7 Sensor Technology Adapter Gateway (STAG) – Sustaining/Kooperativ

Mitautoren: Dovydas Girdvainis und Gerhard Marki

Dieses Praxisbeispiel zeigt eine Lösung auf, wie Sensoren einfach in eine Industrie-4.0-konforme Kommunikationsinfrastruktur eingebunden werden können. Das ***S**ensor **T**echnology **A**dapter **G**ateway* kurz STAG übernimmt hierbei die Vermittlung zwischen den Kommunikationsprotokollen auf Ebene der vernetzten Sensorsysteme mit der MES- bzw. ERP-Ebene, indem es eine automatische Abbildung in die OPC-UA-Welt übernimmt und so Sensorinformationen einfach durch die Leitsysteme abgefragt werden können. Da es sich um einen technologischen Lösungsvorschlag für verschiedene Anwendungen handelt, entfallen in diesem Praxisbeispiel die Anwendungsfall spezifischen Spinnendiagramme.

7.1 Beschreibung des Umfelds

Gerade die Einbindung von Sensoren und deren Informationen in andere Systeme wird durch die Verwendung etablierter Standards vereinfacht. Dies gilt sowohl bei einer Einbindung in höhere Systeme, aber auch bei einer Anbindung z. B. an Maschinensteuerungen wie in der kooperativen Stufe. Selbst beim Aufbau einer in sich geschlossenen Sustaining-Lösung kann die Verwendung etablierter Standards die Implementierung und den Zugriff auf die Daten vereinfachen.

OPC UA ist ein aufkommender Standard, um den Hallenboden mit der IT-Welt zu verbinden. Wie schon in Abschnitt 5.3 dargestellt, beinhaltet OPC UA neben der reinen Datenübertragung auch Möglichkeiten der Datenmodellbeschreibung sowie Funktionalitäten für eine sichere Kommunikation. Dieser Funktionsumfang hat jedoch zur Folge, dass die Implementierung einer OPC UA Schnittstelle nicht unerhebliche Aufwände in der Implementierung, aber auch Anforderungen an die zugrundeliegende Hardware erfordern. Je nach Funktionsumfang sind mittlere Mikroprozessoren mit einem ***R**eal**t**ime **O**perating **S**ystem* (RTOS) und einigen kByte-Speicher bis hin zu embedded Linux-Systemen mit ARM oder X86-Prozessor erforderlich. Gerade für kleinere Sensoren ist dieser Aufwand nicht verhältnismäßig und übersteigt häufig auch die Kompetenzen eines klassischen Sensorentwicklers. Damit aber auch kleinere Sensorsysteme mit wenig Aufwand in die OPC UA Welt integriert werden können, wurde von Hahn-Schickard das *Sensor Technology Adapter Gateway* (STAG) entwickelt, das Thema dieses Praxisbeispiels sein soll.

7.2 Herausforderung

Wie bereits im vorherigen Kapitel angedeutet, stellt die Komplexität von OPC UA an sich schon eine Herausforderung dar, die aber mit nicht unerheblichen Vorteilen und Mehrwerten einhergeht. Mit STAG soll aber nicht nur einen Adapter realisiert werden, der OPC UA sprechen kann, sondern eine modulare Lösung entwickelt werden, die mit verschiedenen Technologien erweitert werden kann und als Vermittler bzw. Übersetzer zwischen der Sensorwelt und der IT-Welt auftreten kann. Aus dieser Vision eines Multiprotokoll-Übersetzer hat sich die Idee von STAG mit den folgenden Unterzielen bzw. technologischen Herausforderungen ergeben:

- **Einfache Anwendbarkeit**
  Das wichtigste Ziel von STAG ist die einfache Anwendbarkeit. Der Sensorentwickler selbst soll hierbei keinen bzw. einen möglichst geringen Aufwand haben, um seinen Sensor an STAG und damit auch die IT-Welt anzukoppeln. Ein Sensor soll sich bei STAG anmelden und wird dann automatisch über die Schnittstelle in die IT-Welt sichtbar und für andere Systeme zugreifbar.
- **Unabhängigkeit von konkreten Sensorsystemen**
  Um die Anwendbarkeit möglichst hoch zu gestalten und den Konfigurationsaufwand möglichst gering zu halten, ist STAG so entwickelt, dass es unabhängig von konkreten Sensorausprägungen ist. STAG soll «*out of the box*» verschiedene Sensorsysteme mit unterschiedlichen Messwerten und Eigenschaften unterstützen. Erreicht wird dies durch die Nutzung einer Sensorbeschreibung, wie sie z. B. im ***L**ight**w**eight **M**achine-**to**-**M**achine Standard* (LwM2M) oder in den Bluetooth GATT-Profilen (**G**enerisches **Att**ributprofil) schon vorgesehen ist.
- **Modularer und erweiterbarer Aufbau zur Unterstützung verschiedener Standards sowohl auf Sensor wie auch IT-Seite**
  Der Systemaufbau von STAG ist so gewählt, dass STAG selbst durch verschiedene Technologie- oder Protokolladapter erweitert werden kann. Dieser modulare Aufbau ermöglicht, dass STAG auch nachträglich noch mit weiteren Schnittstellen ergänzt werden kann oder auch variabel zwischen verschiedenen Protokollen umgeschaltet werden kann.
- **Gemeinsames Verständnis und Datenmodell von Sensorsystemen und Sensorinformationen**
  Damit Sensorinformationen und -daten zwischen verschiedenen Protokollen und Standards übersetzt und transformiert werden können, ist eine einheitliche Definition von Sensorinformation notwendig. Dieses zentrale Datenmodell wird als gemeinsamer Kern der verschiedenen Technologie- und Protokolladapter verwendet und erleichtert die Übersetzung zwischen verschiedenen Technologien signifikant, da dann pro Technologie bzw. Protokoll jeweils nur eine Abbildung in das zentrale Datenmodell erfolgen muss und nicht n-Abbildungen in n unterstützte Technologien.
- **Zentrale Übersetzungstechnologie für das Abbilden zwischen unterschiedlichen Datenmodellen**
  Das Herz von STAG ist das bereits erwähnte gemeinsame zentrale Datenmodell. Wie auch beim Menschen nützt das Herz nicht ohne die Adern, die den Blutfluss von und zum Herzen sicherstellen. Bei STAG nimmt die Übersetzungstechnologie die Rolle der Adern ein. Durch sie wird erst ermöglicht, dass die Daten, die von den Sensoren kommen in das gemeinsame Datenmodell abgebildet werden und von dort dann auch in die Datenadapter weitergeleitet werden, die die Daten dann wieder über spezifische Schnittstellen den höheren Systemen in der IT-Welt zur Verfügung stellen.

