiert. Ziel ist eine universelle Treiber-Suite, die den einheitlichen Zugriff auf Steuerungen verschiedenster Hersteller ermöglicht. Wir wollten ausdrücklich keinen kommerziellen Treiber entwickeln, den man an die Kunden verkaufen kann. Vielmehr sollte PLC4X als Open-Source-Projekt im Incubator der Apache Software Foundation untergebracht werden. Das ist auch kurz vor Weihnachten 2017 gelungen. Somit steht fest, dass das Projekt unter der Apache-2.0-Lizenz entwickelt und verfügbar sein wird.

Die leidigen Lizenzen

Problematisch bei diesem Projekt ist, dass viele Informationen über einige der Protokolle zwar allgemein zugänglich, aber die Quellen in der Regel ungünstig lizenziert sind – zum Beispiel durch die GNU Public License. Die Implementierung einer Software anhand dieser Informationen würde damit unmittelbar als „derivative work“ gelten und somit ebenfalls unter die GPL-Lizenz fallen. Da wir dies auf jeden Fall ausschließen wollten, besteht ein Großteil der Arbeit an Apache PLC4X nicht aus der reinen Implementierung der Treiber, sondern im Aufspüren von Informationen für die Implementierung und der Dokumentation eben dieser Quellen.

Weil diese Detektivarbeit sehr aufwendig ist, sehen wir eine Implementierung allein in Java als Verschwendung an. Das „X“ am Ende von PLC4X soll andeuten, dass wir uns nicht auf eine Plattform oder Sprache beschränken wollen. Derzeit gibt es von Apache PLC4X eine Java-Implementierung und einen Scala-Wrapper. Später wollen wir auch den Einsatz anderer Sprachen wie JavaScript, C und C# ermöglichen.

Unterstützt werden momentan S7-Steuerungen von Siemens via S7-Protokoll (nicht Profinet), Beckhoff-Steuerungen durch das ADS-Protokoll sowie alle Geräte, die das Modbus- und Ethernet-Protokoll beherrschen. Für Ethernet/IP und OPC-UA haben die Arbeiten schon begonnen.

Ziel des Projekts ist eine API, mit der man Anwendungen für programmierbare Steuerungen schreiben kann, die unabhängig vom Modell oder Protokoll der Steuerung sind. Dies ermöglicht die Umstellung einer Anwendung auf eine andere Steuerung oder ein anderes Protokoll, ohne dass die Applikation dazu geändert werden muss.

Unterschiede und Gemeinsamkeiten

An zwei Stellen verbergen sich protokollabhängige Unterschiede: bei der Adressierung der Steuerung selbst sowie von deren Ressourcen.

Für den Verbindungsaufbau und die Adressierung von Ressourcen kommen Strings zum Einsatz. Diese Connection-Strings ähneln den klassischen JDBC Connection Strings.

PLC4X-Programme benötigen zur Entwicklung lediglich das PLC4X-API-Modul, das die Schnittstelle bereitstellt. Die eigentliche Konnektivität realisieren separate Treibermodule zur Laufzeit. Möchte man eine Software von einem Steuerungstyp auf einen anderen ändern, sind nur das passende Treibermodul im Classpath einzubinden und die Connection- und Adress-Strings anzupassen.

Ein Beispiel: Der Connection-String s7://192.168.0.1/0/0 baut via S7-Protokoll eine Verbindung zu einer Steuerung mit der IP-Adresse 192.168.42.100 auf. Die letzten beiden Teile der URL sind treiberabhängig und adressieren im S7-Protokoll „Rack“ und „Slot“ der gewünschten Steuerung. Vorausgesetzt wird, dass im Classpath der Applikation eine Treiberimplementierung für das S7-Protokoll enthalten ist.

