Ostbayerische Technische Hochschule Amberg-Weiden
Fakultät Elektrotechnik, Medien und Informatik

Studiengang Medieninformatik

**Bachelorarbeit**

von

Stephan **Prettner**

# WebGL Anwendungen in Unity: Kommunikation mit OPC UA

WebGL Applications in Unity:
Communication with OPC UA

Ostbayerische Technische Hochschule Amberg-Weiden
Fakultät Elektrotechnik, Medien und Informatik

Studiengang Medieninformatik

**Bachelorarbeit**

von

Stephan **Prettner**

# WebGL Anwendungen in Unity: Kommunikation mit OPC UA

WebGL Applications in Unity: Communication with OPC UA

Bearbeitungszeitraum: von 4. Dezember 2020
bis 3. Mai 2021

1. Prüfer: Prof. Dr. Dieter Meiller

2. Prüfer: Veit Stephan M.Eng.

Ostbayerische Technische Hochschule Amberg-Weiden
Fakultät Elektrotechnik, Medien und Informatik

Ostbayerische Technische Hochschule
Amberg-Weiden

Bestätigung gemäß § 12 APO

---

Name und Vorname
der Studentin/des Studenten: **Prettner, Stephan**

Studiengang: **Medieninformatik**

---

Ich bestätige, dass ich die Bachelorarbeit mit dem Titel:

**WebGL Anwendungen in Unity:**
**Kommunikation mit OPC UA**

selbständig verfasst, noch nicht anderweitig für Prüfungszwecke vorgelegt, keine anderen als die angegebenen Quellen oder Hilfsmittel benützt sowie wörtliche und sinngemäße Zitate als solche gekennzeichnet habe.

---

Datum: 22. März 2021

Unterschrift:

---

Ostbayerische Technische Hochschule Amberg-Weiden
Fakultät Elektrotechnik, Medien und Informatik


Bachelorarbeit Zusammenfassung

---

Studentin/Student (Name, Vorname): **Prettner, Stephan**
Studiengang: Medieninformatik
Aufgabensteller, Professor: Prof. Dr. Dieter Meiller
Ausgabedatum: 4. Dezember 2020
Abgabedatum: 22. März 2021

---

Titel:

**WebGL Anwendungen in Unity:**
**Kommunikation mit OPC UA**

---

Zusammenfassung:

Durch die Erforschung von Anwendungen, die zum Beispiel eine 3D-Darstellung von digitalen Zwillingen ermöglichen, soll eine optimale Abbildung der physischen Maschinen gewährleistet werden können. Mit der Entwicklungsumgebung Unity ist es möglich, solche Anwendungen für verschiedene Zielplattformen zu erstellen. Deshalb soll in dieser Arbeit beantwortet werden, wie es einer in Unity entwickelten WebGL Anwendung möglich ist, mit einem OPC UA Server kommunizieren zu können.

Schlüsselwörter: OPC UA, WebGL, Unity, WebSockets, WebAssembly

# Inhaltsverzeichnis

# Abkürzungs- und Begriffsverzeichnis

| | |
|---|---|
| **.obj** | Dateiformat zum plattformunabhängigen Austausch von dreidimensionalen geometrischen Formen, das von vielen 3D-Programmen unterstützt wird. |
| **.blend** | Dateiformat, das zusätzlich zu dreidimensionalen geometrischen Formen weitere Daten, wie zum Beispiel Texturen, Beleuchtung und Animationen speichert. Wird mithilfe von Blender erstellt und kann in einigen anderen Programmen verwendet werden. |
| **.fbx** | Dateiformat des Unternehmens Autodesk zur Speicherung von dreidimensionalen geometrischen Formen, dem zusätzliche Daten wie Texturen, Beleuchtung und Animationen angehängt werden können. |
| **Asset** | Ein Asset bezieht sich innerhalb dieser Abschlussarbeit auf ein in Unity verwendbares Objekt. Dieses kann aus externen Dateien, wie zum Beispiel einer Audio-Datei oder eines Bildes, stammen und importiert werden. |
| **Binaryen** | Tool zur Konvertierung und Optimierung von WebAssembly-Quellcode. |
| **Blender** | Programm, mit dem 3D-Modelle bearbeitet, texturiert und modelliert werden können. Die von der Blender Foundation entwickelte frei verfügbare Software bietet mithilfe von Plugins zusätzliche Funktionalitäten. |
| **Bytecode** | Mithilfe eines Bytecodes kann plattformunabhängiger Quellcode erstellt werden, da dieser erst auf dem Hostsystem interpretiert wird. Bytecode wird zum Beispiel für Python generiert. |
| **C** | Programmiersprache, in der die meisten grundlegenden Programme für Betriebssysteme entwickelt worden sind. Ist 1972 von Dennies Ritchie vorgestellt worden. |
| **C#** | Objektorientierte Programmiersprache, die von Microsoft entwickelt wurde und 2001 erschienen ist. |
| **C++** | Programmiersprache, die als Erweiterung für C entwickelt worden ist. |

| | |
|---|---|
| **CAD** | Computer-aided design. Mit einem CAD-Modell kann das manuelle Zeichnen eines Modells durch eine digitale Variante ersetzt werden. Dabei wird mithilfe einer CAD-Software eine dreidimensionale Abbildung konstruiert. |
| **Callback** | Funktion, die einer anderen Funktion als Parameter übergeben wird. Dadurch kann diese beispielsweise beim Eintreten eines Ereignisses aufgerufen werden. |
| **China 2025** | Strategischer Plan zur Verbesserung der Fertigungskapazitäten in China, der an Industrie 4.0 angelehnt ist. |
| **DOM** | Document Object Model. Bildet die Struktur eines Dokumentes, das in einem Browser angezeigt werden kann, ab. |
| **Emscripten** | Tool zur Konvertierung von Quellcode verschiedener Programmiersprachen, damit diese im Browser ausgeführt werden können. |
| **GameObject** | Ein Objekt innerhalb einer Unity-Szene. |
| **Hostsystem** | System, auf dem das Programm ausgeführt wird. |
| **HTTP** | Hypertext Transfer Protocol. Protokoll, das definiert, wie eine Webseite geladen wird. |
| **IL2CPP** | Intermediate Language To C++. Tool des Unternehmens Unity, das zur Konvertierung von Skripts nach C++ verwendet wird. |
| **Industrie 4.0** | Zukunftsprojekt der Bundesregierung Deutschland, mit dem die industrielle Produktion digitalisiert werden soll. Dadurch soll unter anderem eine flexible Produktion und optimierte Logistik durch den Einsatz gesammelter Daten ermöglicht werden. |
| **Internet** | Weltweites Netzwerk zur Kommunikation zwischen Computern. |
| **Intranet** | Lokales Netzwerk zur Kommunikation zwischen Computern. Dieses Rechnernetz kann dabei unabhängig vom Internet existieren und eingeschränkte Funktionen bieten. |
| **JavaScript** | Skriptsprache zur Interaktion und Manipulation von Webseiten. |
| **JSON** | JavaScript Object Notation. Datenformat in für den Menschen lesbarer Form. |
| **Persistenz** | Eigenschaft eines Objektes, aktuelle Informationen über längere Zeit aufrechterhalten zu können. Es muss jederzeit möglich sein, den aktuellen Status rekonstruieren zu können. |
| **PHP** | Skriptsprache zur Bereitstellung dynamischer Webseiten. |
| **Polling** | Ermittlung eines Status mithilfe zyklischer Abfragen. |

| | |
|---|---|
| **Proof of Concept** | Demonstration einer Funktionalität komplexer Anwendungen. Im Rahmen des Proof of Concept wird oft ein Prototyp erstellt, um die Machbarkeit zeigen und die Lösung evaluieren zu können. |
| **Proxy** | Komponente eines Systems, die als Schnittstelle zwischen weiteren Komponenten vermittelt. |
| **Python** | Objektorientierte interpretierte Programmiersprache, die mit dem Ziel einer einfachen Übersichtlichkeit entworfen wurde. |
| **Raspberry Pi** | Kleinrechner, der für eine Vielzahl von Anwendungen eingesetzt werden kann. |
| **REST** | Representational State Transfer. Designprinzip, mit dem es möglich ist, flexibel eine zustandslose Schnittstelle für einen Server zu entwickeln. |
| **Schnittstelle** | Die Berührungspunkte mehrerer Komponenten werden als Schnittstellen bezeichnet. Die Schnittstellen beschreiben den Austausch von Daten an diesen Punkten. |
| **Session** | Bezeichnet eine existierende Verbindung eines Clients mit einem Server. |
| **Skript** | Liste von Befehlen, die von einem Computer ausgeführt werden. |
| **Transpilierung** | Übersetzt eine Programmiersprache in eine andere Programmiersprache mit einer ähnlichen Abstraktionsstufe. |
| **URL** | Uniform Resource Locator. Ermöglicht die eindeutige Identifizierung einer Ressource im Internet. |
| **W3C** | World Wide Web Consortium. Konsortium zur Entwicklung von Standards für das Internet. |

# Abbildungsverzeichnis



# Tabellenverzeichnis

# Listings

# Kapitel 1

# Einleitung

## 1.1 Thematische Einführung

Eine weitere Umsetzung des Zukunftsprojektes Industrie 4.0 war bereits in den letzten Jahren im Trend und das Interesse an dieser Umsetzung steigt von Jahr zu Jahr. Unter anderem fördert aufgrund dessen auch die Bundesregierung Deutschland dieses Projekt. Laut dem Bundesministerium für Bildung und Forschung (2020) gehe es dabei vor allem darum, dass die deutsche Industrie gut für die Zukunft der Produktion gerüstet sei. Dies kennzeichne eine starke Individualisierung der Produkte unter den Bedingungen einer hoch flexibilisierten Produktion.

Es ist hierbei wichtig, dass zu jedem Zeitpunkt ein leicht zugänglicher Überblick über die dezentral arbeitenden Maschinen mit dem jeweiligen aktuellen Status existiert. Durch den digitalen Überblick auf die Maschinen soll der aktuelle Status leichter eingesehen und Entscheidungen dadurch effizienter getroffen werden können. Die Entwicklung von Anwendungen, die zu genau diesem Zweck entworfen werden, ist eine direkte Schlussfolgerung daraus.

## 1.2 Forschungsmotivation

Maschinen können mit anderen Maschinen kommunizieren. Dabei wird in der Industrie 4.0 für den Datenaustausch zwischen mehreren Maschinen meist das Protokoll Open Platform Communications Unified Architecture (OPC UA) empfohlen und verwendet (Bundesministerium für Wirtschaft und Energie, 2018).

Um Datensätze beziehungsweise den aktuellen Status einer Maschine für den Menschen visualisieren zu können, ist es notwendig, diese lesbar innerhalb einer Anwendung aufzubereiten. Mit diesen Informationen kann zu jeder Zeit der aktuelle Stand des Systems überwacht werden.

Durch die Erforschung von Anwendungen, die zum Beispiel eine 3D-Darstellung von digitalen Zwillingen ermöglichen, soll eine optimale Abbildung der physischen Maschinen gewährleistet werden können. Anwendungen sollen dabei weltweit verfügbar

sein, solange eine Anbindung an das Internet beziehungsweise Intranet vorhanden ist. Die Motivation zu dieser Arbeit besteht darin, den Entwicklern solcher Anwendungen Lösungen und Wege aufzuzeigen, wie sie bei der Durchführung ihrer Projekte vorgehen könnten und welche Herausforderungen es dabei zu überwinden gilt.

## 1.3 Ziel der Arbeit

Mit der Entwicklungsumgebung Unity ist es möglich, Anwendungen für verschiedene Zielplattformen zu erstellen. Neben PCs werden Spielkonsolen, mobile Geräte sowie Anwendungen für den Webbrowser unterstützt. Mit Unity können Anwendungen als WebGL Anwendung exportiert werden. Grundsätzlich besteht bei der Entwicklung einer solchen Anwendung die Möglichkeit, mit anderen Systemen über das sogenannte WebSocket-Protokoll zu kommunizieren. Die direkte Kommunikation mit einem OPC UA Server ist allerdings nicht möglich.

Bei der Entwicklung mit Unity kann in der Entwicklungsumgebung jederzeit das aktuell entwickelte Programm in einer Vorschau angezeigt werden. Der Export des aktuellen Programmcodes als WebGL Anwendung dauert im Vergleich zum Start der Vorschau jedoch erheblich länger. Bei jeder Änderung des Programmcodes muss, um das Ergebnis der WebGL Anwendung sehen zu können, der Export neu durchgeführt werden.

Daraus lassen sich die Fragestellungen, die in dieser Arbeit behandelt werden, ableiten:

- Wie ist es einer WebGL Anwendung möglich mit einem OPC UA Server zu kommunizieren?
- Wie kann man auf die aktuellen Daten eines OPC UA Servers im Entwicklungsprozess von Unity zugreifen?

## 1.4 Methodisches Vorgehen

Die Fragestellungen sollen mithilfe eines Proof of Concept beantwortet werden. Der Aufbau der Arbeit besteht daraus, dass zu Beginn der Arbeit eine Anforderungsanalyse durchgeführt werden soll. Innerhalb dieser Anforderungsanalyse wird die derzeitige Ist-Situation erfasst, indem Spezifikationen und eventuell derzeitig bestehende Lösungen verglichen werden. Anschließend werden Anforderungen an einen zu entwickelnden Proof of Concept definiert. Zu diesem Proof of Concept wird anschließend ein Konzept zur weiteren Vorgehensweise erarbeitet. In der Implementierungsphase werden die einzelnen Teilsysteme umgesetzt. Mithilfe der Evaluation werden die Anforderungen, die an den Proof of Concept gestellt worden waren, ausgewertet, um schlussendlich ein Fazit ziehen zu können.

# Kapitel 2

# Grundlagen

In diesem Kapitel werden die für diese Abschlussarbeit wichtigsten Technologien und Protokolle erläutert. Für die Kommunikation zwischen einer WebGL Anwendung mit einem OPC UA Server sind dabei einige theoretische Grundlagen zu beachten. Dabei wird zuerst auf die Grundlagen der Implementierung einer WebGL Anwendung eingegangen, um anschließend die tatsächliche Anbindung einer Maschine, die an einen OPC UA Server angeschlossen ist, genauer zu beleuchten. Zu Beginn wird deshalb im *Abschnitt 2.1 HTML5 und WebGL* der grundlegende Aufbau von HTML, sowie ein simples Beispiel einer WebGL Anwendung beschrieben. Anschließend wird im nächsten Abschnitt die Entwicklung einer WebGL Anwendung mit der IDE 2.2 demonstriert. Nachfolgend wird im *Abschnitt 2.3 OPC UA* ein kleiner Überblick über das Protokoll vermittelt. Zum Schluss dieses Kapitels werden in den Abschnitten 2.4 WebSockets und 2.5 WebAssembly vorgestellt.

## 2.1 HTML5 und WebGL

Heutzutage verwenden die meisten Personen regelmäßig verschiedenste Webseiten im Internet. Sei es an einem Smartphone, einem Notebook oder einem anderen Gerät mit Internetzugang. Verschiedenste Firmen erwirtschaften mit dem Anbieten eines oder mehrerer Dienste in dieser Branche ihren Gewinn. Um diese Anwendungen auf den verschiedenen Geräten effizient anzeigen zu können, wird die Hypertext Markup Language (HTML) verwendet. In diesem Abschnitt wird ein kurzer Überblick über diese Technologie gegeben und wie diese mit der Web Graphics Library (WebGL) in Verbindung steht.

Um die verschiedenen Bestandteile einer Webseite übertragen zu können, musste ein Standard geschaffen werden. Die verschiedenen Strukturen werden hier einheitlich kodiert, damit nach der Übertragung über das Internet eine einheitliche Visualisierung in den verschiedenen Browsern stattfinden kann. Im Jahr 1989 wurde dabei laut Bühler, Schlaich und Sinner (2017) von Tim Burners-Lee die erste Version von HTML veröffentlicht. Der Aufbau einer Struktur besteht dabei aus verschiedenen HTML-Elementen, sogenannten Tags, die diese einzelnen Komponenten dabei genauer beschreiben. Somit

können zum Beispiel Überschriften, Verlinkungen zu anderen Webseiten und andere Elemente spezifiziert werden. Bei HTML5 handelt es sich grundsätzlich um die fünfte Fassung des HTML Standards. Die HTML5-Spezifikation wurde am 28. Oktober 2014 dem World Wide Web Consortium (W3C) vorgelegt und enthält eine Fülle an neuen Funktionen, die eine Unterstützung von Video, Audio, 2D- und 3D-Grafiken enthält. In der vierten Fassung von HTML waren diese Funktionen nur mithilfe zusätzlicher Erweiterungen verfügbar.