Wie diese Ziele bzw. Herausforderungen für STAG erreicht wurden und welche technischen Ansätze dafür umgesetzt wurden, erklären die folgenden Abschnitte.

7.3 Lösung

Das Hauptziel des STAG-Projektes ist die Trennung von Sensorsystemen von verschiedenen IIoT-Plattformen oder anderen Anwendungen, die die Sensordaten konsumieren. Diese Aufteilung ermöglicht es dem Endsystembenutzer, die von den Sensoren generierten Daten über IIoT-Plattformen oder -Anwendungen wiederzuverwenden. Dazu müssen keine Konfigurationsdaten

innerhalb der besagten Anwendungen manuell geändert oder erweitert werden. Um dies widerzuspiegeln, ist das STAG Projekt in drei verschiedene Bereiche unterteilt:

1. ***Technology Adapters**:* Module, die für den Aufbau von Sensorabstraktionen zuständig sind;
2. ***STAG-Core***: zentrales Modul, das für die interne Modellverwaltung, das Event Handling, die Informationsverarbeitung sowie die Bereitstellung der Schnittstellen für *Technology* und *Data Consumer Adapter* zuständig ist;
3. ***Data Consumer Adapters***: Module, die die bereitgestellten Sensordaten verwenden und über eine Schnittstelle zur Verfügung stellen.

In diesem Abschnitt werden die technischen Details von STAG durchgegangen, um die Funktionsweise besser verstehen zu können und die Probleme zu identifizieren, bei denen STAG helfen kann. Zunächst wird jeder Modultyp betrachtet und die Basisterminologie definiert, die in diesem Kapitel verwendet werden wird. Zweitens wird die Komponentenübersicht vorgestellt, das den internen Projektaufbau erklärt, sowie wofür jede Komponente zuständig ist und wie die Sensoren von den Datenzugriffsschnittstellen getrennt werden. Drittens wird das interne Informationsmodell erklärt sowie was es definiert und wie es verwendet wird. Und abschließend wird erläutert, wie das automatische Mapping zwischen den Informationsmodellen von Sensoren und Datenkonsumenten erfolgt.

7.3.1 Definition der Module

Zunächst erfolgt die Beschreibung des ***Technology Adapters***: Dieser ist für den Aufbau der Sensorabstraktionen zuständig, allerdings wurde nicht erklärt, welche Art von Sensoren unterstützt werden. Im Allgemeinen unterscheidet man verschiedene Sensorsysteme nach ihren Kommunikationsprotokollen und nicht nach ihrem Zweck, daher ist jeder *Technology Adapter* auf ein Kommunikationsprotokoll spezialisiert, wie Bluetooth, Profibus etc. Dies ist darauf zurückzuführen, dass verschiedene Kommunikationsprotokolle oft sehr unterschiedliche Implementierungen und Semantiken haben, die sie verwenden. Zum Beispiel ist Bluetooth ein drahtloses Kommunikationsprotokoll, das Daten als «*Characteristics*» für verschiedene Ein-/Ausgabepunkte definiert. Im Gegensatz dazu ist Profibus ein drahtgebundenes Kommunikationsprotokoll, das Profile für bestimmte Geräte oder Anwendungen definiert. Jedoch werden die meisten der enthaltenen Funktionalitäten vom Benutzer möglicherweise nicht benötigt. Um dieses Problem zu vermeiden, wurde entschieden, jede Implementierung von Kommunikationsprotokollen in ein eigenes, separates Modul zu gliedern, das *Technology Adapter* genannt wird. Dieser Adapter kann zur Laufzeit angepasst werden und ist dafür verantwortlich, eingehende Sensordaten in das interne Informations-Model zu übersetzen und Ereignisse zu versenden, wenn ein Sensor eine Verbindung herstellt, einen neuen Wert empfängt oder die Verbindung trennt, sowie eingehende Lese-/Schreib- oder Ausführungsbefehle von den *Data Consumer Adapters* zu verarbeiten.

