Entwurf und prototypische Implementierung eines Sprachassistenzsystemes für ARBURGs Maschinensteuerung GESTICA

Praxisarbeit
T3_2000

des Studiengangs Informatik
an der Dualen Hochschule Stuttgart Campus Horb

von
Jonas Weis

04. Oktober 2022

Bearbeitungszeitraum	KW 23 - KW 40
Matrikelnummer	3758638
Kurs	TINF2020
Ausbildungsfirma	ARBURG GmbH & Co KG Arthur-Hehl-Straße, 72290 Loßburg
Betreuer	B. Eng., Benedikt Link

Sperrvermerk

Der Inhalt dieser Arbeit darf weder als Ganzes noch in Auszügen Personen außerhalb des Prüfungsprozesses und des Evaluationsverfahrens zugänglich gemacht werden, sofern keine anderslautende Genehmigung der Firma ARBURG GmbH & Co KG vorliegt.

Eidesstattliche Erklärung

gemäß § 5 (4) der „Studien- und Prüfungsordnung DHBW Technik“ vom 14. Juli 2021. Ich versichere hiermit, dass ich meine Projektarbeit mit dem Thema

Entwurf und prototypische Implementierung eines Sprachassistenzsystemes für ARBURGs Maschinensteuerung GESTICA

selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe.

Loßburg, den 16. September 2022

Ort,Datum

Unterschrift

Zusammenfassung

Abstract

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Quellcodeverzeichnis

1 Einleitung

Dieses Kapitel gibt eine Übersicht über das Unternehmen ARBURG. Nachfolgend wird die Aufgabenstellung aufgezeigt, deren Motivation erklärt sowie die Vorgehensweise geplant.

1.1 Unternehmensportrait

Das Familienunternehmen ARBURG wurde 1923 von Arthur Hehl gegründet. Mittlerweile ist ARBURG einer der weltweit führenden Hersteller von Spritzgießmaschinen zur Kunststoffverarbeitung. Das Stammwerk mit einer Fläche von 210.000 Quadratmetern befindet sich in Loßburg und ist Arbeitsplatz von rund 2.900 Mitarbeitern. Es handelt sich um den einzigen Produktionsstandort. Der Eigenfertigungsanteil der Produktion liegt bei circa 60%. An knapp 100 Standorten weltweit sind insgesamt rund 600 weitere Mitarbeiter für Service, Vertrieb und Beratung angestellt. 2021 betrug der konsolidierte Umsatz 735 Millionen Euro. Im Portfolio von ARBURG befinden sich elektrische, hybride und hydraulische Spritzgießmaschinen (siehe Abbildung 1 a), die ALLROUNDER genannt werden. Außerdem zugehörige Peripherie und Robot-Systeme. Im Bereich der additiven Fertigung ergänzt seit 2013 der Freeformer (siehe Abbildung 1 b) das Angebot der Firma.

(a) Spritzgießmaschine ALLROUNDER

(b) Freeformer

Abbildung 1: Produkte der Firma ARBURG [1]

Zur Bedienung der Maschinen ist die SELOGICA (SELOGICA Maschinensteuerung) und seit 2016 auch die GESTICA (siehe Abbildung 2) in Verwendung.

Abbildung 2: GESTICA [1]

1.2 Motivation

Seit Jahren nimmt die Spracherkennung- und Steuerung eine größere werdende Rolle ein. Bereits 2019 nutzen in Deutschland 51 Prozent der 18-24 Jährigen und etwa ein Drittel der 25-64 Jährigen einen Sprachassistenten per Smartphone [**Sprachassisten.2019**]. Der Absatz von intelligenten Lautsprechern hat sich seit Anfang 2017 mehr als verzehnfacht [**SmarteLautsprecher.2021**]. Auch im Unternehmen ARBURG besteht der Wunsch, die Steuerung der Maschinen durch eine Sprachsteuerung zu verbessern. Aktuell ist dies nicht möglich. Dieser Wunsch ergibt sich aus dem Vorsatz, der Konkurrenz stets einen Schritt voraus zu sein und den aktuellen Stand der Technik voranzubringen. Grundgedanke ist die einfachere und schnellere Bedienung der Maschine für den Nutzer. Je nach Konzeption entfällt die Notwendigkeit für freie und saubere Hände, was die Wartung erleichtert. Die Implementierung eines solchen Features bietet dem Kunden einen Mehrwert und kann somit mehr Umsatz generieren.