Eine der für diese Abschlussarbeit relevantesten Neuerungen bei HTML5 ist das Element Canvas. Canvas bedeutet übersetzt „Leinwand“ und bietet die Möglichkeit mithilfe von JavaScript Grafiken innerhalb dieses Elements zu erzeugen. Da Canvas mittlerweile von allen Browsern unterstützt wird, ist dieses nach Bühler et al. (2017) bedenkenlos einsetzbar. Ein großer Vorteil dieses Elementes ist es, dass Grafiken damit nach Bedarf erstellt werden können. Dadurch muss die Grafik nicht bereits beim Ausliefern der Webseite selbst als Bild, welches dann innerhalb eines Img-Elementes eingebettet werden könnte, vorhanden sein.

Mithilfe der JavaScript Bibliothek WebGL können 2D- und 3D-Grafiken ohne weitere benötigte Plugins innerhalb eines Canvas Elements dargestellt werden. Mit WebGL können grafikintensive Anwendungen implementiert und plattformunabhängig innerhalb von Browsern dargestellt werden. WebGL kann dabei direkt nach der Initialisierung der Webseite verwendet werden. Das bedeutet, dass sobald der Zugriff auf das zu zeichnende Canvas-Element verfügbar ist, dieses verwendet werden kann. Dazu gibt es zwei Möglichkeiten, die im nachfolgenden Beispiel (siehe Listing 2.1) demonstriert werden:

1. Verwenden des Events „DOMContentLoaded“
2. Einbinden des Skripts am Ende des Body-Tags

```
1  <html>
2    <head>
3      <script type="text/javascript">
4        // Möglichkeit 1: Verwenden des Events DOMContentLoaded
5        document.addEventListener("DOMContentLoaded", function(
             event) {
6          // DOM-Struktur wurde geladen, Canvas kann verwendet
             werden
7          // Zeichenfläche auslesen
8          var canvas = document.getElementById("example")
9          // WebGL Bibliothek verwenden
10         var webgl = canvas.getContext("webgl")
11
12         if (webgl !== null) {
13           // Auf die Zeichenfläche zeichnen
14           // ...
15         }
16       })
17     </script>
```

```
18    </head>
19    <body>
20      <p>Beispiel WebGL:</p>
21      <canvas id="example" width="640" height="480"></canvas>
22
23      <!-- Möglichkeit 2: Einbinden des Scripts am Ende des
           Body-Tags -->
24      <script type="text/javascript">
25        // DOM-Struktur wurde geladen, Canvas kann verwendet
             werden
26        // Siehe Zeichenscript oben
27        // EventListener DOMContentLoaded ist hier nicht
             notwendig
28      </script>
29    </body>
30  </html>
```

**Listing 2.1:** Initialisierung einer Canvas für WebGL.

In dem vorangegangenen Beispiel wurde zwar die Initialisierung durchgeführt, um aber etwas zu zeichnen, wird weiterer Quellcode benötigt. Möchte man etwa ein Dreieck zeichnen, muss man für WebGL eine exakte Definition hinterlegen, die selbst bei einem simplen Beispiel schon komplex werden kann (Siehe Listing 2.2).

```
1  if (webgl !== null) {
2    // Auf die Zeichenfläche zeichnen
3    // Zeichenfläche initialisieren
4    webgl.viewport(0, 0, canvas.width, canvas.height)
5    webgl.clearColor(0, 0.5, 0, 1)
6    webgl.clear(webgl.COLOR_BUFFER_BIT)
7
8    // Eckpunkte des Dreiecks definieren
9    var vertices = [
10     -0.5,  0.5, 0.0,
11     -0.5, -0.5, 0.0,
12      0.5, -0.5, 0.0,
13   ]
14
15   // Erstellen der einzelnen Buffer
16   // ...
17   // Berechnen der einzelnen Shader
18   // ...
19   // Shader mit den Buffer-Objekten verknüpfen
20   // ...
21   // Objekte zeichnen
22   webgl.drawArrays(webgl.TRIANGLES, 0, 3)
23 }
```

**Listing 2.2:** Verkürzte Definition eines 2D-Objektes für WebGL.

Eine ausführliche Einführung inklusive eines Beispieles in die Entwicklung einer WebGL Anwendung geben Sung, Pavleas, Arnez und Pace (2015). In diesem Beispiel wird sehr gut erklärt, wie ein einfaches Rechteck auf einem farbigen Hintergrund dargestellt werden kann und worauf man bei der Implementierung achten muss. Um komplexere 2D- und 3D-Grafiken darstellen zu können, empfiehlt es sich eine spezialisierte Software zu verwenden. Der Hauptfokus dieser Abschlussarbeit liegt in der Entwicklung einer WebGL Anwendung in der Entwicklungsumgebung in Unity, um welche es im *Abschnitt 2.2 Unity* geht, die genau solche Funktionen mit sich bringt.

## 2.2 Unity

Mithilfe von Unity können 2D- und 3D-Anwendungen entwickelt werden. Unity unterstützt den Entwickler mit unterschiedlichen Tools und bietet die Möglichkeit, Anwendungen mit benutzerdefinierten Skripts, primär in der Programmiersprache C#, zu erweitern. Der interne Marktplatz bietet Erweiterungen, auf deren Basis die Anwendung aufbauen kann. So lassen sich etwa zusätzliche Assets, ohne dass diese erst erstellt werden müssen, einbinden. Genau hier lässt sich Unity aber auch abgrenzen, da Unity nach Seifert und Wislaug (2017) explizit keine 3D-Modellierungssoftware ist. Unity greift dabei auf externe Spezialsoftware, wie zum Beispiel Blender, zurück. Es werden zwar einige primitive 3D-Grundobjekte zur Verfügung gestellt, doch bilden diese auf keinen Fall die Möglichkeiten einer nativen Anwendung ab. Diese Grundobjekte werden Primitives genannt und enthalten die Abbildung von Kugeln, Zylindern oder Quadern und können für kleine Arbeiten genutzt werden. Für komplexere 3D-Modelle empfiehlt es sich speziell darauf angepasste Modellierungssoftwares zu verwenden.

Eine für diese Abschlussarbeit wichtige Funktionalität innerhalb von Unity ist dabei die Möglichkeit, die in Unity entwickelte Anwendung auf mehrere Zielsysteme exportieren zu können. Dabei werden von Seifert und Wislaug (2017) zum Beispiel Windows Desktop, Linux, iOS, Android, PlayStation, Xbox One und WebGL als mögliche Plattformen aufgelistet. Unity bietet dabei laut Echterhoff (2015) seit der Version 5.3 im Jahr 2015 offiziell die Möglichkeit, WebGL als Zielplattform auswählen zu können. Eine WebGL Anwendung bietet den Vorteil, dass sie innerhalb eines Browsers ausgeführt werden kann. Das bedeutet, man kann innerhalb von Unity eine Anwendung in C# entwickeln, die schlussendlich in einem Browser ausgeführt werden kann. Beim Export der Anwendung werden die 3D-Modelle der Anwendung mithilfe der externen Modellierungssoftware in einzelne Vektoren zerlegt und exportiert. Dadurch entfällt die manuelle Definition für WebGL, wie in *Abschnitt 2.1 HTML5 und WebGL* beschrieben. Durch die Transpilierung des Quellcodes und Konvertierung der Modelle benötigt der WebGL-Export jedoch einige Minuten Zeit.

Die Entwicklungsumgebung kann in ihrem Grundaufbau in mehrere einzelne Teilbereiche aufgeteilt werden. Das Hauptfenster stellt dabei die 3D-Szene mit den inkludierten Modellen, Kameras und Lichtquellen dar. Wie von Hardman (2020) beschrieben gibt es zu den einzelnen Objekten in Unity, sogenannte GameObjects, jeweils weitere Eigenschaften, die manuell oder mithilfe von Skripts manipuliert werden können. 3D-Modelle können in diversen Dateiformaten importiert werden. Gängige Formate sind

etwa .blend, .obj oder .fbx. Die verschiedenen GameObjects können hierarchisch aufgebaut und mithilfe von Drag-and-Drop miteinander referenziert werden. Standardmäßig verwendet Unity zur Bearbeitung der Skripts Visual Studio Community.

## 2.3 OPC UA

Angetrieben durch verschiedene Modernisierungsprojekte der Produktion in der Industrie, wie zum Beispiel der Industrie 4.0 oder China 2025, musste unter anderem auch eine Lösung für den Datenaustausch zwischen Maschinen verschiedener Hersteller gefunden werden. Nach Bök, Noack, Müller und Behnke (2020) habe sich OPC UA in den letzten Jahren zu einem der populärsten Kommunikationsstandards entwickelt. Eine der Vorteile dieses Protokolls sei es, dass eine plattformunabhängige Kommunikation zwischen Maschinen ermöglicht werde. Mittlerweile würden viele Hersteller ihre Systeme und Komponenten mit OPC UA Schnittstellen ausstatten. OPC UA steht für Open Platform Communications Unified Architecture und ist grundlegend eine Weiterentwicklung des Protokolls OPC, zu welchem es abwärtskompatibel ist.

Spezifikationen werden von der OPC Foundation erstellt. Die ersten Versionen erschienen dabei im Jahr 2006. Der Standard besteht laut OPC Foundation (2017a, 2017c) aus 15 einzelnen Normen, die unterschiedliche Schwerpunkte beinhalten. In diesen Spezifikationen wird genau beschrieben, welche technischen Möglichkeiten für den Standard angedacht sind. Dies bedeutet aber nicht gleichzeitig, dass diese auch in jeder Implementierung umgesetzt werden müssen. Die Norm OPC Foundation (2017e) beschreibt explizit die einzelnen Funktionalitäten beziehungsweise Service-Sets, die ein OPC UA Server unterstützen kann. Dadurch entsteht je nach Umfang ein eigenes Server-Profil, das sich nach Bök et al. (2020) an die verfügbaren Ressourcen und der Leistungsfähigkeit der Serverplattform richtet.

Ein OPC UA Server besteht nach der Norm OPC Foundation (2017b) unter anderem aus Objekten. Diese Objekte definieren, wie Daten und Funktionen Clients zur Verfügung gestellt werden können. Eines der Hauptelemente innerhalb von OPC UA sind sogenannte Nodes. Ein Node gehört dabei zu einer NodeClass, einem klar definierten Objekt innerhalb des Standards. Nodes bieten die Möglichkeit, dem Client Informationen zu einem im Server gespeicherten Wert zu liefern, wie zum Beispiel eine Beschreibung, den Datentyp des Wertes oder den Zeitpunkt der letzten Änderung. Bei der Definition eines Datentyps kann dabei auf vordefinierte Datentypen zurückgegriffen werden. Bei Bedarf können nach OPC Foundation (2017b) aber auch weitere Datentypen, sogenannte „Complex DataTypes“, definiert werden.

Eine der wichtigsten Funktionen, die innerhalb der Abschlussarbeit benötigt wird, ist in der Norm OPC Foundation (2017d) beschrieben. Dabei geht es um den Aufbau einer Verbindung zum OPC UA Server, sowie das Beschreiben und Lesen von Nodes. Um auf die Werte eines Nodes zugreifen zu können, muss zuerst eine gültige Verbindung zum OPC UA Server aufgebaut werden. Mithilfe der eindeutigen Identifikation eines Nodes können anschließend Operationen angewendet werden. Die Funktionen Lesen und Schreiben liefern dabei den aktuellen Wert des Nodes zurück beziehungsweise

beschreiben diesen mit dem übergebenen Wert. Nach Aufbau einer Verbindung zu einem Server entspricht dies nur einem Verbindungsaufruf. OPC UA bietet hier aber eine weitere Möglichkeit. Der Client kann bei diesem Node eine sogenannte Subscription einrichten. Das Einrichten dieser Subscription beinhaltet einen Verbindungsaufruf. Der Node wird danach bei jeder Statusänderung den Client über diese informieren. Das hat den Vorteil, dass der Client nicht innerhalb fest definierter Zeitabstände die einzelnen Nodes nach deren aktuellen Werten abfragen muss, sondern in Echtzeit vom OPC UA Server über diese Änderung informiert wird. Voraussetzung dafür ist, dass der OPC UA Server die Verbindung mit dem Client aufrechterhält, damit die neuen Daten kommuniziert werden können.

## 2.4 WebSockets

Für die Kommunikation zwischen Client und Server werden oft WebSockets verwendet. Im Gegensatz zu einer normalen HTTP Anfrage bleibt die Verbindung nach dem Verbindungsaufbau aber geöffnet. Dieses Protokoll wird für den Browser nach Gorski, Lo Iacono und Nguyen (2015) von der Organisation W3C spezifiziert, um einen einheitlichen Standard zu erreichen. WebSockets sind außerdem ein fester Bestandteil von HTML5.

Durch die persistente Verbindung kann eine bidirektionale Kommunikation, also in beide Richtungen, erfolgen. Außerdem muss nicht bei jeder Anfrage ein erneuter Verbindungsaufbau initialisiert werden. Dadurch kann eine performantere und effizientere Übertragung stattfinden. Mittlerweile unterstützen laut Gorski et al. (2015) die meisten Browser WebSockets. Eingesetzt werden WebSockets häufig in Anwendungen, die Daten in Echtzeit übertragen müssen.

Mithilfe sogenannter Callbacks können von der Client- und Serverseite auf neue eingehende Nachrichten reagiert werden. Diese werden in verschiedene Kategorien aufgeteilt, wie zum Beispiel den Aufbau der Verbindung oder den Erhalt eines neuen Datensatzes (Siehe Listing 2.3).

```
1  <script type="text/javascript">
2    // Baut eine neue Verbindung auf
3    // Variable urlWebsocket zeigt auf den Kommunikationspartner
4    var ws = new WebSocket(urlWebsocket)
5
6    ws.onopen = function () {
7      // Verbindung erfolgreich aufgebaut
8
9      // ws.send('') sendet eine Nachricht an den
           Kommunikationspartner
10     ws.send('Verbindung wurde erfolgreich aufgebaut')
11   }
12
13   ws.onmessage = function (message) {
14     // Neue Nachricht erhalten
```

```
15      // Parameter message enthält die übertragene Nachricht
16    }
17
18    ws.onerror = function (error) {
19      // Fehler aufgetreten
20      // Parameter error enthält Informationen über den Fehler
21    }
22
23    ws.onclose = function (closeMessage) {
24      // Verbindung wurde geschlossen
25      // Parameter closeMessage enthält Informationen über den
          Grund
26    }
27  </script>
```

**Listing 2.3:** Implementation eines WebSockets in JavaScript.

## 2.5 WebAssembly

Webanwendungen werden heutzutage nach Rourke (2018) immer noch nachgesagt, im Vergleich zu nativen Anwendungen performant schlechter abzuschneiden. Um die Geschwindigkeit zu erhöhen hat die Standardisierungsorganisation W3C WebAssembly als Bestandteil aktueller Browser spezifiziert. WebAssembly soll dabei JavaScript nicht ersetzen, sondern als eigenständige Komponente zusätzlich integriert werden.

Mithilfe von WebAssembly, oft als Wasm abgekürzt, soll Bytecode erstellt werden, der im Browser ausgeführt werden kann. Von Rourke (2018) wird beschrieben, dass durch die Verwendung des Binärformates eine fast native Geschwindigkeit erreicht werden kann. Außerdem ist es den meisten, in anderen Programmiersprachen geschriebenen Bibliotheken, wie zum Beispiel in C und C++, möglich, Quellcode in dieses Format zu exportieren. Dadurch können zum Beispiel existierende, auf Leistung optimierte Komponenten im Browser wiederverwendet werden.

Zurzeit enthält WebAssembly noch einen kleinen Umfang an Features und beinhaltet laut Fras und Nowak (2020) einige Limitationen, die sich aber Schritt für Schritt auflösen. Im Browser müssen WebAssembly-Module zum Beispiel noch mit JavaScript verknüpft werden, damit diese geladen und verwendet werden können. Zusätzlich ist es den geladenen Modulen noch nicht möglich mit dem DOM zu interagieren. Zurzeit beinhaltet die Einarbeitung in WebAssembly eine steile Lernkurve, da unter anderem Speicher manuell angefordert werden muss und nur wenig Datentypen unterstützt werden. Die Verwendung einer Zeichenkette ist zum Beispiel noch nicht möglich.

# Kapitel 3

# Forschungsstand und Forschungslücke

Nachdem im vorherigen Kapitel die theoretischen Grundlagen der Technologien für die Kommunikation einer WebGL Anwendung mit einem OPC UA Server erläutert wurden, werden in den folgenden Abschnitten verwandte Arbeiten ausgewertet. Dies schafft einen Überblick, welche Probleme und Lösungen es in diesem Forschungsgebiet bereits gibt. Dabei werden vor allem Limitierungen, die bei der Durchführung dieser Abschlussarbeit auftreten können, betrachtet. Am Ende dieses Kapitels wird im letzten Abschnitt ein Fazit gezogen.