Als nächstes soll das Hauptmodul ***STAG Core*** vorgestellt werden – Herz und Seele des Projekts. Dieses Modul definiert alle Teile, die von den *Technology Adapters* und *Data Consumer Adapters* gleichermaßen verwendet werden. Dazu gehören:

1. ***Information Model***: definiert Abstraktionen, die zur Modellierung von Senor-/Aktor-Systemen verwendet werden, die von den *Technology Adapters* instanziiert werden, und wird von *Data Consumer Adapters* verwendet.
2. ***Information Model Manager***: verwaltet den internen Zustand des *Information Model* sowie die *Technology Adapters* und *Data Consumer Adapters*.

3. ***Technology Adapter Interface***: definiert eine gemeinsame Schnittstelle für alle *Technology Adapters*.
4. ***Data Consumer Adapter Interface***: definiert eine gemeinsame Schnittstelle für alle *Data Consumer Adapters*.

Um diesen Abschnitt nicht zu sehr zu verkomplizieren, wird das Hauptmodul *STAG Core* im nachfolgenden Abschnitt 8.4.3 noch etwas ausführlicher behandelt.

Im Gegensatz zu ihren Gegenstücken, den *Technology Adapters*, unterscheiden wir beim Modul ***Data Consumer Adapters*** nicht auf Basis des Kommunikationsprotokolls, sondern nach ihrem Zweck. Das liegt daran, dass diese Module – aus Sicht der Autoren – Anwendungen sind, die Daten verbrauchen, daher der Name – *Data Consumer Adapters*. Diese Module verarbeiten Daten auf unterschiedliche Weise und es ist nicht ungewöhnlich, dass sie sich ein Kommunikationsprotokoll untereinander teilen. Zum Beispiel gibt es einen Datenbank-Adapter und einen IIoT-Plattform-Adapter. Beide verwenden MQTT als Kommunikationsprotokoll, aber die Art und Weise, wie sie die Daten verwenden, ist unterschiedlich. Der Datenbank Adapter speichert einfach alle Ereignisbenachrichtigungen in einem Klartextformat zur Fehlersuche und interagiert nicht mit den Abstraktionen. Der *IIoT-Platform Adapter* hingegen wandelt diese Abstraktionen in ein Informationsmodell um, das von der Plattform selbst verwendet wird.

Zusammenfassend lässt sich sagen, dass das STAG-Projekt in drei verschiedene Hauptmodule aufgeteilt wurde:

- ***Technology Adapters***, die das von den Sensoren/Aktoren verwendete Kommunikationsprotokoll handhaben und Abstraktionen instanziieren, die von späteren Modulen verwendet werden können.
- ***STAG Core***, der die Konstruktion dieser Abstraktionen erleichtert und als Middleware zwischen den *Technology Adapters* und den *Data Consumer Adapters* fungiert.
- Und schließlich ***Data Consumer Adapters***, die die Daten der Abstraktionen verwendet.

7.3.2 Architekturübersicht

Das Diagramm in Bild 7.1 stellt nochmals die drei Hauptbestandteile von STAG dar:

- ***Technology Adapters*** *(1)* – eingefärbt in Dunkel- und Hellblau;
- ***STAG Core*** *(2)* – eingefärbt in Grüntönen;
- ***Data Consumer Adapters*** *(3)* – eingefärbt in Rosa und Lila.

Die beiden Hauptmodule *Technology Adapters* als auch *Data Consumer Adapters* wurden bereits im vorherigen Abschnitt ausführlich behandelt. Deshalb konzentriert sich dieser Abschnitt ausschließlich auf *STAG Core*.

STAG Core *(2)* hat vier Unterkomponenten. Im oberen Teil findet sich die Unterkomponente ***Information Model*** *(2.1)*. Diese Unterkomponente ist die Grundlage für die Erstellung von digitalen Abbildern der Geräte, den sogenannten *Device Abstraction*, die im folgenden Abschnitt 8.4.4. noch genauer betrachtet werden.

Auf der linken Seite findet sich die Unterkomponente ***Technology Adapter Interface*** *(2.2)*. Diese folgt dem Entwurfsmuster Fassade und bietet eine gemeinsame Schnittstelle für Adapter zur Erstellung der jeweiligen *Device Abstraction* an, indem sie das *Device Abstraction Builder Interface* verwendet. Zusätzlich ermöglicht die Implementierung des *Event Model Interface* dem *Technology Adapter*, Benachrichtigungen an den verwalteten *Data Consumer Adapter* über den internen Event-Mechanismus zu senden.