1.3 Aufgabenstellung und Vorgehensweise

Aufgabe ist es, eine Sprachsteuerung für die ARBURG-Maschinen zu konzeptionieren. Zunächst gibt eine Literaturrecherche Aufschluss darüber, wie verschiedene Spracherkennungen in der Theorie funktionieren und welche Chancen und Probleme dort auftreten können. Folgend werden unterschiedliche Anforderungen an ein Sprach-Framework definiert. Anhand dieser folgt eine Analyse verschiedener Software auf ihre Eignung. Schließlich findet die Auswahl der geeignetsten für die Nutzung im Hause ARBURG statt. Außerdem werden

Anforderungen für das Konzept der Sprachsteuerung definiert. Dies geschieht mithilfe des Befragens von Service-Technikern und Mitarbeitern des Schulungscenters. Diese haben zum einen direkten Kontakt mit dem Kunden und dessen Wünschen. Zum anderen kennen sie sich mit der Maschine aus und können daher die Sinnhaftigkeit verschiedener Ideen abschätzen. Dabei ist es nicht vorgesehen, jede Aktion auch durch einen Sprachbefehl ausführen zu können. Hier muss abgegrenzt werden, was sinnvoll und realistisch umzusetzen ist. Neben dem Ansteuern bereits vorhandener Funktionalitäten sollen, wenn notwendig, gegebenenfalls neue implementiert werden. Auf der anderen Seite dürfen gewisse Szenarien aufgrund von Sicherheitsaspekten nicht umgesetzt werden. Der Datenschutz stellt auch einen wichtigen Punkt zur Berücksichtigung dar. Anhand dieser Überlegungen und Anforderungen wird ein Konzept erstellt und eine Softwarearchitektur modelliert. Folgend findet eine Implementierung der Kernfunktionen statt. Tests in verschiedenen Lärmumgebungen sollen Aufschluss über die Performance geben.

1.4 Ziel der Arbeit

Ziel der Arbeit ist die Auswahl einer Software sowie die Erstellung eines Konzeptes zur Nutzung einer Spracherkennung in der GESTICA. Das Konzept muss dabei realisierbar, sinnvoll und zukunftsfähig sein, da die Entwicklung im Bereich der Spracherkennung noch nicht zu Ende ist. Das Endprodukt hat daher den Aspekten Austauschbarkeit, Veränderbarkeit und Erweiterbarkeit zu genügen. Kernkomponenten des Konzeptes werden dabei prototypisch implementiert. Anhand dieser Implementierung wird ein Fazit über die Sinnhaftigkeit der Sprachsteuerung für ARBURG gezogen. In diesem sollen folgende Fragen geklärt sein.

- Ist eine passende Software verfügbar?
- Ist die ausgewählte Software ausreichend oder muss gegebenenfalls eine andere verwendet und daher Anforderungen an diese verändert werden?
- Erlaubt die GESTICA-Infrastruktur ein problemloses Einbinden der Sprachsteuerung?
- Ist eine Sprachsteuerung in lauten Umgebungen überhaupt nutzbar?
- Gibt es genug Use Cases um eine vollständige Implementierung des Konzeptes zu rechtfertigen?

2 Grundlagen der Softwarearchitektur

In diesem Kapitel werden Grundlagen der Softwarearchitektur im Hause ARBURG aufgezeigt.

2.1 .NET

Beim .NET-Framework handelt es sich um eine Softwareplattform von Microsoft, welche die Anwendungsentwicklung für Microsoft-Betriebssysteme vereinheitlicht. Es eignet sich für Webseiten, Services und Desktop-Apps [2]. Zum einen stellt es eine Klassenbibliothek für umfangreiche Funktionalitäten zur Verfügung. Zum anderen wird mit der CLR (Common Language Runtime) eine Laufzeitumgebung bereitgestellt, welche sich unter anderem um die Ausführung von Anwendungen, Garbage Collection und Thread-Management kümmert. Die Applikationen können in C-Sharp, F-Sharp oder Visual Basic geschrieben werden. Sie werden in einen sprachunabhängigen *Intermediate Language Code* kompiliert und in Assemblys gespeichert (.dll/.exe). Bei der Ausführung einer Applikation übersetzt ein just-in-time Compiler diese Assemblys in Maschinencode, der auf dem jeweiligen Computer ausgeführt werden kann [2].

Für ein besseres Verständnis später folgender Code-Ausschnitte folgt eine Erklärung zu zwei Funktionalitäten in .NET.

Out-Parameter werden durch das Schlüsselwort *out* festgelegt. Sie ermöglichen eine Rückgabe des Erfolges als auch des gesuchten Wertes. Somit wird direkt überprüft, ob weiter mit diesem Wert gearbeitet werden kann.

Quellcode 1: Beispiel zu out-Parametern

```
private bool TryGetValue(out int value)
{
  value = valueService.GetValue();
  if(value == null) return false;
  else return true;
}
```

Optionale Parameter werden durch den Zusatz = *null* festgelegt. Beim Aufruf kann ein optionaler Parameter entweder gesetzt werden oder nicht. Auch eine Variable mit dem Wert *null* kann übergeben werden. Ersetzbar wäre diese Methode durch Überladung. In der Implementierung ist diese Lösung jedoch notwendig aufgrund von Polymorphie (Abschnitt 2.2).