## 3.1 Verwandte Arbeiten

In diesem Abschnitt soll auf gängige Probleme und Lösungen in Bezug auf Systeme, die mit OPC UA kommunizieren, eingegangen werden, indem ein Überblick über die existierenden Varianten geschafft wird. Soweit möglich wurde nach webfähigen Systemen gefiltert. Dabei konnten unterschiedliche Herangehensweisen identifiziert werden. Größtenteils werden dabei Systeme für zwei Anwendungsfelder erforscht:

- SCADA
- Mixed Reality

**SCADA** steht für Supervisory Control and Data Acquisition und dient zur Überwachung und Steuerung von Industrieanlagen. Die Sensordaten werden dabei gesammelt und in einer Oberfläche visualisiert, um daraus weitere Erkenntnisse schließen zu können. Einige SCADA Systeme besitzen bereits Komponenten zur Erfassung von OPC UA Datenströmen.

**Mixed Reality Systeme** verfolgen ähnliche Ziele wie SCADA Systeme. Dabei soll dem Benutzer mithilfe aktueller Maschinendaten der Status einer oder mehrerer Maschinen angezeigt werden. Beim Beobachten einer Industrieanlage durch den Benutzer eines solchen Systems soll diese dabei um Zustands- und Zusatzinformationen erweitert werden. In manchen Anwendungsfällen wird dabei auch die Steuerung einer solchen Industrieanlage durch den Benutzer ermöglicht.

In den folgenden Abschnitten sollen dabei einige Beispiele aufgeführt werden, wie solche Anwendungen umgesetzt werden könnten. Dabei existieren einige Limitierungen, die es kritisch im Zusammenhang mit der Forschungsfrage dieser Abschlussarbeit zu hinterfragen gilt.

### 3.1.1 Kommunikation eines Laserplotters mit einer Microsoft HoloLens

#### Zusammenfassung

In der Arbeit von Olbort, Herden, Kutscher und Anderl (2020) wird ein Konzept gezeigt, wie ein Laserplotter mit einer Microsoft HoloLens kommunizieren kann. Dazu wurde ein Prototyp einer Mixed Reality Anwendung entwickelt. Die Mixed Reality Anwendung wird dabei mithilfe von Unity entwickelt. Zu einem vorhandenen CAD-Modell sollen zusätzliche Informationen angezeigt werden. Diese zusätzlichen Daten kommen von einem OPC UA Server, der auf einem Raspberry Pi installiert wurde. Nach Olbort et al. (2020) steht für Unity keine kompatible Bibliothek zur Verfügung, die eine Kommunikation mit einem OPC UA Server ermöglicht. Deshalb wurde in dieser Implementierung der Ansatz gewählt, auf dem Raspberry Pi eine Datenbank und einen Webserver einzurichten. Mithilfe eines OPC UA Clients auf dem Raspberry Pi wird eine Verbindung zu dem OPC UA Server aufgebaut und in einem bestimmten Zeitintervall die jeweils aktuellen Daten in die Datenbank geschrieben. Auf diese Informationen kann schließlich mithilfe eines PHP-Skripts zugegriffen werden. Laut Olbort et al. (2020) wurde nur ein lesender Zugriff auf die Maschinendaten umgesetzt. Ein bidirektionaler Informationsfluss sei sehr wünschenswert, wurde innerhalb dieser Arbeit jedoch nicht umgesetzt.

#### Reflexion

Aufgrund der Implementierung des Prototypen von Olbort et al. (2020), welche die Maschinendaten zuerst in eine Datenbank für einen späteren Abruf speichert, kann keine aktuelle Anzeige von Informationen gewährleistet werden. Mithilfe des PHP-Skripts werden Datensätze zeitverzögert, nach Abhängigkeit des in der Anwendung definierten Zeitintervalls, angefordert und angezeigt. Dieses Konzept liegt im Widerspruch mit der gewünschten Umsetzung dieser Abschlussarbeit, die Daten in Echtzeit zu übertragen. Im Optimalfall wird die WebGL Anwendung bei einer Aktualisierung eines Sensors darüber informiert. Dadurch wäre es nicht notwendig, innerhalb eines festgelegten Zeitintervalls Abfragen an den Server zu senden, um zu erfahren, ob es eine Änderung der Sensordaten gab. Mit den frei verfügbaren Bibliotheken, wie zum Beispiel opc-ua-client, UA.NET Standard sowie der proprietären Bibliothek game4automation, gibt es zudem einige verfügbare OPC UA Clients, die in C# geschrieben wurden und Unity verwendbar sind.

### 3.1.2 Industrie 4.0 mit HoloBuilder

**Zusammenfassung**

Speicher, Tenhaft, Heinen und Handorf (2015) stellen in Ihrer Arbeit die Plattform „HoloBuilder“ vor. Hier wird gezeigt, wie eine Mixed Reality Anwendung mithilfe dieser Webanwendung erstellt werden kann. Relevant für diese Abschlussarbeit ist hier vor allem die Möglichkeit, diese um eine OPC UA Anbindung erweitern zu können. Erwähnt wird die Anzeige der 3D-Modelle in einem Browser, die innerhalb einer WebGL Anwendung stattzufinden scheint. In einer Fallstudie erstellen Speicher et al. (2015) dabei eine Mixed Reality Anwendung, in der eine Stanzmaschine mit zusätzlichen Maschinendaten erweitert wird. Dabei wird außerdem gezeigt, dass eine bidirektionale Kommunikation möglich ist.

**Reflexion**

Die technischen Details der Kommunikation zwischen der Mixed Reality Anwendung sowie der Maschine werden hier nicht näher erläutert. Ob es sich hier um eine native Anbindung der Anwendung an den OPC UA Server handelt oder ob eine Lösung wie im vorherigen Abschnitt verwendet wird, ist aufgrund der proprietären Lösung nicht nachvollziehbar. Zwar wird in der Arbeit erwähnt, dass die Anwendung auch im Browser aufrufbar ist, jedoch wird daraus nicht klar ersichtlich, ob es sich dabei auch um die OPC UA Komponente handelt.

### 3.1.3 Evaluation eines webbasierenden SCADA Systems

**Zusammenfassung**

In seiner Arbeit evaluiert Paronen (2015) die Fähigkeiten und Limitierungen von Web-Technologien für die Erstellung eines SCADA Systems. Dabei wird ein webfähiger OPC UA Client erstellt, mit dem einzelne Funktionalitäten eines SCADA Systems analysiert werden können. Paronen (2015) vergleicht anschließend seine Erkenntnisse mit anderen webbasierenden SCADA Systemen. Innerhalb dieser Arbeit zeigt Paronen (2015) drei Möglichkeiten auf, wie OPC UA Daten in einem Browser verwendet werden könnten:

**Pre-rendered user interfaces**

Eine der Möglichkeiten sei, laut Paronen (2015), das Verwenden einer Thin-Client Architektur. Dabei wird dem Client die ganze Webseite mit den aktuellen OPC UA Informationen ausgeliefert. Der Server kann an verschiedene OPC UA Server angebunden werden und erstellt zum Zeitpunkt der Client-Anfrage die angeforderte Webseite. Der Client selbst kommuniziert daher nicht direkt mit dem OPC UA Server. Einige Hersteller bieten dem Client aber zusätzlich noch die Möglichkeit, aktuelle OPC UA Daten mithilfe von XHR-Anfragen vom Webserver abzufragen. Dabei wird dann nicht mehr die ganze Webseite, sondern nur der jeweilige Teil zurückgegeben und aktualisiert.

#### Web APIs

Die Kapselung des Zugriffs auf einen OPC UA Server hinter einer Webschnittstelle sei, nach Paronen (2015), eine weitere Möglichkeit. Dabei wird der Zugriff auf einen OPC UA Node mit einer REST-Schnittstelle ermöglicht. Als umgesetzte proprietäre Lösung führt Paronen (2015) dabei HyperUA von Projexsys an.

Einen ähnlichen Ansatz zur Umsetzung einer Webschnittstelle von Schiekofer, Scholz und Weyrich (2018) sei es, das OPC UA Protokoll generell um eine REST-Schnittstelle zu erweitern. Hierbei gehe man davon aus, dass man jeden Node einer eindeutigen URL zuordnen könne. Dabei existiere aber das grundsätzliche Problem, dass OPC UA kein zustandsloses Protokoll sei. OPC UA benötige eine Session, die vor der Abfrage von Daten eines Nodes aufgebaut werden müsse. Zwar ließen sich laut Schiekofer et al. (2018) grundlegende Operationen, wie Lesen und Schreiben, durchführen, möchte man aber bei Änderung eines Sensors benachrichtigt werden, sei dies nicht möglich. Die Informationen zum Client, die zur Kommunikation mit diesem benötigt werden, seien zu diesem Zeitpunkt nicht mehr verfügbar.

#### Native Stack

Paronen (2015) führt in seiner dritten Möglichkeit an, dass aufgrund der Tatsache, dass die meisten Browser JavaScript unterstützen, es möglich sein müsste, OPC UA Server direkt aus einem Browser ansprechen zu können. Dabei werden Versuche von Hennig, Braune und Damm (2010) sowie Freund, Martin, Braune und Steinkrauss (2013) erwähnt, in denen eine native Kommunikation zwischen einem Browser und einem OPC UA Server möglich seien.

#### Reflexion

Die erste vorgestellte Möglichkeit von Paronen (2015), in der Webseiten vorgefertigt ausgeliefert werden, scheinen für SCADA-Systeme stimmig zu sein. Für den Anwendungsfall dieser Abschlussarbeit, in der eine WebGL Anwendung Daten mit einem OPC UA Server austauschen soll, ist dies aber nicht möglich. WebGL zeichnet den Inhalt der Anwendung in ein Canvas-Element und erfordert keine DOM-Struktur. Benötigt werden hier nur die Datensätze der OPC UA Nodes. Wünschenswert ist aufgrund dessen die Umsetzung mithilfe eines Native Stacks. Nach der Recherche innerhalb dieser Abschlussarbeit konnten die erwähnten Bibliotheken von Hennig et al. (2010) sowie Freund et al. (2013) zur weiteren Analyse aber nicht gefunden werden. Möglich sei deshalb zwar immer noch die Implementierung einer Webschnittstelle, wie von Paronen (2015) oder Schiekofer et al. (2018) beschrieben wurde, doch gebe es dabei Limitationen bei der Benachrichtigung der Clients, sobald neue Sensorwerte verfügbar sind.

## 3.2 Fazit

In den meisten Studien zur Kommunikation mit OPC UA Servern wird keine direkte bidirektionale Kommunikation mit einer Webanwendung angestrebt. Oft wird ein

Proxy verwendet, um zwischen verschiedenen Protokollen zu vermitteln. Dabei wurde bei den demonstrierten Konzepten der Vorteil verloren, bei einer Statusänderung eines Sensors automatisch benachrichtigt werden zu können. Native Ansätze zur direkten Kommunikation einer Webanwendung mit einem OPC UA Server scheint es nicht zu geben. Die hohe Anzahl proprietärer Lösungen lässt dies aber nicht vollkommen ausschließen.

Zur Erreichung des Ziels der Abschlussarbeit wird deshalb im nächsten Kapitel ein Anforderungskatalog an einen Proof of Concept formuliert, um anschließend ein Konzept auszuarbeiten, wie eine mögliche Lösung ohne diese Limitierungen aussehen könnte.

# Kapitel 4

# Anforderungsanalyse

Nachdem im vorherigen Kapitel ein Überblick über den aktuellen Forschungsstand gegeben sowie zu die schließenden Forschungslücken benannt wurden, müssen in diesem Kapitel konkrete Anforderungen an einen umzusetzenden Proof of Concept definiert werden. Innerhalb dieses Kapitels werden Anforderungen formuliert, zu denen im nächsten Kapitel ein Konzept erarbeitet wird, um mit den derzeit im Forschungsfeld vorhandenen Limitationen umgehen zu können. Die Anforderungen werden außerdem innerhalb eines Anforderungskatalogs festgehalten, um diese im *Kapitel 7 Evaluierung* gegenprüfen zu können. Hiermit kann festgestellt werden, ob noch weitere Limitierungen existieren und welche Erkenntnisse daraus gezogen werden können.

## 4.1 Anforderungskatalog

In diesem Abschnitt werden Anforderungen an ein Proof of Concept definiert, die nach der Umsetzung evaluiert werden. Dazu werden Anforderungen an ein System gestellt, mit denen die allgemeinen Forschungsfragen dieser Abschlussarbeit beantwortet werden sollen. Die funktionalen Anforderungen werden gesammelt und detailliert beschrieben. Zusätzlich werden diese Anforderungen für die weitere Vorgehensweise priorisiert. Um im weiteren Verlauf dieser Abschlussarbeit einfacher auf die hier definierten funktionalen Anforderungen referenzieren zu können, wird das Kürzel FA verwendet. Bei der in den Anforderungen genannten WebGL Anwendung handelt es sich um aus den mit Unity generierten Export.

### 4.1.1 Anforderung FA-1

| | |
|---|---|
| **Identifikator** | FA-1 |
| **Beschreibung** | Das System muss der WebGL Anwendung das Auslesen und Beschreiben eines OPC UA Nodes ermöglichen. |
| **Zielsetzung** | Die WebGL Anwendung kann auf aktuelle Werte von OPC UA Nodes zugreifen. |
| **Priorität** | Hoch |
| **Anmerkungen** | Eine der grundlegendsten Funktionalitäten, die das System unterstützen muss, ist die Kommunikation zwischen den beiden Hauptkomponenten, einer WebGL Anwendung und einem OPC UA Server. Erfüllt das System diese Anforderung, dann ist es möglich, mit der WebGL Anwendung Nodes auszulesen und zu beschreiben. |

**Tabelle 4.1:** Detaillierte Spezifikation der Anforderung FA-1.

### 4.1.2 Anforderung FA-2

| | |
|---|---|
| **Identifikator** | FA-2 |
| **Beschreibung** | Das System muss es dem OPC UA Server ermöglichen, die WebGL Anwendung bei einer Änderung eines OPC UA Nodes über den aktuellen Status zu informieren. |
| **Zielsetzung** | Die WebGL Anwendung wird über eine Änderung eines Nodes benachrichtigt. |
| **Priorität** | Hoch |
| **Anmerkungen** | Im vorherigen *Kapitel 3 Forschungsstand und Forschungslücke* wurde festgestellt, dass es zurzeit keine Möglichkeit gibt, diese Anforderung in einer WebGL Anwendung zu erfüllen. Innerhalb dieser Abschlussarbeit wird deshalb auf mögliche Lösungen eingegangen. |

**Tabelle 4.2:** Detaillierte Spezifikation der Anforderung FA-2.

### 4.1.3 Anforderung FA-3

| | |
|---|---|
| **Identifikator** | FA-3 |
| **Beschreibung** | Das System muss während der Entwicklungszeit in Unity eine Anbindung an OPC UA Server ermöglichen. |
| **Zielsetzung** | Die Anwendung muss auf dem Hostsystem nativ ausführbar sein. |
| **Priorität** | Mittel |
| **Anmerkungen** | Zur Anbindung neuer OPC UA Server ist es notwendig, diese in der WebGL Anwendung zu definieren. Unity bietet die Möglichkeit, die noch nicht exportierte WebGL Anwendung als native Anwendung innerhalb des Hostsystems auszuführen. Das erhöht die Entwicklungsgeschwindigkeit erheblich, da allein ein einzelner Export der Anwendung als WebGL Anwendung mehrere Minuten benötigt. Es ist deshalb wünschenswert, das System so zu gestalten, dass dieser Vorteil nicht verloren geht. |

**Tabelle 4.3:** Detaillierte Spezifikation der Anforderung FA-3.

### 4.1.4 Anforderung FA-4

| | |
|---|---|
| **Identifikator** | FA-4 |
| **Beschreibung** | Das System muss dem Entwickler eine einfache Verwendung der OPC UA Nodes in Unity ermöglichen. |
| **Zielsetzung** | Mithilfe von kompatiblen GameObjects muss auf die Werte verknüpfter OPC UA Nodes zugegriffen werden können. |
| **Priorität** | Niedrig |
| **Anmerkungen** | Bei der Entwicklung von komplexen Anwendungen in Unity soll dem Entwickler eine Möglichkeit geschafft werden, einfach auf sich ändernde Strukturen der OPC UA Server reagieren zu können. Dabei ist es angedacht, Daten von OPC UA Nodes mithilfe von Drag-and-Drop in Unity verwenden zu können. |

**Tabelle 4.4:** Detaillierte Spezifikation der Anforderung FA-4.

# Kapitel 5

# Konzept

In diesem Kapitel wird ein Konzept für die Umsetzung des Proof of Concepts und dessen definierten Anforderungen aus dem vorherigen Kapitel vorgestellt. Hinführend zu diesem Konzept werden im ersten Abschnitt mögliche Alternativen und ihre Limitierungen erörtert. Das Konzept für die weitere Vorgehensweise wird im *Abschnitt 5.2 Proxy zwischen OPC UA und WebSockets* ausführlich beschrieben.