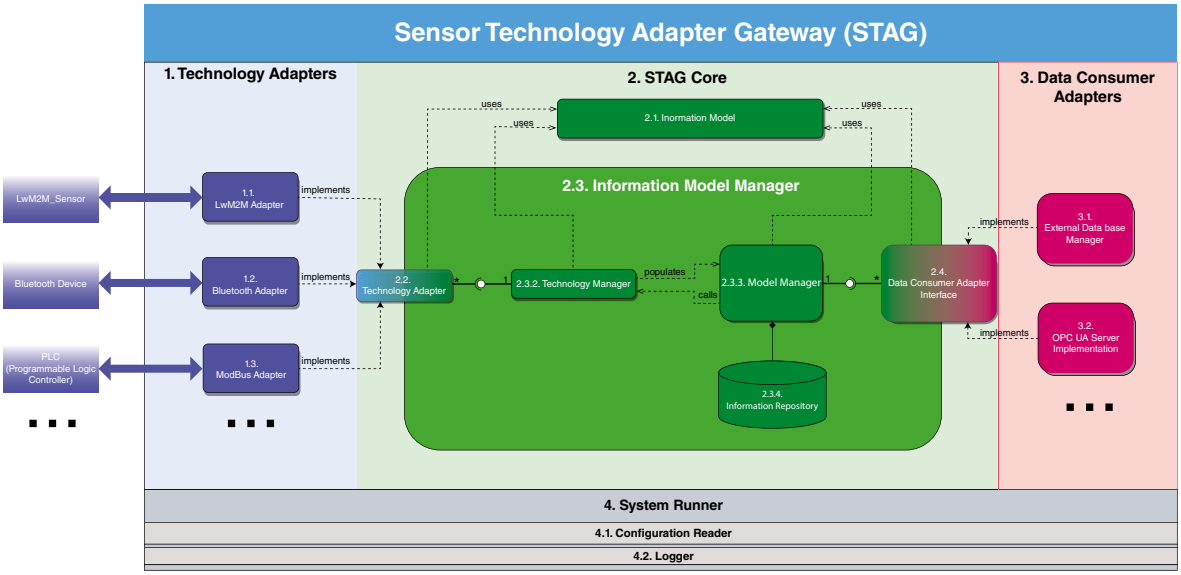

Bild 7.1 *STAG Component Diagram*

Die Unterkomponente *Technology Adapter Interface* ist mit dem ***Information Model Manager*** *(2.3)* verbunden, der wiederum drei Subkomponenten hat, die jeweils eine andere Funktion abdecken. Das *Technology Adapter Interface* ist mit einem Teil des *Information Model Manager* verbunden, der ***Technology Manager*** *(2.3.2)* genannt wird. Er steuert das Starten und das Stoppen einzelner Adapter und verbindet den *Device Abstraction Builder* mit der Implementierung des *Event Model* innerhalb des *Model Manager*. Der ***Model Manager*** (2.3.3) ist für die Pflege der *Device Abstraction* innerhalb des ***Information Repository*** *(2.3.4)* verantwortlich. Dieser Teil verwaltet, wie bereits erwähnt, *Device Abstraction Builder* und *Event Model*-Implementierungen, die von *Technology Adapters* und *Data Consumer Adapters* verwendet werden. Abschließend finden wir auf der rechten Seite des *Information Model Manager* die Unterkomponente ***Data Consumer Adapter Interface*** *(2.4)*, die einen Mechanismus für die *Data Consumer Adapters* bereitstellt, um auf Ereignisse der *Device Abstraction* über das interne *Event Model* zu hören, sowie auf die *Device Abstraction* selbst zuzugreifen.

In Bild 7.1 ist unten eine übergreifende Basiskomponente für alle Komponenten dargestellt. Diese Komponente wird ***System Runner*** (4) genannt. Sie ist dafür verantwortlich die Konfigurationen der *Technology Adapters* zu lesen und einen gemeinsamen Logging-Mechanismus für jeden Teil des Systems bereitzustellen. Daneben verwaltet *System Runner* die Threads jeder Systemkomponente.

Nach einer genaueren Betrachtung des *STAG Core* Modules wurde aufgezeigt, dass dieses das *Information Model* beinhaltet, das von *Technology Adapters* und *Data Consumer Adapters* gleichermaßen verwendet wird. Außerdem enthält es einen gemeinsamen Mechanismus zur Erstellung von *Device Abstraction* und ein internes *Event Model*, um *Data Consumer Adapters* über Ereignisse der *Device Abstraction* zu informieren. Darüber hinaus wurde festgestellt, dass der genaue Einstiegspunkt sowohl für *Technology Adapters* sowie *Data Consumer Adapters*, der *Information Modal Manager* ist. Die *Technology Adapters* verwenden den *Technology Manager* als ihren Einstiegspunkt. Dagegen verwenden die *Data Consumer Adapters* den *Model Manager* als ihren Einstiegspunkt. Und um all diese Komponenten, Unterkomponenten und Teile in ein zusammenhängendes Ganzes zu bringen, wird ein *System Runner* verwendet.

7.3.3 Internes Informationsmodell

Nachdem erläutert wurde, wozu das STAG-Projekt dient, wie es funktioniert und aufgebaut ist, kann die Frage beantwortet werden: Was sind *Information Model* und *Device Abstraction*, die immer erwähnt, aber nie erklärt werden? Zunächst muss definiert werden, was der Begriff ***Information Model*** im Allgemeinen und im Kontext des STAG-Projektes bedeutet. Danach wird ein Klassendiagramm (Bild 7.2) des *STAG Information Model* betrachtet, das unter Verwendung der Notation der ***U****nified* ***M****odeling* ***L****anguage* (UML) erstellt wurde, und analysiert, welche Art von Informationen damit beschrieben werden können. Danach folgt ein generisches Beispiel der *Device Abstraction*, das in Bild 7.3 dargestellt wird.