Listing 1: Parameter auslesen

```
public static void main(String[] args) throws Exception {
  try (PlcConnection plcConnection =
    new PlcDriverManager().getConnection(args[0])) {

    Optional<PlcReader> reader = plcConnection.getReader();

    // Check if this connection support reading of data.
    if (reader.isPresent()) {
      PlcReader plcReader = reader.get();

      // Parse an address string.
      Address inputs = plcConnection.parseAddress(args[1]);

      ////////////////////////////////////////////////////////
      // Read synchronously ...
      // NOTICE: the ".get()" immediately lets this thread pause till
      // the response is processed and available.
      TypeSafePlcReadResponse<Byte> plcReadResponse = plcReader.read(
        new TypeSafePlcReadRequest<>(Byte.class, inputs)).get();

      System.out.println("Inputs: " + plcReadResponse.getResponseItem()
        .orElseThrow(() -> new IllegalStateException("Not found"))
        .getValues().get(0));

      ////////////////////////////////////////////////////////
      // Read asynchronously ...
      CompletableFuture<TypeSafePlcReadResponse<Byte> asyncResponse =
        plcReader.read(new TypeSafePlcReadRequest(Byte.class, inputs));

      asyncResponse.thenAccept(bytePlcReadResponse -> {
        Byte dataAsync = bytePlcReadResponse.getResponseItem()
          .orElseThrow(() -> new IllegalStateException("Not found"))
          .getValues().get(0);
        System.out.println("Inputs: " + dataAsync);
      });

      // do something else ...
    }
  }
  // Catch any exception or the application won't be able to finish if something goes wrong.
  catch (Exception e) {
    logger.error("S7 PLC reader sample", e);
  }
}
```

Es werden auch andere Protokolle als IP unterstützt. Unabhängig vom Modell der Steuerung und des verwendeten Protokolls sieht der Code zum Auslesen eines Parameters aus einer Steuerung immer aus wie in Listing 1 und kann mit den folgenden Argumenten aufgerufen werden:

```
s7://192.168.0.1/0/0 INPUTS/0
```

Damit kann sich das Programm per S7-Protokoll mit einer S7-Steuerung mit der IP-Adresse 192.168.0.1 verbinden und dort ein Byte mit allen acht Zuständen der ersten acht Digitaleingänge abfragen.

Bei der Entwicklung der API wurde Wert darauf gelegt, dass der Entwickler möglichst viele Freiheiten hat. Im Codebeispiel sieht man eine synchrone und eine asynchrone Leseoperation.

Abstraktion, Emulation und Simulation

Nicht alle Steuerungen und Protokolle bieten identische Funktionen an. Das Projektteam löst dies, indem es die direkt verfügbaren Funktionen auch direkt implementiert und die nicht vorhandenen nur emuliert. Ein Beispiel ist das Lesen und Schreiben von Daten. Hier implementiert die PLC4X-API generell das Lesen und Schreiben mehrerer Werte in einem Request. Allerdings beherrschen Siemens-Steuerungen via S7-Protokoll lediglich das Lesen mehrerer Adressbereiche in einem Request, nicht aber das Schreiben von mehr als einem Wert. Darum simuliert Apache PLC4X für den Anwender transparent diese fehlende Funktion, indem es hinter den Kulissen mehrere Write-Requests absetzt.

Da es auch Funktionen gibt, die gar nicht emuliert werden können, haben wir funktionale Interfaces eingeführt, die sämtliche Operationen einer Domäne bereitstellen. Das PlcConnection-Objekt verfügt über Get-Methoden, um auf diese zuzugreifen. Sie liefern Java-Optionals zurück, mit denen man bequem prüfen kann, ob eine Domäne überhaupt existiert. Momentan sind die folgenden funktionalen Interfaces definiert:

- PlcReader (Lesen von Ressourcen)
- PlcWriter (Schreiben von Ressourcen)
- PlcSubscriber (Abonnement von Ressourcen – Steuerung informiert aktiv über Alarme, Events und Werteänderungen)

Einige Protokolle erlauben es, Funktionen auf der Steuerung aufzurufen. Hier dürfte es bald weitere Interfaces geben, sobald entsprechende Treiber zur Verfügung stehen.

Denkbar sind ebenfalls Treibervarianten, die aus Sicherheitsgründen nicht alle Funktionen beherrschen, etwa Read-only-Varianten.