## 5.1 Grundlegende Überlegungen

Innerhalb dieses Abschnittes werden Überlegungen vorgestellt, wie in Zukunft eine optimale Kommunikation zwischen einer WebGL Anwendung und einem OPC UA Server aussehen könnte. Dabei müssen zurzeit geltende technische Spezifikationen miteinbezogen werden.

Die Funktionalität eines OPC UA Servers wird definiert durch die Umsetzung der Normen der OPC UA Foundation. Dadurch entstehen, je nach umgesetzten Umfang, unterschiedliche Bibliotheken. Ein OPC UA Server auf einem Mikrocontroller benötigt gegebenenfalls weniger Funktionalitäten als auf einer größeren Industrieanlage. Diese Normen werden von der OPC Foundation definiert und sind öffentlich verfügbar. Dadurch können OPC UA Bibliotheken entwickelt werden, die frei verfügbar sind. Die für die im vorherigen Kapitel definierten Anforderungen FA-1 und FA-2 benötigten Funktionalitäten sind im vierten Teil der Spezifikation der OPC Foundation (2017e) detailliert beschrieben.

FA-1 benötigt einen einfachen Lese- und Schreibzugriff auf einen OPC UA Node. Um FA-2 umsetzen zu können, müssen Informationen vom Server zum Client gesendet werden können. Der Client muss sich dazu bei einem OPC UA Server registrieren, eine sogenannte Subscription etablieren. Dabei muss eine Verbindung aufrechterhalten werden, damit der OPC UA Server Daten an die Anwendung übertragen kann. Für diese Anforderungen gibt es bereits einige Bibliotheken, die dies erfolgreich umsetzen, dementsprechend muss nicht alles neu entwickelt werden.

Um Unity um einen OPC UA Client zu erweitern, muss die Entwicklungsumgebung

eine kompatible Bibliothek, wie zum Beispiel opc-ua-client oder UA.NET Standard, einbinden. Anschließend kann für das Hostsystem eine Anwendung erstellt werden, die mit einem OPC UA Server kommunizieren kann. Diese hier aufgeführten Schritte würden dem in der Abbildung 5.1 skizzierten Aufbau entsprechen.

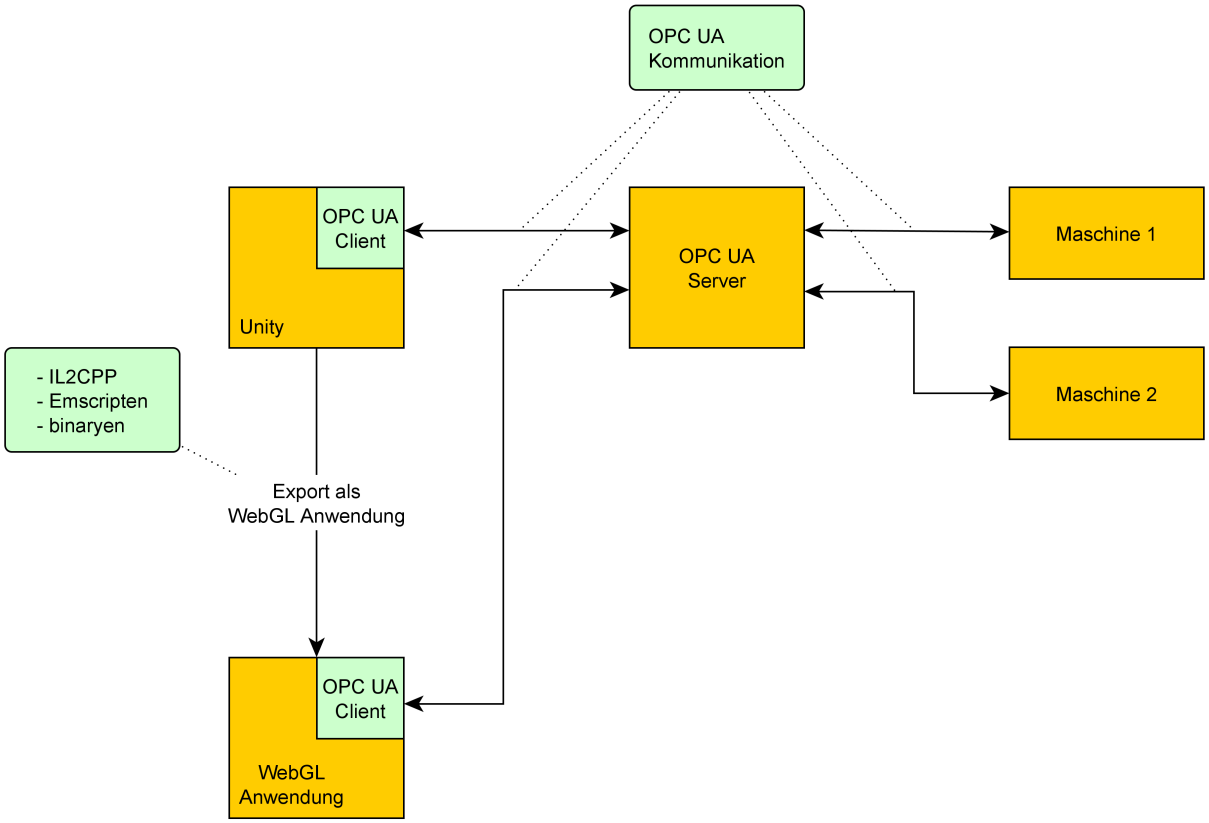


**Abbildung 5.1:** Grundlegende Idee der Kommunikation einer WebGL Anwendung und Unity mit einem OPC UA Server.

Dieses Konzept lässt sich jedoch nicht wie gezeigt umsetzen. Der kritische Schritt liegt dabei bei der Konvertierung einer existierenden Bibliothek eines OPC UA Clients in ein WebAssembly-Modul für die WebGL Anwendung. Um dies genauer zu verstehen wird kurz erläutert, wie Unity eine für das Hostsystem native Unity Anwendung zu einer WebGL Anwendung konvertiert.

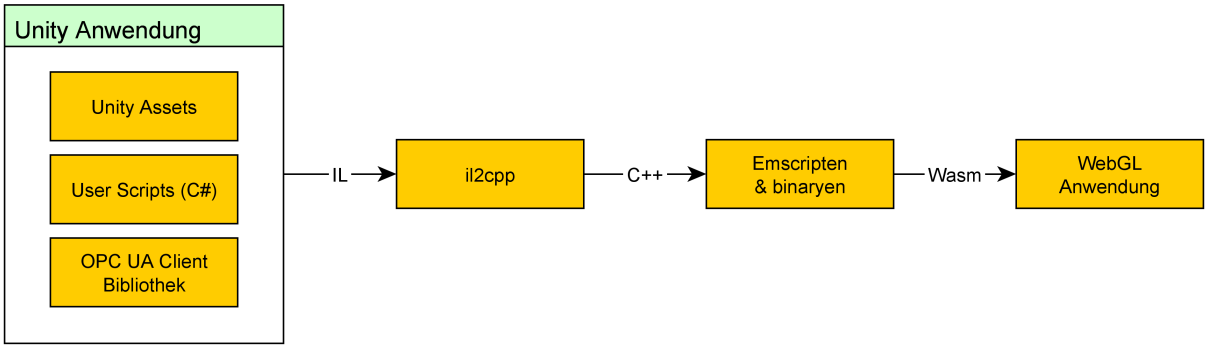


**Abbildung 5.2:** Export einer Unity Anwendung zu einer WebGL Anwendung.

Wie man in Abbildung 5.2 erkennen kann, ist es dabei irrelevant, welcher OPC UA Client in Unity verwendet wird. Man kann OPC UA Clients, die in verschiedenen Programmiersprachen implementiert wurden, verwenden. Unity konvertiert diese mit dem internen Tool IL2CPP in einem Zwischenschritt zu C++ Code, der schlussendlich mit den beiden Tools emscripten und binaryen zu einem WebAssembly-Modul konvertiert wird.

Für die schlussendliche Ausführung des WebAssembly-Moduls relevant ist der Browser selbst. Innerhalb eines Browsers werden Anwendungen in einer sogenannten „Sandbox“ ausgeführt. Dabei werden Berechtigungen aus Sicherheits- und Kompatibilitätsgründen eingeschränkt. Hier kann verhindert werden, dass Anwendungen außerhalb des Browsers zum Beispiel Daten aus dem Hostsystem auslesen können. Im Falle eines zu importierenden OPC UA Clients führt genau dieser Vorteil aber zu einem Problem. Diese Bibliotheken benötigen für die asynchrone Ausführung beziehungsweise zum Aufbau von Verbindungen bestimmte Technologien, wie zum Beispiel SSL, Sockets und Multithreading, die mit den restriktiven Einschränkungen der Browser nicht ausführbar sind.

Erwähnenswert ist, dass in Zukunft die erwähnten Probleme möglicherweise mithilfe des „WebAssembly System Interface“ gelöst werden könnten. Das Ziel dieses Projekts ist es, Lösungen für zuvor genannte Limitierungen und Berechtigungsprobleme zu finden. Sollte es Neuerungen in diesem Projekt zu diesem Thema geben, lohnt es sich, den Ansatz des Exports eines OPC UA Clients als WebAssembly-Modul erneut zu evaluieren.

Innerhalb meiner Abschlussarbeit habe ich mich dazu entschlossen nicht weiter nach einer Möglichkeit zur Konvertierung eines OPC UA Clients in ein WebAssembly-Modul zu suchen. Das Ziel dieser Abschlussarbeit ist es einen funktionsfähigen Proof of Concept zu demonstrieren und der zuvor beschriebene Ansatz scheint aufgrund der vorliegenden Limitierungen nicht zielführend zu sein. Deshalb wird im nächsten Abschnitt das schlussendlich verwendete Konzept vorstellt.

## 5.2 Proxy zwischen OPC UA und WebSockets

In diesem Abschnitt wird ein Konzept für die Umsetzung eines Proof of Concept vorgestellt. Dabei wird nicht, wie im vorherigen Abschnitt erläutert, eine existierende OPC UA Client Bibliothek in die WebGL Anwendung als WebAssembly-Modul exportiert. Die Grundidee dieses Konzeptes ist es, ein Protokoll zu verwenden, das als Substitution einer OPC UA Verbindung hin zu einer Schnittstelle, die diese Kommunikation in das eigentliche OPC UA Protokoll übersetzt, zu verwenden.

Im *Kapitel 3 Forschungsstand und Forschungslücke* konnten bereits solche Ansätze vorgestellt werden. Datensätze des OPC UA Servers wurden dabei in Datenbanken zwischengespeichert, um diese dann später mithilfe eines weiteren Aufrufs aus der Anwendung anzufordern. Dadurch konnte jedoch keine bidirektionale Kommunikation, also der Aufbau einer Subscription von der Anwendung zum OPC UA Server wie in FA-2 beschrieben, hergestellt werden.

Innerhalb der Abschlussarbeit wurde deshalb das Konzept entworfen, mithilfe des WebSocket Protokolls in der WebGL Anwendung eine Verbindung zu einer Schnittstelle aufzubauen, die als Proxy fungiert. Beim Verbindungsaufbau erkennt der Proxy diesen Versuch und öffnet eine Verbindung zum angeforderten OPC UA Server. Somit wird eine Verbindung von der WebGL Anwendung bis hin zum OPC UA Server ermöglicht (Siehe Abbildung 5.3). Zusätzlich kann eine Subscription zu einem oder mehreren OPC UA Nodes angelegt werden. Durch die Bidirektionalität der Verbindung ist außerdem eine Benachrichtigung des Clients durch den Server bei einer Statusänderung einer Maschine möglich.

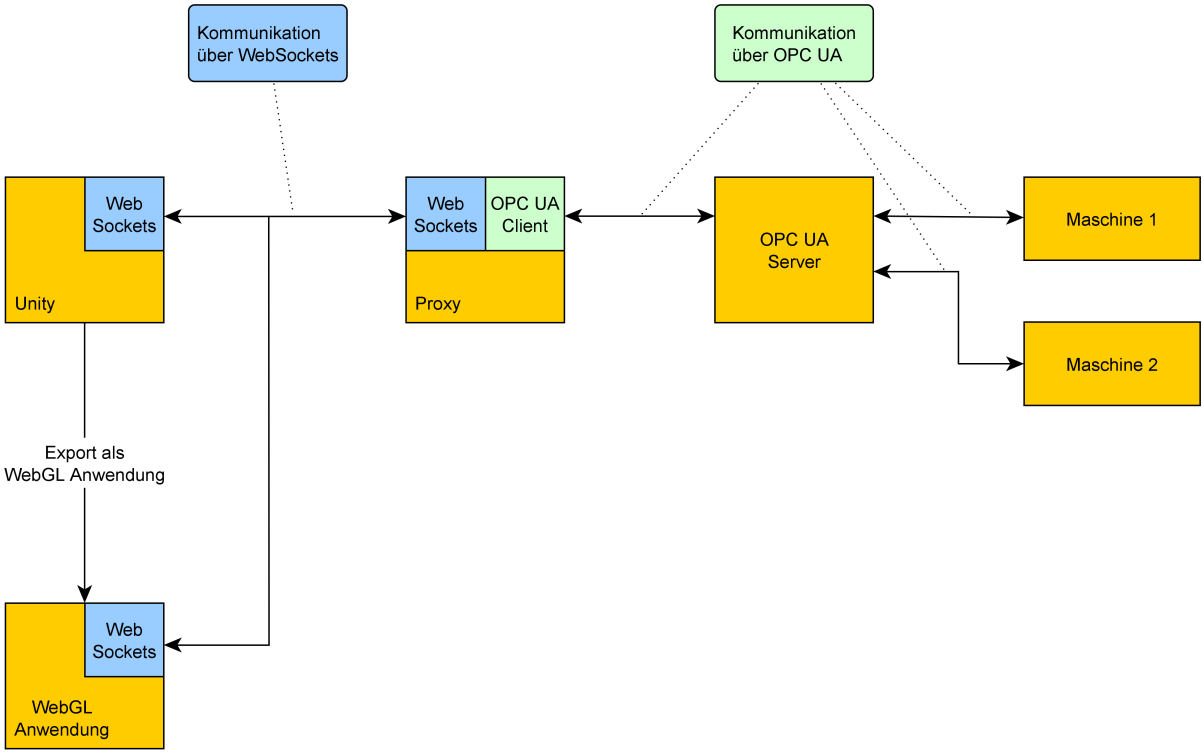


**Abbildung 5.3:** Konzept einer Kommunikation einer WebGL Anwendung mit einem OPC UA Server.

Die Schnittstelle selbst muss dabei nur aus zwei Komponenten bestehen. Auf der einen Seite benötigt sie die Möglichkeit, WebSocket-Verbindungen aufzubauen, um mit der WebGL Anwendungen kommunizieren zu können. Auf der anderen Seite muss sie einen OPC UA Client integrieren, um Verbindungen mit dem OPC UA Server aufbauen zu können. Dabei ist es irrelevant welche Bibliotheken oder Programmiersprachen verwendet werden.

Mit diesem Konstruktionsaufbau ist es möglich, die Anforderungen FA-1 und FA-2 umzusetzen. Es können Datensätze auf OPC UA Nodes geschrieben und ausgelesen werden. Der Client kann sich außerdem für eine längere Zeit registrieren, um bei Änderungen von Sensordaten benachrichtigt zu werden. Das ist möglich, da die WebSocket-Verbindung zwischen der WebGL Anwendung und dem Proxy ebenso wie Verbindung zwischen Proxy und OPC UA Server aufrechterhalten werden können und somit die Informationen zum Client nach Verbindungsaufbau nicht verloren gehen.