In der objektorientierten Softwareentwicklung ist ein Informationsmodell eine Sammlung von Klassen, die verwendet werden, um etwas zu beschreiben. In STAG werden verschiedene Sensoren beschrieben, die etwas messen können, Aktoren, die eine Art von Aktionen ausführen können, oder manchmal eine Kombination aus beiden. Um dies zu tun, ist das *Information Model* in zwei verschiedene Bereiche aufgeteilt. Der erste Teil ist das, was wir als statische Information bezeichnen. Diese Information, die einen Sensor oder einen Aktor beschreiben, ändert sich nie. Sie definiert, wer dieses System gebaut hat, was es tut und wie es funktioniert. Der andere Teil unseres *Information Model* wird dynamische Information genannt. Diese Art von Information ändert sich ständig und stellt den aktuellen Zustand des Systems bzw. der erfassten Daten dar.

Um die statischen und dynamischen Informationen besser zu verstehen, soll ein einfaches Beispiel dienen. Ein Temperatursensor soll modelliert werden. Dieser Sensor misst die Temperatur an drei unterschiedlichen Punkten und berechnet den Mittelwert zwischen diesen Punkten. Die Tatsache, dass dieser Sensor die Temperatur misst und dass er dies an drei Punkten im Raum tut sowie deren Mittelwert berechnet, ist eine statische Information**.** Dies liegt daran, dass es keine Möglichkeit gibt das Verhalten und den Aufbau zu verändern, ohne den Sensor selbst zu verändern. Die Messwerte an den drei Punkten und der berechnete Mittelwert wären jedoch dynamische Informationen. Mit diesem Wissen betrachtet man nun das UML-Klassendiagramm des *STAG Information Model*, das in Bild 7.2 dargestellt ist.

Die Basis jeder einzelnen Klasse innerhalb des *STAG Information Model* ist die Klasse ***Named Element***. Diese Klasse definiert Eigenschaften wie den Namen des Elements, seine Identifikationsnummer und eine Beschreibung. Wie aus dem Diagramm ersichtlich wird, leiten sich nur zwei weitere Klassen direkt von ihr ab, nämlich die Klassen *Device Element* und *Device*. ***Device*** ist eine direkte Abstraktion eines Sensors oder eines Aktors als Ganzes und beschreibt die gesamte Funktionalität dieser Entität, während die Klasse ***Device Element*** einen bestimmten Teil dieser Entität beschreibt und ohne sie nicht existieren kann. Um auf das Beispiel des Temperatursensors zurückzukommen, würde *Device* einen Temperatursensor beschreiben und ein *Device Element* würde jeweils eines der Sensorelemente sowie die Funktionalität der Mittelwertberechnung beschreiben.

Es reicht jedoch nicht aus, nur die Klasse *Device Element* zu haben, um eine Entität zu beschreiben. Daher werden drei weitere Klassen zur Verfügung gestellt, die von ihr abgeleitet werden und helfen den Zweck des modellierten Teils besser zu verdeutlichen. Die erste Spezialisierung ist die Klasse ***Device Element Group***. Diese Klasse fasst andere *Device Element* Klassen zusammen. Im Beispiel des Temperatursensormodells wäre diese eine logische Gruppe, die die drei Temperaturpunkte als Unterelemente enthalten würde. Das Gruppieren von Teilen hilft, die modellierte Entität zu vereinfachen und dem Benutzer zu ermöglichen, ihre Fähigkeiten auf einen Blick zu verstehen. Da die *Device Element Group* vom Typ *Device Element* ist, kann man mehrere *Device Element Group* zusammenfassen, was eine mehrschichtige Modellierung von *Device Abstraction* ermöglicht.