Quellcode 2: Beispiel zu optionalen Parametern

```
private void ExecuteCommand(string name = null)
{
  if(name != null) [...]
  else [...]
}
```

2.2 Objektorientierung

OOP (Objektorientierte Programmierung) ist ein Konzept zur Minderung von Komplexität in der Softwareentwicklung. Es werden Klassen und Objekte genutzt, um eine Software zu strukturieren. Die Grundelemente der OOP sind:

- Datenkapselung: Daten gehören explizit zu einem Objekt, ein Zugriff kann nur für Berechtigte über eine klar definierte Schnittstelle erfolgen (Beispielweise über Getter). Die Konsistenz der Daten kann somit besser sichergestellt werden [3]. Programmteile haben nur auf das Zugriff, dass sie benötigen.

- Polymorphie: Eine Oberklasse gibt die Grundstruktur an, erbende Klassen implementieren diese in verschiedenen Ausführungen. Beispielsweise hat der Obertyp *Tier* die Methoden *fortbewegen* und *ernähren*. Davon erben die Untertypen *Haifisch* und *Kamel*. Beide bewegen sich fort und ernähren sich. Der *Haifisch schwimmt* und *jagt Tiere*. Das Kamel dagegen *läuft* und *frisst Pflanzen*. Die gleichnamigen Methoden werden also unterschiedlich implementiert. Nun kann im Programm abstrakt *Tier* verwendet und die Methode *ernähren* aufgerufen werden. Erst zur Laufzeit (auch spätes Binden genannt), je nach Zuweisung eines Objektes, wird entschieden, ob die Methode beim *Haifisch* oder *Kamel* ausgeführt wird. Dies sorgt für Austausch-, Änder- und Erweiterbarkeit [3].

- Vererbung: Attribute und Methoden einer Oberklasse können an Unterklassen weitergegeben und somit wiederverwendet werden. Das Beispiel aus der Polymorphie

wird erneut aufgegriffen. Sowohl *Haifisch* als auch *Kamel* besitzen Attribute wie *Größe*, *Gewicht* und *Schnelligkeit*. Im Falle der Vererbung können diese direkt von der Oberklasse übernommen werden. Dies verhindert Redundanz und fördert Wiederverwendbarkeit [3]. Auch redundante Methoden können von der Oberklasse genutzt werden.

2.3 Composite Components und Dependency Inversion

Die Composite Components Architecture zielt auf eine Entkopplung von Abhängigkeiten auf Komponenten-Ebene. Im Fall von ARBURG ist damit die Projektebene gemeint. Dependency Inversion verfolgt dieses Ziel auf Klassenebene. Konkret soll eine Abhängigkeit nur von Abstraktionen und nicht von Details stattfinden. Dies wird durch die Trennung einer Komponente in den Implementierungsteil und Contracts-Teil (zu Deutsch: Vertrag) erreicht. Für jede Implementierung einer Komponente gibt es auch einen Contract. Dieser besteht letztlich aus Schnittstellen und Datenklassen. Es findet also eine Abstrahierung statt. Abhängigkeiten dürfen dabei nur zu Contracts bestehen (siehe Abbildung 3) und nicht zu Implementierungen. Dies soll für eine bessere Änder- und Austauschbarkeit sorgen, denn der Contract bleibt bestehen, während die Implementierung angepasst oder ausgetauscht werden kann [4].

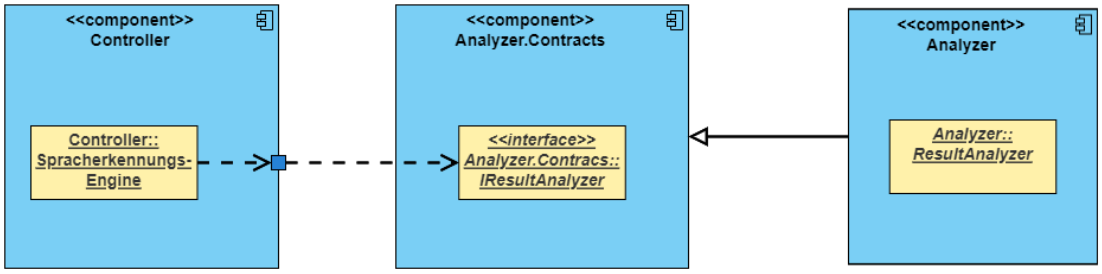

Abbildung 3: Composite Components

2.4 Dependency Injection

Ein Entwurfsmuster aus der Objektorientierung ist die DI (Dependency Injection), also Abhängigkeitsinjektion. Es soll Abhängigkeiten entkoppeln und auf ein Minimum reduzieren. Dabei handelt es sich um eine Weiterentwicklung der *Inversion of Control* aus 2004 durch Martin Fowler [5]. Konkret umgesetzt wird dies durch eine Fabrik oder einen Container, in welchen Objekte zentral erzeugt werden. Dieser injiziert diese dann passend, wenn benötigt. Folgend ein Beispiel:

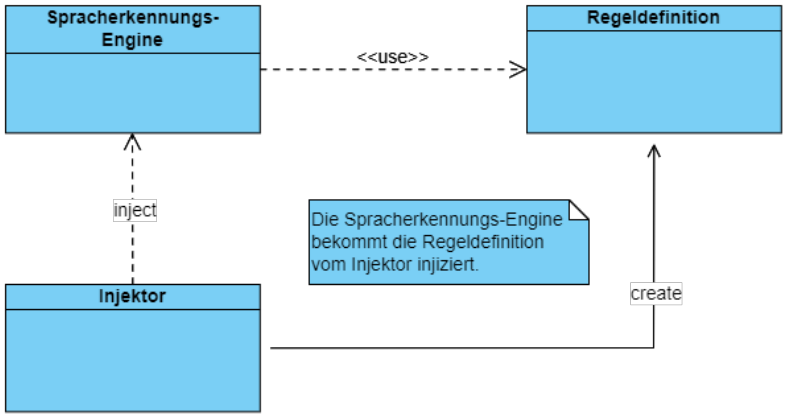

Abbildung 4: Dependency Injection

Im Beispiel (siehe Abbildung 4) braucht die *Spacherkennungsengine* eine Instanz der *Regeldefinition*. Diese wird zentral im Injektor erzeugt und ihr dann injiziert. Dazu wird zwischen drei Methoden unterschieden [5], im Unternehmen ARBURG wird folgende genutzt:

- Constructor Injection: Die Abhängigkeit wird direkt im Konstruktor als Argument übergeben.

Die Objekte müssen dabei einmalig dem DI-Container zugewiesen werden. Dies sieht folgendermaßen aus.

Quellcode 3: Dependency Injection

```
this.ExportPart<ISpeechAssist, SpeechAssist>().AsSingleton();
this.ExportPart<ITextResourcesExtractor, TextResourcesExtractor>();
this.ExportPart<ITokenHolder, TokenHolder>().AsSingleton();
this.ExportPart<IResultAnalyzer, ResultAnalyzer>();
```

Gemäß dem Dependency Inversion Prinzip (siehe Abschnitt 2.3) wird jeder Implementierung eine Abstraktion zugeordnet. Die Instanzen können als einmalig, also als Singleton, definiert werden, um immer das gleiche Objekt zu nutzen.

Literaturverzeichnis

[1] ARBURG GmbH + Co KG, Hrsg., *Mediendatenbank*, 2021.

[2] *.NET and .NET Framework*. [Online]. URL: https://dotnet.microsoft.com/en-us/learn/dotnet/what-is-dotnet-framework (besucht am 10. 02. 2022).

[3] Lahres, B. und Rayman, G., *Objektorientierte Programmierung - Das umfassende Handbuch*, 2st. Rheinwerk Computing, 2021, ISBN: 9783836214018.

[4] Tielke, D., „Composite Components: Eine Architektur für die Praxis, Teil 3“, *dotnetpro*, 2015. (besucht am 25. 02. 2022).

[5] Fowler, M., *Inversion of Control Containers and the Dependency Injection pattern*, 2004. [Online]. URL: https://martinfowler.com/articles/injection.html (besucht am 25. 02. 2022).

Anhang

A Sprachen der Steuerung

In der Steuerung sind momentan folgende Sprachen verfügbar:

Sprache	Kürzel
Deutsch	de-DE
Bulgarisch	bg-BG
Portugiesisch (Brasilien)	pt-BR
Tschechisch	cs-CZ
Dänisch	da-DK
Griechisch	el-GR
Englisch (GB)	en-GB
Spanisch	es-ES
Estnisch	et-EE
Finnisch	fi-FI
Französisch	fr-FR
Kroatisch	hr-HR
Ungarisch	hu-HU
Italienisch	it-IT
Japanisch	ja-JP
Koreanisch	ko-KR
Litauisch	lt-LT
Lettisch	lv-LV
Norwegisch	no-NO
Niederländisch	nl-NL
Polnisch	pl-PL
Rumänisch	ro-RO
Russisch	ru-RU

Slowakisch	sk-SK
Slowenisch	sl-SL
Serbisch	sr-RS
Schwedisch	sv-SE
Türkisch	tr-TR
Vietnamesisch	vi-VN
Chinesisch	zh-CN

Tabelle 1: In der Steuerung verfügbare Sprachen und ihre Kürzel