Ein weiterer Vorteil dieses Konzeptes ist es, dass nach dem initialen Verbindungsaufbau von der WebGL Anwendung zum Proxy die zu versendenden Datenpakete nicht mehr den kompletten Header wie in einer standardmäßigen HTTP-Anfrage enthalten müssen, da die beiden Kommunikationsparteien sich bereits gegenüber verifiziert haben (Siehe Gorski et al. (2015)). Dadurch lässt sich eine schlankere Kommunikation, als in einigen Arbeiten im *Kapitel 3 Forschungsstand und Forschungslücke* beschrieben, implementieren, da in diesen normale HTTP-Anfragen verwendet wurden. Im Vergleich zu normalen HTTP-Anfragen müssen WebSockets nicht bei jeder Datenübertragung die Verbindung neu initialisieren. Die Verbindung zwischen Client und Server bleibt erhalten und es können bidirektional Daten übertragen werden. Dadurch, dass die Verbindung bestehen bleibt, können bei jeder Anfrage mehrere hundert Bytes eingespart werden. Außerdem ermöglicht die Persistenz der Verbindung dem Server, Statusänderungen einer Maschine direkt dem Client zu übermitteln, ohne dass dieser sie erst verzögert anfragen müsste. Das sogenannte „Polling“, das in den Arbeiten im *Kapitel 3 Forschungsstand und Forschungslücke* oft verwendet wurde, wird somit überflüssig.

### 5.2.1 Konzeption FA-1

In diesem Abschnitt werde ich kurz auf die geplante Umsetzung der Anforderung FA-1 eingehen. FA-1 beschreibt das Auslesen sowie Beschreiben eines OPC UA Nodes. Der Ablauf beim Beschreiben entspricht dem des Auslesens wie in Abbildung 5.4 ersichtlich, zusätzlich wird aber noch der zu setzende Wert übermittelt. Der Client, die WebGL Anwendung, öffnet dazu einen WebSocket zum Proxy. Anschließend wird eine Anfrage zum Abfragen eines Wertes an den Proxy gesendet. Dieser öffnet eine OPC UA Verbindung zum OPC UA Server und leitet diese Anfrage weiter. Als Antwort erhält der Proxy vom OPC UA Server den aktuellen Wert des OPC UA Nodes und leitet diese Rückgabe an die noch offene WebSocket-Verbindung des Clients weiter. Dadurch kann auf Anfrage des Clients der aktuelle Wert eines OPC UA Nodes zurückgeliefert oder gesetzt werden.

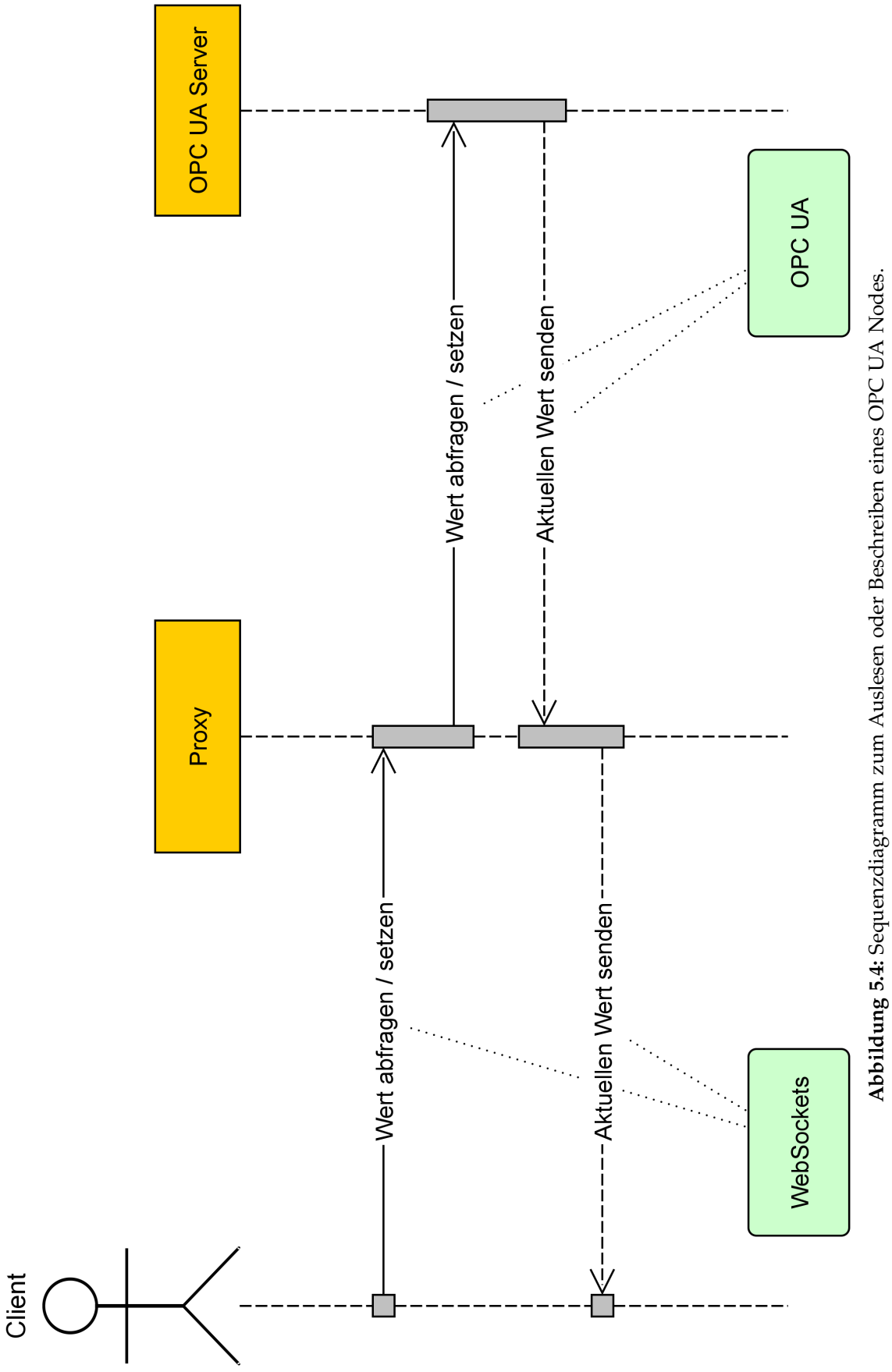


**Abbildung 5.4:** Sequenzdiagramm zum Auslesen oder Beschreiben eines OPC UA Nodes.

### 5.2.2 Konzeption FA-2

Die Konzeption der Anforderung zur Benachrichtigung eines Clients bei einer Statusänderung wird im folgenden Abschnitt thematisiert. Damit der Server dem Client bei jeder Änderung einer Maschine die aktuellen Werte übermitteln kann, muss er diesen identifizieren können. Dazu registriert sich der Client zuerst über einen WebSocket mithilfe des Proxys beim OPC UA Server (Siehe Abbildung 5.5).

Die beim Proxy eingegangene Anfrage zur Registrierung wird zu einer Erstellung einer neuen Subscription beim OPC UA Server übersetzt. Ab dem Zeitpunkt der Etablierung der Subscription wird der Proxy bei jeder Änderung mit den neuen Werten informiert. Der Proxy überträgt diese über den noch geöffneten WebSocket zurück an den anfangs registrierten Client.

Bei der Erstellung einer Subscription muss sich der Client nicht erneut für die nächste Änderung registrieren, da der WebSocket durch die persistente Verbindung erhalten bleibt. Das bedeutet, dass durch das wiederholte Setzen des Status einer Maschine dieser erneut über den Proxy an den Client übermittelt wird, ohne dass dieser aktiv den aktuellen Wert anfragen muss.

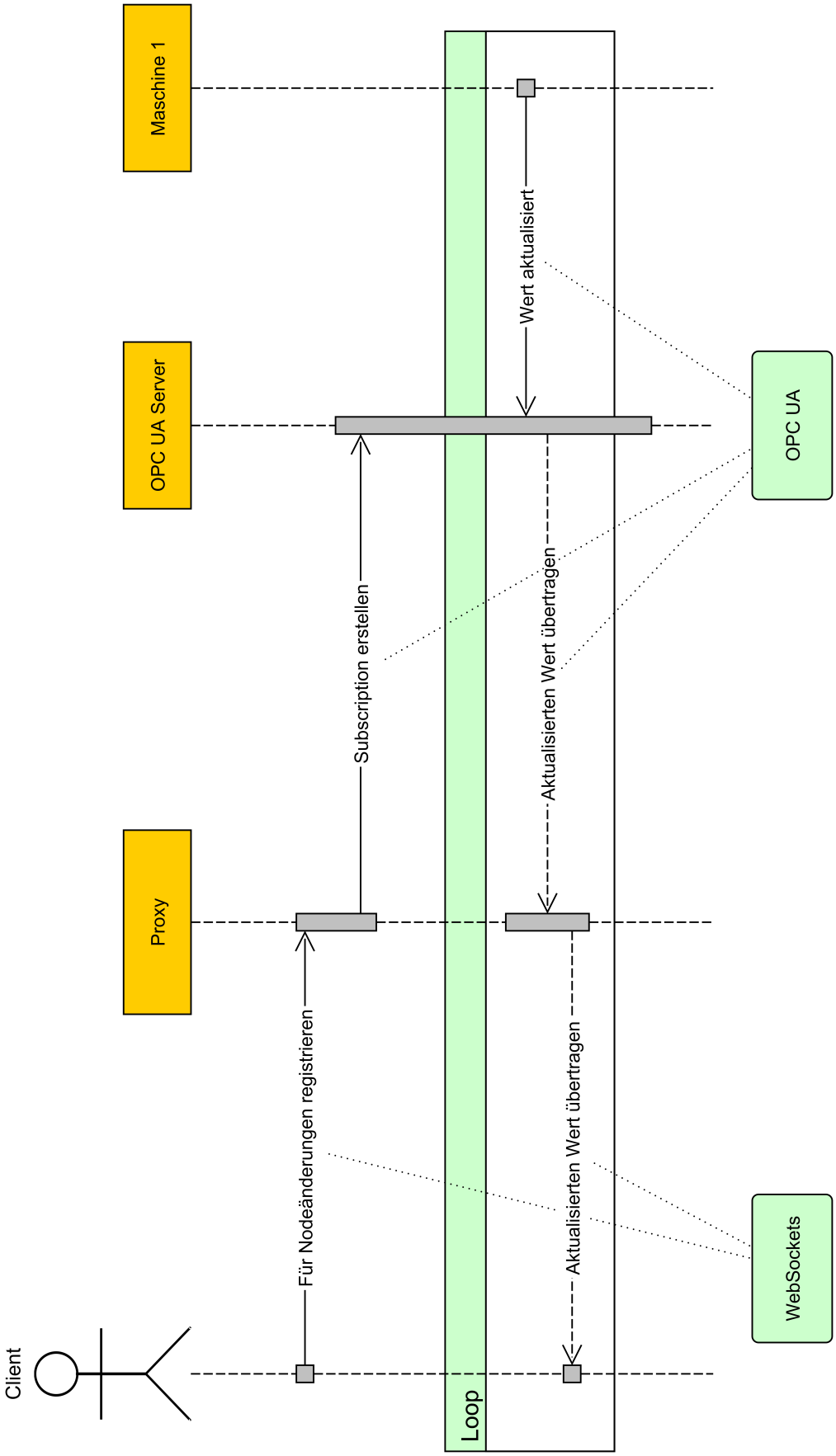


**Abbildung 5.5:** Sequenzdiagramm zur Benachrichtigung eines Clients über die Änderung eines OPC UA Nodes.

### 5.2.3 Konzeption FA-3

Ziel der FA-3 ist es, dem Entwickler die Möglichkeit zu geben, die Anwendung nativ innerhalb von Unity sowie nach einem Export als WebGL Anwendung ausführen zu können. Dazu ist eine einheitliche WebSockets Schnittstelle zum Proxy, der die Anfragen an den OPC UA Server weiterleitet, notwendig. Zwar werden für die einzelnen Programmiersprachen Implementierungen für das WebSocket Protokoll angeboten, doch ist es, ähnlich zu den in *Abschnitt 5.1 Grundlegende Überlegungen* beschriebenen Limitierungen, nicht möglich eine WebSockets-Bibliothek aus Unity direkt als Bibliothek für die WebGL Anwendung zu verwenden. Deshalb empfiehlt es sich, eine einheitliche, gemeinsame Schnittstelle während der Entwicklung zu definieren, die zur Laufzeit die jeweiligen nativen Implementierungen verwendet (Siehe Abbildung 5.6). Mithilfe eines sogenannten .jslib Plugins kann, wie von Unity Technologies (2018b) beschrieben, der Code der nativen Implementierungen innerhalb einer abstrakten gemeinsamen Codebasis zusammengeführt werden.

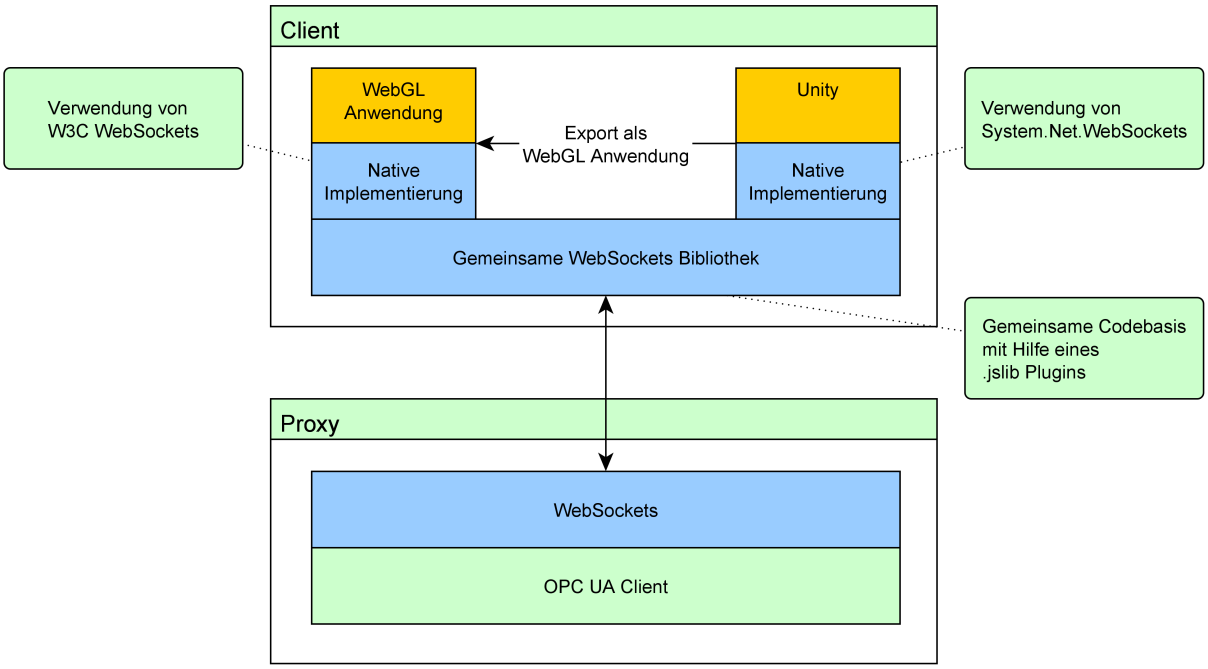


**Abbildung 5.6:** Konzept zur Verwendung einer einheitlichen WebSockets Bibliothek.

### 5.2.4 Konzeption FA-4

Innerhalb von Unity können GameObjects hierarchisch ineinander verschachtelt und mit zusätzlichen Informationen angereichert werden. Damit ist es möglich, eine ähnliche Struktur wie auf einem OPC UA Server zu erstellen (Siehe Abbildung 5.7). Somit müssen innerhalb von Unity keine weiteren externen Komponenten eingebunden werden, um mit der Entwicklung der Anwendung starten zu können. Nach der Konfiguration der GameObjects sollen diese in der Lage sein, Daten mithilfe des Proxys auszutauschen und der Anwendung aktuelle Daten zur Verfügung zu stellen. Mithilfe einer Untergliederung in mehreren hierarchisch geordneten Ebenen sind somit

auch mehrere an einem Proxy angebundene OPC UA Server vorstellbar. Die neu definierten GameObjects, die mit den OPC UA Nodes synchronisiert werden, können dann anschließend mithilfe von Drag-and-Drop in anderen GameObjects referenziert werden. Zum Beispiel können so verschiedene in Unity verwendete 3D-Modelle mit den aktuellen Daten des referenzierten GameObjects gleichzeitig angereichert werden.