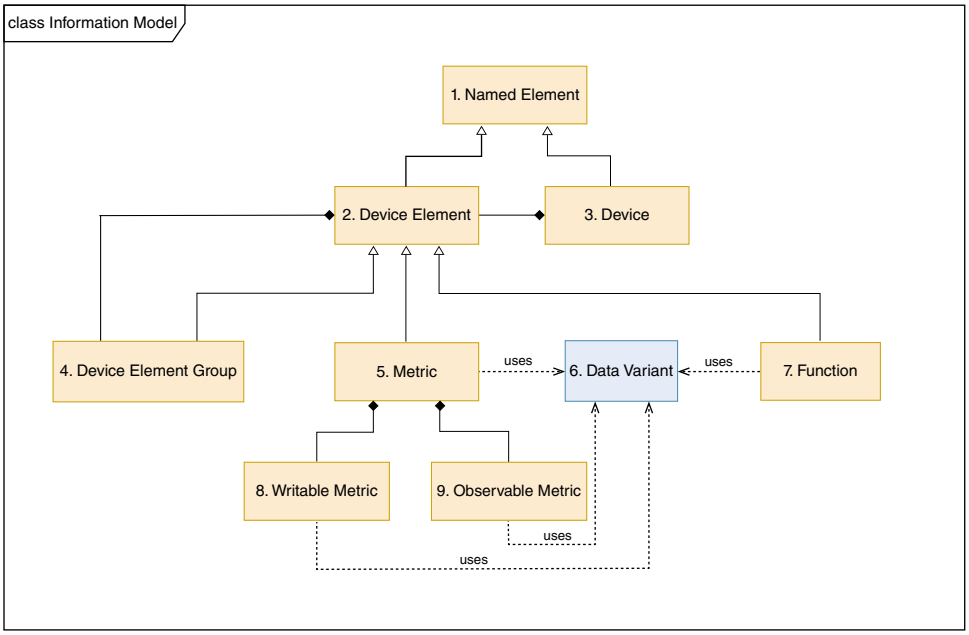

Bild 7.2 *STAG Information Model Class Diagram*

Alle zuvor genannten Klassen modellieren statische Informationen und bieten keine Interaktionsmöglichkeit mit den Funktionen der modellierten Entität. Um die dynamischen Informationen zu modellieren, verwendet man die Klassen *Metric* oder *Function*, die vom *Device Element* abgeleitet sind, und somit selbst über einige statische Informationen verfügen. Der Grund für zwei getrennte Klassentypen für die Modellierung dynamischer Informationen ist einfach: ***Metric*** modelliert den lesbaren Wert, z. B. die aktuelle Temperatur oder den Mittelwert aller drei Messwerte, während ***Function*** eine Aktion modelliert, die die Entität ausführen kann, z. B. die Fähigkeit, den Sensor neu zu starten oder ein Ventil zu öffnen. Außerdem besitzt eine *Function* keinen Rückgabewert, ist also nicht auslesbar, während eine *Metric* immer einen auslesbaren Wert hat.

Beide Klassen sind von dem Hilfstyp ***Data Variant*** abhängig. *Data Variant* selbst ist keine Klasse, sondern eine Sammlung von Datentypen wie eine Ganzzahl (*Integer*) oder eine Textzeichenfolge (*String*), daher ist sie in einer anderen Farbe als die Klassen markiert. Dieser Typ wird von allen Klassen verwendet, die dynamische Informationen modellieren, da es immer einen Datentyp gibt, der mit dem Wert der modellierten *Metric* oder einer *Function* verknüpft ist. Zum Beispiel würde der Temperatursensor Werte vom Typ Integer zurückgeben.

Die letzten beiden Klassen innerhalb des *STAG Information Model* sind Dekoratoren für die *Metric*-Klasse. Sie erweitern die Funktionalität um eine lesbare *Metric*. Diese Dekorator-Klassen sind die ***Writable Metric***, die es erlauben, den aktuellen Wert zu ändern, und die ***Observable Metric***, die angibt, dass diese *Metric* in der Lage ist, einen Wert periodisch oder eine Wertänderung zu melden. *Writable Metric* kann anstelle von *Function* verwendet werden, wenn ein Rückgabewert benötigt wird, oder als Label. *Observable Metric* reduziert den Leseaufwand für sich ändernde Werte, da diese nicht durch permanentes Auslesen aktuell gehalten werden müssen. Der Grund, warum diese beiden Klassen nicht direkt von *Metric* erben, sondern sie mit zusätzlicher Funktionalität ausstatten, ist, dass jeder einzelne *Metric*-Typ immer lesbar, aber nur manchmal schreibbar oder beobachtbar ist.

Alle zuvor genannten Klassen werden verwendet, um eine ***Device Abstraction*** zu erstellen, die den Sensor oder Aktor im digitalen Raum modelliert. Um besser zu verstehen, wie diese aussieht, dient Bild 7.3. In diesem Diagramm sieht man, dass das *Device* einer Zwiebel ähnelt, die aus einer *Device Element Group* besteht, die selbst wieder weitere *Metric*-, *Function*- oder sogar andere *Device Element Group*-Elemente Entitäten enthält. Wie bereits erwähnt, erlaubt dieser Verschachtelungsansatz Teile einer Entität logisch zu strukturieren und komplexere Teile in dedizierten *Device Element Group* zusammenzufassen.

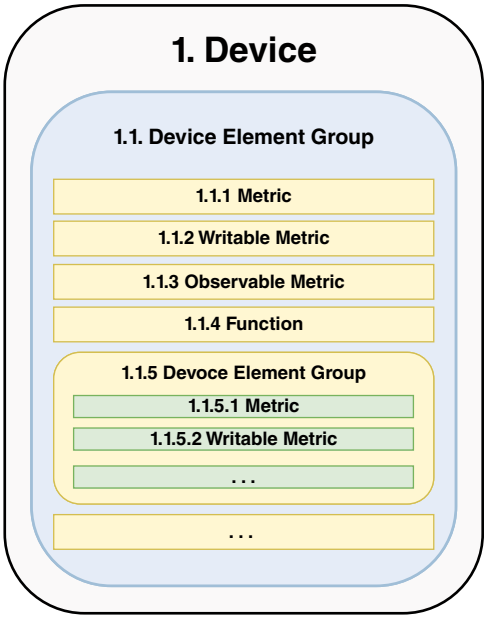

Bild 7.3 *STAG Device Abstraction*

Zusammengefasst wurden *STAG Information Model* und *Device Abstraction* erläutert, was diese bedeuten und wie sie funktionieren. Außerdem haben wurde gezeigt, dass das *STAG Information Model* Entity-Informationen sich in zwei Kategorien unterteilt, nämlich in statische und dynamische Informationen. Abschließend wurde kurz aufgezeigt, wie eine generische *Device Abstraction* dargestellt werden könnte.