Mit Apache PLC4X ist es möglich, einen eigenen Protokolladapter zu bauen, der alle möglichen Daten aus Steuerungen abruft und zum Beispiel in einer Datenbank ablegt. Oder man könnte sie mittels eines anderen Protokolls wie Apache Kafka weiterleiten und anschließend wie gewohnt verarbeiten.

Vom sinnvollen Umgang mit IoT-Daten

Im direkten Vergleich mit OPC-UA versucht PLC4X, auf bestehende Anlagen zuzugreifen, ohne bei diesen einzugreifen, indem neue Treiber entwickelt werden. OPC-UA hingegen verwendet bestehende Treiber , um auf angepasste Anlagen zuzugreifen.

Ein sogenanntes Retrofitting von Bestandsanlagen, ohne diese anzupassen, ist somit meistens nur mit Apache PLC4X möglich. Allerdings ist eine direkte und ungefilterte Weiterleitung roher Maschinendaten nicht unbedingt von Vorteil. Denn es werden je nach Granularität Unmengen an unnützen Daten erhoben, übertragen und gespeichert. In vielen Fällen könnte es sich darum als günstiger erweisen, die Daten lokal vorzuverarbeiten.

Zu diesem Zweck kommt ein weiteres neues Apache-Projekt ins Spiel: Apache Edgent (incubating), eine kompakte Plattform zur eventbasierten, lokalen Vorverarbeitung von Datenströmen. Edgent ist leichtgewichtig und flexibel aufgebaut und für den Betrieb auf sehr kleinen Systemen geeignet. Es bildet eine ideale Brücke, um Daten zum Beispiel via Apache PLC4X aus den Steuerungen auszulesen, lokal vorzuverarbeiten oder zu puffern und anschließend an andere Systeme weiterzuleiten, etwa an Apache Kafka. Apache PLC4X bietet von Haus aus Module für die Integration in Apache Edgent an.

Listing 2 zeigt, wie einfach eine Apache-Edgent-Applikation mit PLC4X implementiert werden kann. (js@ix.de)

Listing 2: Apache-Edgent-Applikation

```
import com.google.gson.JsonObject;
import org.apache.edgent.connectors.kafka.KafkaProducer;
import org.apache.edgent.function.Supplier;
import org.apache.edgent.providers.direct.DirectProvider;
import org.apache.edgent.topology.TStream;
import org.apache.edgent.topology.Topology;
import org.apache.plc4x.edgent.PlcConnectionAdapter;
import org.apache.plc4x.edgent.PlcFunctions;

import java.util.HashMap;
import java.util.Map;
import java.util.concurrent.TimeUnit;

public class PlcLogger {

  public static void main(String[] args) {
    // Get a plc-adapter that's connected to a Siemens S7 PLC.
    PlcConnectionAdapter plcAdapter = new PlcConnectionAdapter( s7://192.168.0.1/0/0 );

    // Create a new Edgent topology.
    DirectProvider dp = new DirectProvider();
    Topology top = dp.newTopology( plc-logger );

    // Create a supplier that is able to read the state of the first digital input of the PLC.
    Supplier<@$p>Boolean> plcSupplier = PlcFunctions.booleanSupplier(plcAdapter,  INPUTS/0.0 );

    // Start polling our plc source in the given interval.
    TStream<@$p>Boolean> source = top.poll(plcSupplier, 100, TimeUnit.MILLISECONDS);

    // Convert the byte into a string.
    TStream<@$p>String> jsonSource = source.map(value -> {
      JsonObject jsonObject = new JsonObject();
      jsonObject.addProperty( main-drive , value ?  on  :  off );
      return jsonObject.toString();
    });

    // Publish the stream to a Kafka topic.
    Map<@$p>String,Object> kafkaConfig = new HashMap<@li.dunkelrot>();
    kafkaConfig.put( bootstrap.servers ,  192.168.0.43:9092 );
    KafkaProducer kafka = new KafkaProducer(top, () -> kafkaConfig);
    kafka.publish(jsonSource,  values );

    // Submit the topology to the provider (Start the stream).
    dp.submit(top);
  }

}
```

Christofer Dutz

arbeitet bei codecentric als Senior IT-Consultant.