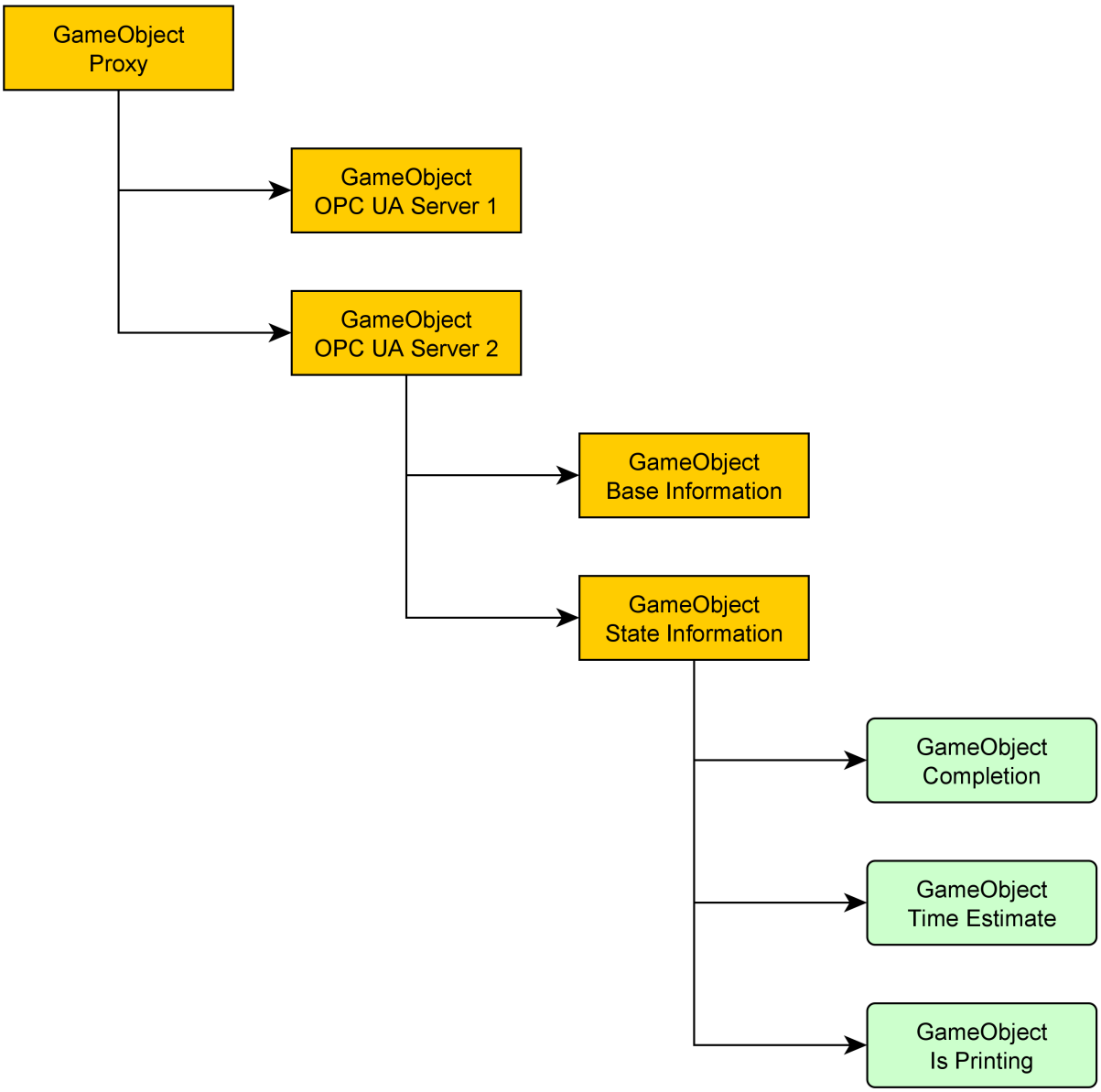


**Abbildung 5.7:** Konzept zur Verwendung von GameObjects als Äquivalent eingebundener OPC UA Nodes.

## 5.3 Fazit

In diesem Kapitel wurde im *Abschnitt 5.1 Grundlegende Überlegungen* festgestellt, dass eine direkte Verbindung über den Standard OPC UA zwischen dem Client und dem OPC UA Server aufgrund technischer Restriktionen nicht möglich ist. Deshalb wurde im *Abschnitt 5.2 Proxy zwischen OPC UA und WebSockets* ein Konzept entworfen, mithilfe dem es möglich ist, eine persistente Verbindung zur Kommunikation zwischen der WebGL Anwendung und dem OPC UA Server zu etablieren. Dabei wird eine Schnittstelle implementiert, die die Kommunikation zwischen WebSockets und OPC UA verwalten soll. Im *Kapitel 6 Implementierung* wird dieses Konzept umgesetzt und auf auftretende Probleme und Lösungen eingegangen.

# Kapitel 6

# Implementierung

Anhand des im vorherigen Kapitel erstellten Konzeptes wird in den nächsten Abschnitten eine konkrete Lösung vorgestellt. Zuallererst wird im *Abschnitt 6.1 OPC UA Server* die erste Komponente des Systems erstellt. Anschließend wird im *Abschnitt 6.2 Proxy* auf Probleme und Lösungen der Umsetzung der Schnittstelle zwischen den beiden Protokollen eingegangen. Im *Abschnitt 6.3 WebGL Anwendung* wird die Umsetzung der dritten Komponente, die für die Vollständigkeit des entworfenen Konzeptes benötigt wird, vorgestellt. In diesem Abschnitt wird die Kommunikation der WebGL Anwendung mit dem Proxy beschrieben.

## 6.1 OPC UA Server

Für die Entwicklung des Systems wird ein OPC UA Server benötigt, der in diesem Abschnitt beschrieben wird. Mithilfe dieser Komponente sollen Werte einer Maschine simuliert werden. Ich habe mich in meiner Abschlussarbeit dazu entschlossen, einen 3D-Drucker zu simulieren. Dieser soll den aktuellen Druckfortschritt, die geschätzte Druckdauer, den Modellnamen sowie die derzeitige Druckaktivität zurückliefern. Die WebGL Anwendung soll nach Fertigstellung des Systems in der Lage sein, eine Simulation eines Drucks zu starten und einen digitalen Zwilling der Maschine abzubilden.

### 6.1.1 Struktur

Die Struktur des OPC UA Servers inklusive der verwendeten Datentypen ist in Abbildung 6.1 ersichtlich. Der Node Object dient für die untergeordneten Nodes als Wurzelknoten. Diese repräsentieren, wie in OPC Foundation (2017d) beschrieben, die Komponenten eines Hardware Systems.

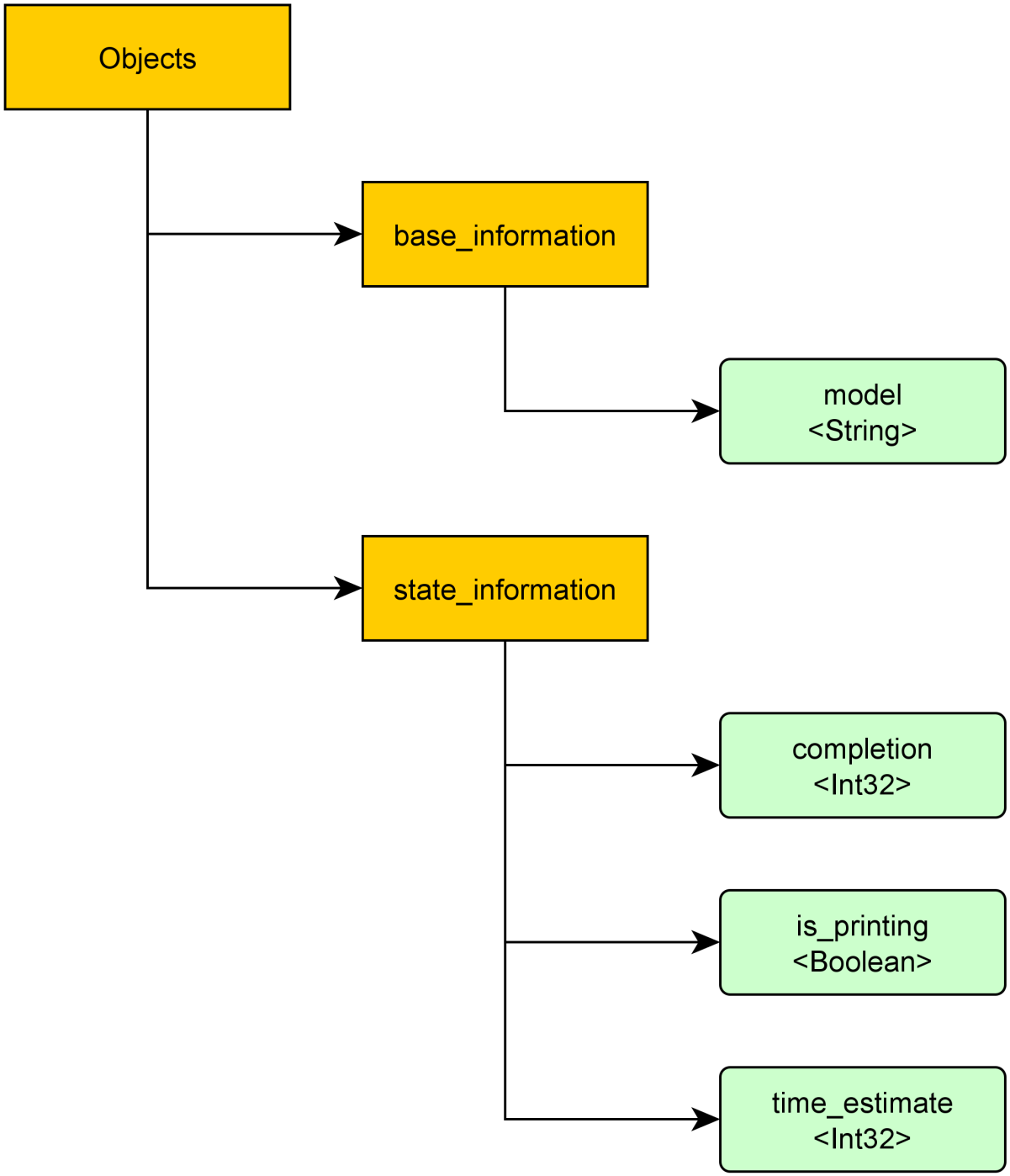


**Abbildung 6.1:** Simulierte Struktur des OPC UA Servers eines 3D-Druckers.

### 6.1.2 Simulation des digitalen Zwillings

Für die Generierung von konkreten Anwendungsfällen habe ich für die einzelnen Anforderungen, die in den vorherigen Kapiteln formuliert wurden, Szenarien entwickelt, die mit dieser Simulation abbildbar sind. Beispielsweise soll der Wert des Nodes *model* einmalig, wie in FA-1 gefordert, abgerufen werden können. Eine Änderung des Modells während der Laufzeit ist nicht zu erwarten. Das Starten eines Druckvorgangs mithilfe des Clients beeinflusst den Node *is_printing*. Durch die Etablierung jeweils einer Subscription auf die Nodes *completion* und *time_estimate* werden, wie in FA-2 beschrieben, die aktuellen Werte der Simulation nach einer Statusänderung, wie im *Kapitel 5 Konzept* vorgestellt, an den Proxy übertragen. Nach der Übertragung der Daten an den Proxy werden diese weiter an die WebGL Anwendung übermittelt.

Die Implementierung des OPC UA Servers wurde, wie von Oroulet, Heine und Thephez (2015) beschrieben, mithilfe der auf Python basierenden Bibliothek python-opcua durchgeführt. Mit dieser wurde die im vorherigen Abschnitt definierte Struktur abgebildet (Siehe Listing 6.1).

```
1  from opcua import ua, Server
2
3  if __name__ == "__main__":
4      # Server erstellen
5      server = Server()
6
7      # Namensraum registrieren
8      ns = "http://www.oth-aw.de"
9      idx = server.register_namespace(ns)
10
11     # Objektknoten auslesen
12     objects = server.get_objects_node()
13
14     # OPC UA Nodes erstellen
15     base = objects.add_object(idx, "base_information")
16     state = objects.add_object(idx, "state_information")
17
18     # OPC UA Nodes erstellen (base)
19     model = base.add_variable(idx, "model", "Drucker 1")
20     # OPC UA Nodes erstellen (state)
21     completion = state.add_variable(idx, "completion", 0)
22     is_printing = state.add_variable(idx, "is_printing", False)
23     time_estimate = state.add_variable(idx, "time_estimate", 0)
24
25     # Schreibschutz des Nodes für Clients aufheben
26     is_printing.set_writable()
27
28     # Server starten
29     server.start()
```

**Listing 6.1:** Erstellung der OPC UA Serverstruktur mit python-opcua.

Mithilfe der in Listing 6.1 durchgeführten Implementierung ist es möglich, den Server zu starten und auf diesen zuzugreifen. Dabei enthalten die Nodes die definierten Standardwerte. Um einen Druckauftrag starten zu können, wurde der Server um eine Routine erweitert, die innerhalb eines Intervalls die Werte der einzelnen Nodes aktualisiert. Zur Demonstration wurde innerhalb dieser Abschlussarbeit exemplarisch eine Simulation, basierend auf der aktuellen Druckaktivität, implementiert (Siehe Listing 6.2).

```
1  # Simulation eines Druckauftrages
2  current_completion = 0
3  current_time_estimate = 200
4
5  while True:
6      if is_printing.get_value() == True and current_completion <
           100:
7          # Druckauftrag gestartet
8          current_completion += 1
9          current_time_estimate -= 2
10     else:
11         # Druckauftrag abgeschlossen
12         current_completion = 0
13         current_time_estimate = 200
14         is_printing.set_value(False)
15
16     # Werte der Nodes aktualisieren
17     completion.set_value(ua.DataValue(ua.Variant(
          current_completion, ua.VariantType.Int32)))
18     time_estimate.set_value(ua.DataValue(ua.Variant(
          current_time_estimate, ua.VariantType.Int32)))
19
20     time.sleep(0.03)
```

**Listing 6.2:** Simulation der aktuellen Werte der Nodes.

## 6.2 Proxy

In diesem Abschnitt wird die Schnittstelle zwischen den beiden Protokollen WebSocket und OPC UA erstellt. Aufbauend auf die im *Abschnitt 6.1 OPC UA Server* durchgeführte Implementierung wird eine Komponente entwickelt, die, wie im *Kapitel 5 Konzept* vorgestellt, für die bidirektionale Kommunikation zwischen der WebGL Anwendung und dem OPC UA Server verwendet wird.