7.3.4 Automatisches Mapping auf das interne Informationsmodell

Jedes Informationsmodell ist nur so gut wie die Person, die es zur Modellierung von Entitäten verwendet, und es ist sehr einfach, beim Erstellen einer *Device Abstraction* Fehler zu machen. Um die meisten der typischen Fallstricke während dieses Prozesses zu vermeiden, wird ein ***Device Builder Interface*** zur Verfügung gestellt, das durch eine *Technology Adapters*-Instanz beim Erstellen einer neuen *Device Abstraction* immer verwendet werden sollte. Diese Schnittstelle ist im *STAG-Core* implementiert und wird vom *Technology Manager* zur Verfügung gestellt, sobald er im System initialisiert wird. Wie der Name *Device Builder Interface* schon andeutet, ist die Schnittstelle basierend auf dem *Builder-Pattern* implementiert worden. Auf die konkrete Implementierung gehen wird im nächsten Abschnitt genauer eingegangen.

Die Basis der *Device Abstraction*, die *Device*-Klasse aus dem *STAG Information Model*, wird beim Aufruf des *Device Builder Interface*-Konstruktors erzeugt. Dieser Aufruf weist der *Device Abstraction* die minimalen statischen Informationen zu und beinhaltet die Informationen für die interne ID-Generierung. Nach Beendigung des Konstruktors gibt es einige Methoden, die es ermöglichen, die *Device*-Klasse zu erweitern, indem man verschiedene *Device Element* Instanzen bereitstellt, wie z. B. *Device Element Group*, *Function* oder *Metric*. Alle diese Methoden beginnen mit `std::string add*(*)`, z. B.:

```
std::string addDeviceElementGroup(const std::string &name,
                                  const std::string &desc)
```

Diese Methode fügt eine neue *Device Element Group* Instanz zum *Device* hinzu. Jeder Methodenaufruf sorgt für die entsprechende Verknüpfung zwischen *Device Element* und *Device* oder dessen übergeordneter *Device Element Group* und weist automatisch die interne *Device Element*-ID zu, die dann an den Aufrufer zurückgegeben wird. Bei *Device Element*, das dynamische Informationen modelliert, weisen diese Methode auch dem *Data Variant-Container* und der modellierten Funktionalität, in Form von Funktions-Call-Backs, den passenden Wert zu, z. B.:

```
std::string addWritableMetric(const std::string &group_refid,
                              const std::string &name,
                              const std::string &desc,
                              DataType data_type,
                              ReadFunctor read_cb,
                              WriteFunctor write_cb)
```

Im obigen Beispiel würde *DataType* verwendet werden, um den Datentyp der *Data Variant* zuzuweisen und *ReadFunctor* und *WriteFunctor* würden die Funktionsrückrufe festlegen.

Neben all den spezialisierten Methoden bietet das *Device Builder Interface* auch einen generischen Ansatz, der das Hinzufügen jeder Art von *Device Element* und die manuelle Angabe seines Typs ermöglicht, wodurch erfahrene Entwickler ihre *Device Abstraction* feiner anpassen oder den *Device Element* Typ programmatisch auflösen können. Diese Methode wird auch intern von jeder anderen `std::string add*(*)` Methode verwendet. Daher kann diese Methode sicher verwendet werden, um gültige *Device Element* Instanzen zu erstellen.

```
std::string addDeviceElement(const std::string &group_refid,
                             const std::string &name,
                             const std::string &desc,
                             ElementType type,
                             DataType data_type,
                             std::optional<ReadFunctor> read_cb,
                             std::optional<WriteFunctor> write_cb)
```

Sobald alle gewünschten *Device Element*-Instanzen zur *Device Abstraction* hinzugefügt wurden, wird eine `getResult()` Methode aufgerufen, um eine vollfunktionsfähige Instanz der *Device*-Klasse zurückzugeben, die dann vollständig im Besitz des Aufrufers ist. Der Destruktor des *Device Builder Interface* kann ebenfalls aufgerufen werden. Es ist beabsichtigt, die gesamte Lebensdauer einer *Device Builder Interface*-Instanz für die Erstellung eines einzigen *Device* zu

verwenden. Auf diese Weise stellen wir sicher, dass bei der Erstellung einer neuen *Device Abstraction* keine vorherigen Informationen der *Device Abstraction* verbleiben, wodurch ein sauberer Ansatz ermöglicht wird.

Zum Abschluss wurde erklärt, dass es mit dem *Device Builder Interface* eine Schnittstelle gibt, die immer verwendet werden sollte, wenn eine *Device Abstraction* erstellt wird. Es wurde gezeigt, welche Methoden sie enthält und wie sie verwendet werden können. Weiterhin wurde erläutert, dass eine einzelne Instanz des *Device Builder Interface* nur eine einzige *Device Abstraction* erzeugt und dass sie ausschließlich für die *Technology Adapters* zugänglich ist.

7.4 Umsetzung und Zielerreichung

Nachdem der theoretische Aufbau von STAG erörtert wurde, kann nun der aktuelle Stand des Projektes betrachtet werden. Dieser Abschnitt konzentriert sich darauf, welche Adapter implementiert wurden, wie das Projekt installiert werden kann und welche zukünftigen Erweiterungen geplant sind.