### 6.2.1 Anbindung an den OPC UA Server

Der Proxy wurde mit dem auf JavaScript basierenden Framework Node.js entwickelt. Mithilfe dieses Frameworks ist es möglich, einen Webserver zu betreiben. In dieser Abschlussarbeit werden für die Anbindung an den OPC UA Server die Bibliotheken

node-opcua und async verwendet. Mit node-opcua kann zum Beispiel eine Verbindung zum OPC UA Server aufgebaut oder ein Wert eines Nodes ausgelesen werden. Async erleichtert die Implementierung der asynchron zusammenhängenden Methoden. Im Listing 6.3 wird ein Beispiel zum Aufbau einer Session zu einem OPC UA Server mit den beiden Bibliotheken demonstriert.

```
1  const opcua = require("node-opcua");
2  const async = require("async");
3
4  client = opcua.OPCUAClient.create();
5  endpoint = "opc.tcp://127.0.0.1:4840/opcua/server"
6  session = undefined
7
8  async.series([
9      // Zum OPC UA Server verbinden
10     function (callback) {
11         client.connect(endpoint, function (err) {
12             callback(err);
13         });
14     },
15     // Eine Session zu diesem OPC UA Server aufbauen,
16     // nachdem eine Verbindung zum Server hergestellt wurde
17     function (callback) {
18         client.createSession(function (err, s) {
19             if (!err) {
20                 // Session wurde aufgebaut und kann jetzt
21                 // weiterverwendet werden
22                 session = s;
23             }
24
25             callback(err);
26         });
27     }
28     ],
29     // Fehler innerhalb der Serie aufgetreten
30     function (err) {
31         if (err) {
32             console.log("Error: ", err);
33         }
34     }
35 )
```

**Listing 6.3:** Aufbau einer Session zu einem OPC UA Server.

Um den aktuellen Wert eines Nodes auslesen zu können, muss, wie in Listing 6.3 gezeigt, eine Session hergestellt werden. Diese kann anschließend verwendet werden, um mithilfe eines Callbacks den aktuellen Status auszulesen (Siehe Listing 6.4).

```
1  // Liest den aktuellen Wert des Nodes aus.
2  // Die Antwort findet mithilfe des übergebenen
3  // Callbacks statt.
4  function readNode (nodeId, callback) {
5      let nodeToRead = {
6          nodeId: nodeId,
7          attributeId: opcua.AttributeIds.Value
8      }
9
10     // Zugriff auf etablierte Session
11     session.read(nodeToRead, callback)
12 }
13
14 // Beispielhafte Verwendung
15 nodeId = "ns=2;i=3"
16
17 readNode (nodeId, (node, dataValue) => {
18     // Gibt den aktuellen Wert des Nodes bei Erhalt
19     // der Antwort aus
20     console.log(`${node} - ${dataValue}`)
21 })
```

**Listing 6.4:** Auslesen des aktuellen Wertes eines Nodes.

Für die Etablierung einer Subscription, wie in FA-2 vorgestellt, muss zuerst eine neue Subscription erstellt werden. Mit dieser Subscription kann sich ein Client beim OPC UA Server für Nodes registrieren, bei denen er bei einer Statusänderung informiert werden soll. Mit dem Callback, den der Client bei der Registrierung hinterlegt hat, können die aktuellen Werte der Nodes übertragen werden (Siehe Listing 6.5). Der Client kann nur bei einer noch geöffneten Verbindung mit dem OPC UA Server benachrichtigt werden.

```
1  // Erstellt eine neue Subscription zum OPC UA Server für die
2  // übergebenen Nodes. Der Client wird bei einer
3  // Änderung eines Wertes des Nodes mithilfe des
4  // übergebenen Callbacks benachrichtigt
5  function monitorNodes (nodes, onDataChange) {
6      // Subscription erstellen
7      let subscription = opcua.ClientSubscription.create(session,
           subscriptionConfig);
8
9      // Nodes überwachen
10     nodes.forEach(node => {
11         const itemToMonitor = opcua.ReadValueIdLike = {
12             nodeId: opcua.resolveNodeId(node),
13             attributeId: opcua.AttributeIds.Value
14         };
15
16         const monitoredItem = opcua.ClientMonitoredItem.create(
17             subscription,
18             itemToMonitor,
19             monitoredItemConfig,
20             opcua.TimestampsToReturn.Both
21         );
22
23         // Callback, das aufgerufen werden soll,
24         // wenn sich der Wert eines Nodes ändert.
25         monitoredItem.on("changed", onDataChange);
26     });
27
28     return subscription;
29 }
30
31 // Beispielhafte Verwendung
32 subscribeNodes = ["ns=2;i=4"]
33
34 monitorNodes (subscribeNodes, dataValue => {
35     // Gibt den aktuellen Wert des Nodes bei Erhalt
36     // der Antwort aus
37     console.log(dataValue)
38 })
```

**Listing 6.5:** Erstellen einer Subscription für zu überwachende Nodes.

### 6.2.2 Behandlung der eingehenden WebSocket-Anfragen

Um der WebGL Anwendung die Möglichkeit anzubieten, eine WebSocket-Verbindung mit dem Proxy herstellen zu können, muss ein WebSocket am Server geöffnet werden. Dazu wurde in dieser Abschlussarbeit die Bibliothek ws, wie von Pinca, Stangvik und Kazemier (2011) beschrieben, verwendet. Nachdem ein WebSocket an einem Port erzeugt worden ist, kann auf eingehende Anfragen der WebGL Anwendung reagiert werden.

Die Kommunikation der WebGL Anwendung mit dem Proxy findet mithilfe von JSON, einem lesbaren Datenformat, statt. Dazu sind Anfragen definiert worden, mit denen die Anforderungen FA-1 und FA-2 implementiert werden können. Diese bestehen aus der durchzuführenden Aktion und dem OPC UA Server, an den die zusätzlichen Daten der Anfrage weitergeleitet werden (Siehe Listing 6.6). Beim Setzen eines Wertes muss außerdem der Datentyp und Wert übermittelt werden.

```
1  // Liest den Wert eines Nodes aus
2  // FA-1
3  {
4      "action" : "GetValue",
5      "server" : "opc.tcp://127.0.0.1:4840/opcua/server",
6      "node" : "ns=2;i=3",
7      "dataType" : "",
8      "value" : ""
9  }
10
11 // Setzt den Wert eines Nodes
12 // FA-1
13 {
14     "action" : "SetValue",
15     "server" : "opc.tcp://127.0.0.1:4840/opcua/server",
16     "node" : "ns=2;i=5",
17     "dataType" : "Boolean",
18     "value" : "true"
19 }
20
21 // Erstellt eine Subscription
22 // FA-2
23 {
24     "action" : "CreateSubscription",
25     "server" : "opc.tcp://127.0.0.1:4840/opcua/server",
26     "node" : "ns=2;i=6",
27     "dataType": "",
28     "value" : ""
29 }
```

**Listing 6.6:** Aufbau der Anfragen an den Proxy.

Erreicht eine Anfrage die WebSocket-Schnittstelle des Proxys, wird diese verarbeitet. Durch die Persistenz der aufgebauten Verbindung zwischen der WebGL Anwendung

und dem Proxy kann, sobald die Daten vom OPC UA Server erhalten wurden, die Antwort zurückübertragen werden. Die Verknüpfung der eingehenden WebSocket-Anfragen mit dem OPC UA Server kann in Listing 6.7 nachvollzogen werden. Sobald eine geöffnete Session und die entsprechend durchzuführende Aktion, wie dies ab Zeile 20 der Fall ist, vorliegt, kann die Kommunikation wie in *Unterabschnitt 6.2.1 Anbindung an den OPC UA Server* beschrieben, durchgeführt werden. Im Callback wird schlussendlich das Ergebnis mit der Methode ws.send() an die WebGL Anwendung übertragen.

```
1  // Einbinden der WebSocket-Bibliothek
2  const WebSocket = require('ws');
3  // Eigene Bibliothek zur Verwaltung von geöffneten Sessions,
4  // um bereits geöffnete Sessions wiederverwenden zu können.
5  const sessionManager = require('./opcua/clientSessionManager');
6
7  // Initialisierung
8  const wss = new WebSocket.Server({ port: 8089 });
9  const clients = new sessionManager.ClientSessionManager()
10
11 wss.on('connection', function connection(ws) {
12     // WebGL-Anwendung hat sich zum Proxy verbunden
13     ws.on('message', function incoming(data) {
14         // Es wurde eine neue Nachricht erhalten
15         // Daten der Anfrage auslesen
16         let request = JSON.parse(data);
17
18         // Session zum OPC UA Server etablieren
19         clients.useClientConnection(request.server, (client) =>
               {
20             if (request.action === "CreateSubscription") {
21                 // ...
22                 // Kommunikation mit OPC UA Server
23                 // ...
24                 // ws.send(result) sendet im Callback
25                 // schlussendlich eine Antwort an
26                 // den Client zurück
27             } else if (request.action === "SetValue") {
28                 // ...
29             } else if (request.action === "GetValue") {
30                 // ...
31             }
32         })
33     });
34 });
```

**Listing 6.7:** Weiterleiten der WebSocket-Anfragen an den OPC UA Server.

## 6.3 WebGL Anwendung

Teil dieses Abschnittes ist die Entwicklung der WebGL Anwendung in Unity. Mithilfe der Kommunikation mit dem Proxy werden aktuelle Werte des OPC UA Servers für den digitalen Zwilling übertragen. Dabei werden zuerst die technischen Details zur Verbindung zum Proxy erläutert. Anschließend wird beschrieben, wie diese vom Entwickler der WebGL Anwendung verwendet werden können.

Zuallererst wird auf die Kommunikation des Clients mit dem Proxy eingegangen. Dabei weicht die Architektur aufgrund der verwendeten Zielplattform ab. Während der Entwicklungszeit wird die Anwendung innerhalb von Unity auf der Architektur des Hosts ausgeführt. Nach dem Export zu einer WebGL Anwendung wird sie innerhalb eines Browsers ausgeführt. Dies führt zu Unterschieden in der zur Verfügung stehenden Bibliotheken, mit denen WebSockets implementiert werden können. Während im Browser die Bibliothek W3C WebSockets zur Verfügung steht, kann in Unity beispielsweise System.Net.WebSockets verwendet werden (Siehe *Unterabschnitt 5.2.3 Konzeption FA-3*).

Um eine einheitliche Schnittstelle in Unity verwenden zu können, kann ein sogenanntes Interface, das die Eigenschaften und Methoden einer Klasse beschreibt, verwendet werden. Dadurch kann, abhängig von der Architektur, die jeweils benötigte Bibliothek eingesetzt werden. Mithilfe der von Unity Technologies (2018a) beschriebenen plattformabhängigen Kompilation können unterschiedliche Codefragmente benutzt werden (Siehe Listing 6.8).

```
1  interface IWebSocket {
2      // Eigenschaften und Methoden,
3      // die ein WebSocket besitzt
4  }
5
6  // Plattformabhängige Codefragmente, die bei
7  // der Kompilation verwendet werden.
8  #if UNITY_WEBGL && !UNITY_EDITOR
9      // Verwendete Codefragmente im Browser
10     class WebSocket : IWebSocket() {
11         // Bei der Implementierung werden
12         // W3C WebSockets
13         // verwendet.
14     }
15 #else
16     // Verwendete Codefragmente auf der
17     // Host-Architektur
18     class WebSocket : IWebSocket() {
19         // Bei der Implementierung werden
20         // System.Net.WebSockets
21         // verwendet.
22     }
23 #endif
24
```

```
25 // Neues Objekt der Klasse WebSocket.
26 // Es stehen die Eigenschaften und Methoden des Interface
27 // IWebSocket unabhängig von der Architektur zur Verfügung.
28 WebSocket ws = new WebSocket()
```

**Listing 6.8:** Verwendung von plattformabhängigen Code in Unity.

Basierend auf diesem Prinzip ist es möglich, eine wie in FA-3 geforderte Anbindung an den OPC UA Server während der Entwicklungszeit umzusetzen. In dieser Abschlussarbeit wurde dafür die Bibliothek NativeWebSockets verwendet. Mithilfe dieser ist es, wie von Dreyer, Bogin und Ukhanov (2019) beschrieben, möglich, WebSockets für die verschiedenen von Unity unterstützten Zielplattformen zu implementieren.

In Unity wurde für die Verwendung eines OPC UA Nodes ein GameObject angelegt, das mit einem innerhalb der Abschlussarbeit erstellten Skript, dem NodeTwinWebSocket, erweitert wurde. Dieses dient zur Verknüpfung des GameObjects mit dem verwendeten OPC UA Node des angegebenen OPC UA Servers. Dadurch kann der aktuelle Wert ausgelesen und gesetzt werden (Siehe FA-1). Zusätzlich kann das GameObject bei einer Änderung des OPC UA Nodes automatisch benachrichtigt werden (Siehe FA-2). Für die Referenzierung des GameObjects innerhalb von Unity ist es möglich, wie in Abbildung 6.2 gezeigt, Drag-and-Drop zu verwenden (Siehe FA-4).

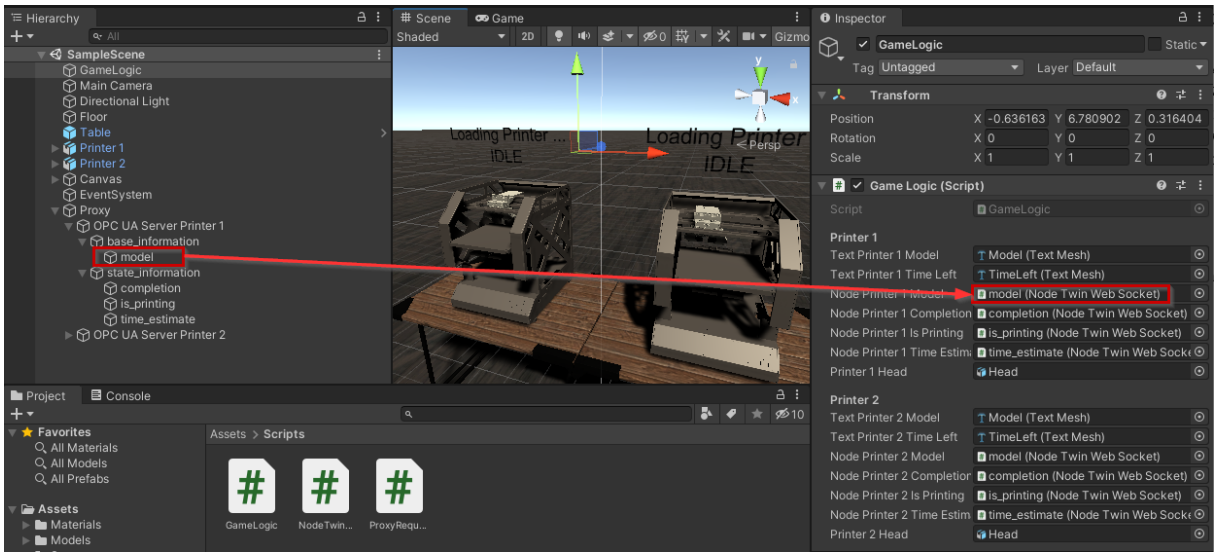


**Abbildung 6.2:** Referenzieren der GameObjects mithilfe von Drag-and-Drop in Unity.

Die Konfiguration der einzelnen GameObjects zur Anbindung an den Proxy sowie dem OPC UA Server erfolgt mithilfe von Unity (Siehe Abbildung 6.3). Dabei kann außerdem angegeben werden, ob beim Start der Anwendung automatisch eine Subscription erstellt werden soll. Der aktuelle Wert ist anschließend, sobald eine valide Antwort erhalten wird, für die weitere Verwendung verfügbar (Siehe Listing 6.9).

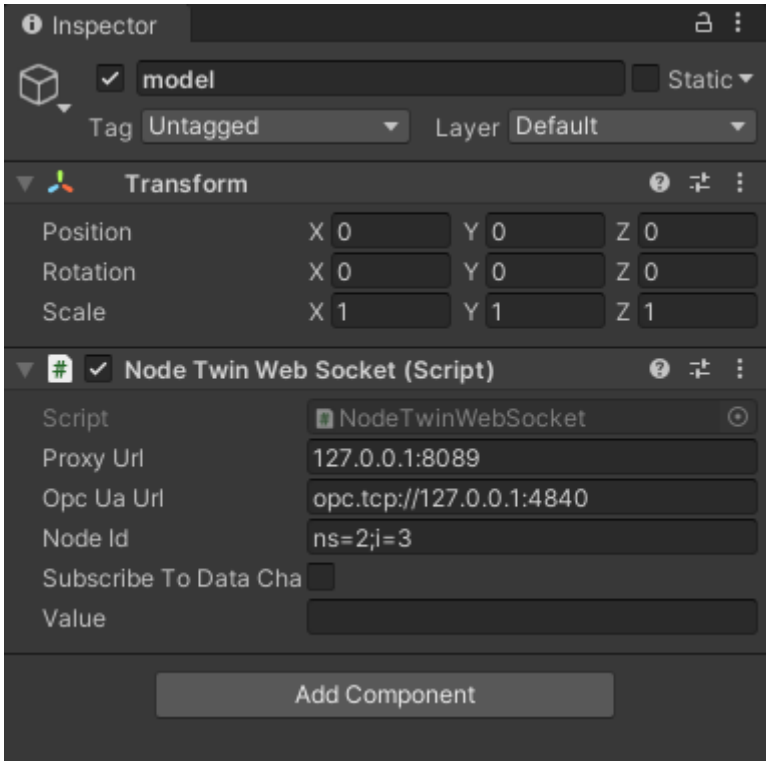


**Abbildung 6.3:** Konfiguration des erstellten Skripts eines GameObjects.

```
1  // GameObject, das mit einem OPC UA Node verknüpft wurde
2  public NodeTwinWebSocket model;
3
4  void Update()
5  {
6      // Prüfe, ob ein gültiges Objekt referenziert wurde
7      if (model != null) {
8          // Ausgabe des aktuellen Wertes auf die Konsole
9          Debug.Log(model.Value);
10     }
11 }
```

**Listing 6.9:** Zugriff auf den aktuellen Wert des GameObjects.

# Kapitel 7

# Evaluierung

In diesem Kapitel wird die im *Kapitel 6 Implementierung* entwickelte Lösung evaluiert. Dazu wird diese mit den im *Kapitel 4 Anforderungsanalyse* formulierten Anforderungen im *Abschnitt 7.1 Auswertung der implementierten Anforderungen* verglichen. Anschließend wird im *Abschnitt 7.2 Fazit* ein Fazit gezogen.

## 7.1 Auswertung der implementierten Anforderungen

Innerhalb dieses Abschnittes wird auf die einzelnen Anforderungen eingegangen und der jeweilige Soll-Zustand mit dem Ist-Zustand verglichen. Dabei wird außerdem auf aufgetretene Probleme hingewiesen.

FA-1 beschreibt die Kommunikation zwischen einer WebGL Anwendung und einem OPC UA Server. Das System konnte, wie im *Kapitel 5 Konzept* beschrieben, mithilfe einer Schnittstelle zwischen den beiden Protokollen OPC UA und WebSockets, implementiert werden. Das Übersetzen der Anfragen in das jeweils andere Protokoll wurde mit einem Proxy umgesetzt.

Es gilt zu beachten, dass die in der Implementierung übermittelten Datentypen in den einzelnen Komponenten des Systems verfügbar sein müssen. Ist dies nicht der Fall, so müssen sie durch Äquivalente ersetzt werden. Außerdem wird eine entsprechende Umwandlung der verschiedenen Datentypen zur Laufzeit notwendig.

Durch die Persistenz der Verbindungen ist es dem Server möglich, zeitverzögert Antworten zu übertragen. Dadurch kann sich der Client für die Benachrichtigung zukünftiger Statusänderungen bei Maschinen, wie in FA-2 formuliert, registrieren. Die WebGL Anwendung kann eine Anfrage, wie in *Unterabschnitt 6.2.2 Behandlung der eingehenden WebSocket-Anfragen* implementiert, zum Proxy übertragen, sodass dieser eine Subscription zum OPC UA Server erstellt. Mithilfe dieser Subscription wird der Proxy bei jeder Änderung vom OPC UA Server informiert und kann die Mitteilung über den geänderten Wert dem registrierten Client weiterleiten.

Unity bietet die Möglichkeit, abhängig von der verwendeten Zielplattform, unter-

schiedliche Codefragmente zu verwenden. Dies wurde für die Umsetzung der FA-3 verwendet. Dadurch kann für die Ausführung innerhalb einer WebGL Anwendung anderer Quellcode als für andere Zielplattformen definiert werden. Mithilfe eines Interfaces, wie in *Abschnitt 6.3 WebGL Anwendung* beschrieben, kann dennoch eine einheitliche Schnittstelle entwickelt werden. Dadurch kann die Anwendung komplett in Unity entwickelt werden und muss erst für die Verwendung im Browser als WebGL Anwendung exportiert werden.

Mit dem entwickelten Proof of Concept kann innerhalb von Unity ein GameObject, das mit einem OPC UA Node verknüpft wurde, wie in FA-4 beschrieben, mit Drag-and-Drop referenziert werden. Außerdem kann, wie im *Abschnitt 6.3 WebGL Anwendung* gezeigt wurde, die Anbindung zum OPC UA Node ohne Änderung im Quellcode innerhalb von Unity konfiguriert werden. Dadurch ist es dem Entwickler der Anwendung möglich, auf Änderungen innerhalb des OPC UA Servers zu reagieren.

## 7.2 Fazit

Mithilfe des im *Kapitel 5 Konzept* ausgearbeiteten Konzeptes sind alle im *Kapitel 4 Anforderungsanalyse* formulierten Anforderungen umsetzbar. Innerhalb des *Kapitel 6 Implementierung* konnte der Proof of Concept innerhalb meiner Abschlussarbeit umgesetzt werden (Siehe Abbildung 7.1). Durch die Einführung des Proxys als zusätzliche Komponente innerhalb des Systems müssen eventuell auftretende Inkompatibilitäten von Datentypen beachtet werden.

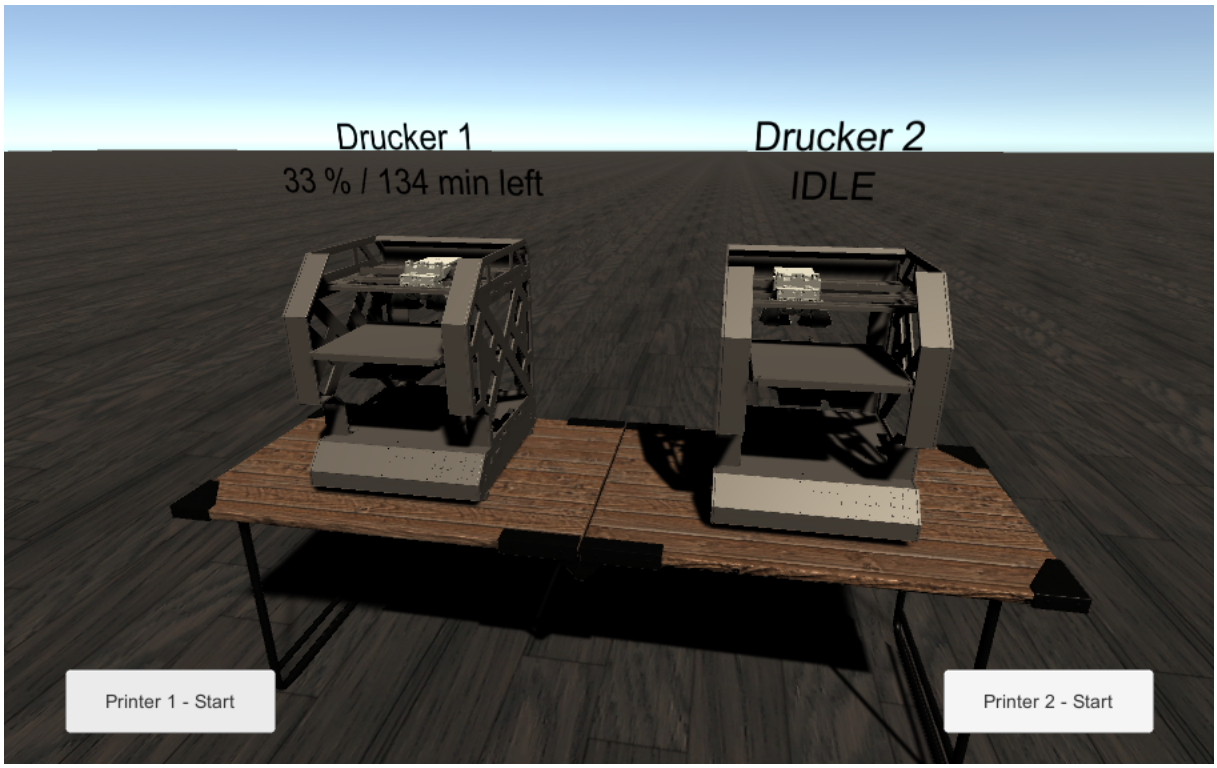


**Abbildung 7.1:** Implementierung des Proof of Concept.

# Kapitel 8

# Zusammenfassung und Ausblick

Innerhalb dieser Abschlussarbeit wurde nach einer Möglichkeit gesucht, einen digitalen Zwilling zu entwickeln, der in einem Browser visualisierbar ist. Dieser wird an einen OPC UA Server angebunden, um aktuelle Werte der zu repräsentierenden Maschine zu erhalten. Mithilfe der Entwicklungsumgebung Unity kann ein 3D-Modell einer Maschine als WebGL Anwendung exportiert und im Browser ausgeführt werden. Die Anbindung dieser exportierten Anwendung an einen OPC UA Server ist nach dem derzeitigen Stand der Forschung nur bedingt möglich. Zwar gibt es Versuche, aktuelle Daten einzubinden, doch müssen diese erst innerhalb einer Datenbank zwischengespeichert werden. Zusätzlich musste festgestellt werden, dass viele Produkte nur proprietär verfügbar und die verwendeten Lösungswege damit nicht nachvollziehbar sind.

Deshalb wurde, nach der Definition der Anforderungen an einen Proof of Concept, die Anbindung einer WebGL Anwendung an einen OPC UA Server in den Vordergrund gerückt. Dabei soll jedoch nicht die Möglichkeit der Entwicklung einer Anwendung für verschiedene Zielplattformen in Unity verloren gehen.

Während der Konzeptionierung ist festgestellt worden, dass es zum derzeitigen Zeitpunkt nicht möglich ist, eine WebGL Anwendung direkt mit einem OPC UA Server zu verbinden. Deshalb wurde schlussendlich ein Konzept entworfen, indem eine Schnittstelle zwischen der WebGL Anwendung und dem OPC UA Server eingeführt wird. Diese Schnittstelle, der Proxy, ist für die Kommunikation mit dem OPC UA Server und der Verbindung mit dem Client, die mithilfe von WebSockets implementiert wird, zuständig. Durch die mit WebSockets erreichbare bidirektionale und persistente Verbindung ist es möglich, Anfragen der WebGL Anwendung über die Schnittstelle an den OPC UA Server zu übertragen. Außerdem kann auch der OPC UA Server Antworten an den Client weiterleiten lassen. Dadurch werden schlussendlich die Etablierungen von Subscriptions möglich. Eine Zwischenspeicherung von Maschinendaten innerhalb von Datenbanken ist in dieser umgesetzten Lösung nicht notwendig.

Bei der Entwicklung der Anwendung ist im Proof of Concept die Möglichkeit umgesetzt worden, OPC UA Nodes mit GameObjects zu verknüpfen. Dadurch kann in Unity mithilfe eines GameObjects ein OPC UA Node abgebildet werden. Mit dieser Abbildung können Werte aus einem OPC UA Server ausgelesen oder gesetzt wer-

den. Die Konfiguration eines OPC UA Nodes wurde in dieser Abschlussarbeit noch manuell durchgeführt. Vorstellbar wäre in Zukunft, dass dem Entwickler der Anwendung existierende an den Proxy angebundene OPC UA Server und OPC UA Nodes vorgeschlagen oder automatisch in Unity importiert werden.

Im Zuge der Abschlussarbeit wurde auch die Möglichkeit in Betracht gezogen, existierende OPC UA Clients in ein WebAssembly-Modul zu konvertieren, um dieses anschließend in der WebGL Anwendung verwenden zu können. Aufgrund technischer Restriktionen innerhalb der Ausführung in Browsern ist dies nach derzeitigem Stand noch nicht möglich. Sollten sich dabei Änderungen ergeben, wäre das ein guter Ansatzpunkt zur direkten Anbindung einer WebGL Anwendung an einen OPC UA Server über den Standard OPC UA.

# Literaturverzeichnis

Bök, P.-B., Noack, A., Müller, M. & Behnke, D. (Hrsg.). (2020). *Computernetze und Internet of Things*. Wiesbaden: Springer Fachmedien Wiesbaden. doi: 10.1007/978-3-658-29409-0

Bühler, P., Schlaich, P. & Sinner, D. (2017). *HTML5 und CSS3*. Berlin, Heidelberg: Springer. doi: 10.1007/978-3-662-53916-3

Bundesministerium für Bildung und Forschung. (2020). *Industrie 4.0.* Zugriff am 26.11.2020 auf `https://www.bmbf.de/de/zukunftsprojekt-industrie-4-0-848.html`

Bundesministerium für Wirtschaft und Energie. (2018). *Sichere Implementierung von OPC UA für Betreiber, Integratoren und Hersteller.* Zugriff am 26.11.2020 auf `https://www.plattform-i40.de/PI40/Redaktion/DE/Downloads/Publikation/hm-2018-opc.pdf?__blob=publicationFile&v=6`

Dreyer, E., Bogin, T. & Ukhanov, B. (2019). *Native WebSockets.* Zugriff am 03.03.2021 auf `https://github.com/endel/NativeWebSocket/blob/master/README.md`

Echterhoff, J. (2015). *Unity 5.3 WebGL Updates.* Zugriff am 23.02.2021 auf `https://blogs.unity3d.com/2015/12/07/unity-5-3-webgl-updates/`

Fras, K. & Nowak, Z. (2020). WebAssembly – Hope for Fast Acceleration of Web Applications Using JavaScript. In L. Borzemski, J. Świątek & Z. Wilimowska (Hrsg.), *Information Systems Architecture and Technology: Proceedings of 40th Anniversary International Conference on Information Systems Architecture and Technology – ISAT 2019* (Bd. 1050, S. 275–284). Cham: Springer International Publishing. doi: 10.1007/978-3-030-30440-9

Freund, M., Martin, C., Braune, A. & Steinkrauss, U. (2013). JSUA — An OPC UA JavaScript framework. In *2013 IEEE 18th Conference on Emerging Technologies & Factory Automation (ETFA)* (S. 1–4). IEEE. doi: 10.1109/ETFA.2013.6648128

Gorski, P. L., Lo Iacono, L. & Nguyen, H. V. (2015). *WebSockets: Moderne HTML5-Echtzeitanwendungen entwickeln*. München: CARL HANSER Verlag.

Hardman, C. (2020). *Game Programming with Unity and C#: A Complete Beginner's Guide*. Berkeley, CA: Apress.

Hennig, S., Braune, A. & Damm, M. (2010). JasUA: A JavaScript Stack enabling web browsers to support OPC Unified Architecture's Binary mapping natively. In *2010 IEEE 15th Conference on Emerging Technologies & Factory Automation (ETFA 2010)* (S. 1–4). IEEE. doi: 10.1109/ETFA.2010.5641005

Olbort, J., Herden, H., Kutscher, V. & Anderl, R. (2020). Mixed Reality for Visualization of Operating Data and Semantic Self-Descriptions of Machines using OPC UA. In B. Mrugalska, S. Trzcielinski, W. Karwowski, M. Di Nicolantonio & E. Rossi

(Hrsg.), *Advances in Manufacturing, Production Management and Process Control* (Bd. 1216, S. 149–156). Cham: Springer International Publishing. doi: 10.1007/978-3-030-51981-0

OPC Foundation. (2017a). *OPC Unified Architecture, Part 1: Overview and Concepts.* Zugriff am 29.01.2021 auf https://opcfoundation.org/developer-tools/specifications-unified-architecture/part-1-overview-and-concepts/

OPC Foundation. (2017b). *OPC Unified Architecture, Part 3: Address Space Model.* Zugriff am 29.01.2021 auf https://opcfoundation.org/developer-tools/specifications-unified-architecture/part-3-address-space-model/

OPC Foundation. (2017c). *OPC Unified Architecture, Part 4: Services.* Zugriff am 29.01.2021 auf https://opcfoundation.org/developer-tools/specifications-unified-architecture/part-4-services/

OPC Foundation. (2017d). *OPC Unified Architecture, Part 5: Information Model.* Zugriff am 25.02.2021 auf https://opcfoundation.org/developer-tools/specifications-unified-architecture/part-5-information-model/

OPC Foundation. (2017e). *OPC Unified Architecture, Part 7: Profiles.* Zugriff am 29.01.2021 auf https://opcfoundation.org/developer-tools/specifications-unified-architecture/part-7-profiles/

Oroulet, Heine, A. & Thephez. (2015). *python-opcua.* Zugriff am 12.3.2021 auf https://github.com/FreeOpcUa/python-opcua/blob/master/README.md

Paronen, T. (2015). *A web-based monitoring system for the Industrial Internet* (Master's thesis). Aalto University, Espoo.

Pinca, L., Stangvik, E. O. & Kazemier, A. (2011). *ws: a Node.js WebSocket library.* Zugriff am 12.3.2021 auf https://github.com/websockets/ws/blob/master/README.md

Rourke, M. (2018). *Learn WebAssembly*. Birmingham: Packt Publishing.

Schiekofer, R., Scholz, A. & Weyrich, M. (2018). REST based OPC UA for the IIoT. In *2018 IEEE 23rd International Conference on Emerging Technologies and Factory Automation (ETFA)* (S. 274–281). IEEE. doi: 10.1109/etfa.2018.8502516

Seifert, C. & Wislaug, J. (2017). *Spiele entwickeln mit Unity 5*. München: CARL HANSER Verlag. doi: 10.3139/9783446453685

Speicher, M., Tenhaft, K., Heinen, S. & Handorf, H. (2015). Enabling industry 4.0 with holobuilder. In D. W. Cunningham, P. Hofstedt, K. Meer, I. Schmitt & B. Gesellschaft für Informatik e. V. (Hrsg.), *GI-Edition Proceedings Band 246, 45. Jahrestagung der Gesellschaft für Informatik - INFORMATIK 2015* (S. 1561–1575). Bonn: Köllen.

Sung, K., Pavleas, J., Arnez, F. & Pace, J. (2015). *Build Your Own 2D Game Engine and Create Great Web Games*. Berkeley, CA: Apress. doi: 10.1007/978-1-4842-0952-3

Unity Technologies. (2018a). *Platform dependent compilation.* Zugriff am 03.03.2021 auf https://docs.unity3d.com/2019.1/Documentation/Manual/PlatformDependentCompilation.html

Unity Technologies. (2018b). *WebGL: Interacting with browser scripting.* Zugriff am 20.02.2021 auf https://docs.unity3d.com/Manual/webgl-interactingwithbrowserscripting.html

# Anhang

**Übersicht über elektronische Anhänge:**

- Quellcode des implementierten Systems innerhalb der Abschlussarbeit
- Abschlussarbeit in elektronischer Form