Projektzustand

Im vorangegangenen Abschnitt wurde sehr ausführlich über die STAG-Architektur und ihre Komponenten gesprochen, aber es wurde weder die Implementierung des *Technology Adapter* noch des *Data Consumer Adapter* erklärt.

Um den gesamten Projektstatus besser zu verstehen, veranschaulicht Bild 7.4 die aktuelle Implementierungsübersicht. Diese Abbildung ähnelt dem im vorigen Abschnitt gezeigten STAG-Komponentendiagramm, zeigt jedoch die tatsächlich implementierten Module und ihre Gruppierung und Zuordnung.

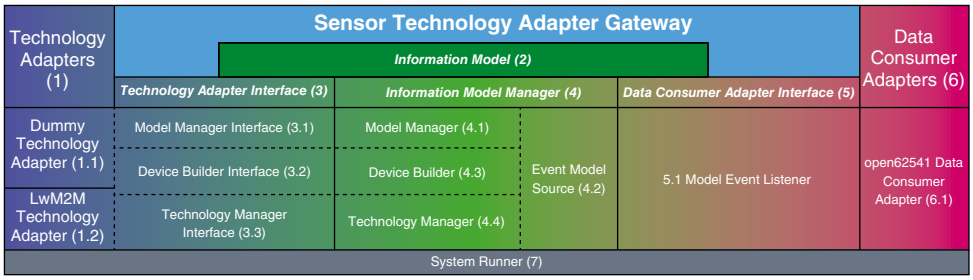

Bild 7.4 *Implementierungsübersicht*

Wie in Bild 7.4 zu sehen ist, verfügt das Projekt derzeit über zwei Implementierungen von ***Technology Adapters*** (1) und eine Implementierung eines ***Data Consumer Adapters*** (6). Die erste *Technology Adapter* Implementierung heißt ***Dummy Technology Adapter*** (1.1). Dieses Modul dient als Beispiel und hilft den Benutzern die Funktionsweise besser zu verstehen und bietet die Möglichkeit erste Erfahrungen mit dem internen *Information Model* zu sammeln. Der *Dummy Technology Adapter* ist ein sehr vereinfachtes Modul und dient maßgeblich als Beispiel und ist nicht für den produktiven Einsatz gedacht.

Der ***LwM2M Technology Adapter*** (1.2) ist die zweite Implementierung. Er dient der Einbindung von Sensoren, die basierend auf dem LwM2M Standard kommunizieren. Innerhalb des Moduls wird eine eigene Implementierung eines LwM2M Servers verwendet, der eine Konver-

tierung des LwM2M Gerätemodells in das interne Informationsmodell übernimmt. Es wurde entschieden, eine eigene Implementierung für den LwM2M Server zu erstellen, da es zu diesem Zeitpunkt keine gute Open-Source-Lösung gab, die das LwM2M Protokoll in der gewählten Programmiersprache C++ implementiert.

Der ***open62541 Data Consumer Adapter*** (6.1) ist eine Implementierung und damit ein Beispiel des *Technology Adapters*. Dieses Modul verwendet Daten aus dem internen *Information Model* und wandelt dies in ein OPC UA Informationsmodell um. Dieses Modell wird dann in der OPC UA-Server-Implementierung basierend auf der Open-Source-Lösung open62541 gespeichert und bereitgestellt. Die Wahl fiel auf die open62541 OPC UA-Implementierung, weil sie quelloffen, vollständig konform mit dem OPC UA-Standard ist und zusätzlich auch noch portabel (Windows/Linux) und einfach zu integrieren ist.

Der letzte Teil im Zusammenhang mit dem Projektstatus, ist die Installation und Inbetriebnahme. Ein System kann die beste Lösung der Welt sein, wenn es jedoch schwierig zu beschaffen und zu installieren ist, wird die Akzeptanz stark leiden. Deshalb wurden vorgefertigte Installations-Pakete für verschiedene Systemdistributionen entwickelt. Diese Pakete enthalten bereits alle eingebauten Module, Konfigurationsdateien und erforderlichen Lizenzen. Dadurch ist ein leichtes Herunterladen und Entpacken am gewünschten Installationsort möglich. STAG muss dann nur noch per Kommandozeilenbefehl gestartet werden oder kann auch automatisch als Dienst gestartet werden. Die Installationspakete wie auch die zugehörige Dokumentation ist auf der Projektwebsite (http://stag.hahn-schickard.de) zu finden.

Zukunftsplanung

Zum Abschluss dieses Kapitels soll einen Ausblick auf die Zukunftspläne des STAG-Projektes gegeben werden. Gegenwärtig wird die Implementierung einer dynamischen Abhängigkeitsbehandlung sowohl für ***Technology Adapters*** als auch für ***Data Consumer Adapters*** geplant. Dies hat zur Folge, dass sogar zur Laufzeit unterschiedliche Technologien aktiviert oder auch deaktiviert werden können. Außerdem sollen neue ***Data Consumer Adapter***, z. B. MQTT-Provider, sowie neue ***Technology Adapters***, z. B. das Bluetooth-LE-GATT-Profil, hinzugefügt werden und somit die Anwendungsbereiche von STAG auch über die Anwendung Industrie 4.0 bis in das Internet der Dinge erweitert werden.