Konzeptionierung und Realisierung eines Multiagentensystems am Beispiel des Projektes InMachine

Conceptual design and implementation of a multi-agent system in context of the project InMachine

An der Fachhochschule Dortmund
im Fachbereich Informatik
Studiengang Informatik
Vertiefung Praktische Informatik
erstellte Thesis

zur Erlangung des akademischen Grades
Master of Science
M. Sc.

von
David Grimm
geboren am 01.03.1990

Andreas J. Wojtok
geboren am 08.12.1988

Betreuung durch:
Prof. Dr. Martin Hirsch
Dortmund, 21. Juli 2017

Kurzzusammenfassung

Unternehmen stehen auf Grund von Globalisierung, Konkurrenzdruck und immer schneller agierenden Märkten vor der Herausforderung auf diese Veränderungen flexibel reagieren zu müssen. Unternehmen die sich schnell auf Marktveränderungen einstellen können haben einen Wettbewerbsvorteil der beibehalten werden muss. Beim verarbeitenden Gewerbe resultiert das in einer Optimierung der Produktionsplanung- und Steuerung. Um eine Optimierung der Produktionsplanung- und Steuerung vornehmen zu können muss zunächst Einblick in diese zur Verfügung stehen. Kleinen und mittelständischen Unternehmen (KMUs) sind in der Regel nicht in der Lage die Kosten und Komplexität von Softwarelösungen von Herstellern wie Siemens, Dassault, oder SAP zu handhaben. Aufgrund dessen ist es notwendig Softwarelösungen anzubieten die genau auf das Einsatzszenario in KMUs zugeschnitten sind.

Ziel dieser Arbeit ist das Erstellen einer Softwarelösung, um eine Optimierung der Produktionsplanung- und Steuerung zu ermöglichen, durch Einblick und Rückmeldung der Produktionsprozesse. Um dieses Ziel zu erreichen wurden zunächst Anforderungen an das Gesamtsystem gestellt. Diese Anforderungen fließen in das zu erstellende Softwarekonzept ein. Beim Softwarekonzept wurde besonders auf die lose Koppelung der Komponenten und der flexiblen Kommunikation geachtet, dadurch ist es möglich das Softwarekonzept zukunftssicher aufzustellen und das nachträgliche Erweitern der Software zu ermöglichen. Durch die Anforderung an Unternehmen flexibel auf Marktveränderungen reagieren zu können resultiert auch die Anforderung an die eingesetzte Software flexibel auf neue Gegebenheiten angepasst werden zu können. Wird diese Anpassbarkeit bereits beim Softwarekonzept berücksichtigt können Änderungen einfacher, robuster, und zu geringeren Kosten realisiert werden. Nach der Erstellung des Softwarekonzeptes wurde dieses prototypisch Implementiert. Dieses Vorgehen sichert die Qualität und Konsistenz des Softwarekonzeptes ab. Abschließend wird eine Zusammenfassung gegeben, ein Fazit gezogen und ein Ausblick gewährt.

Abstract

Companies are facing the challenge to adapt their behavior and react on changes initiated through globalization, competitive pressure, and fast changing markets. Companies that are capable of changing their behavior have a competitive advantage and this advantage has to be maintained. In the manufacturing industry this results in optimizing the production planning and scheduling. To enable the possibility to optimize the production planning and scheduling you have to gain insight in the production-process. Small to medium sized companies are not capable of handling the costs and complexity of software-solutions from Siemens, Dassault, or SAP. Because of that, it is necessary to create a tailor-made software-solution that fits exactly the needs of small to medium sized companies.

Our goal is to create a software-solution that is capable to support the production planning and scheduling by gaining real-time insight and feedback from the production-processes. To obtain this goal requirements for the software-solution were defined. While creating the software-concept loose coupling and flexible communication were encouraged to create a system that is future proof and easy to extend. Because companies have to be flexible, so has to be the software they are using. If this flexibility is considered while developing the software-concept, it is possible to easily extend the software to low costs. After creating the software-concept it will be implemented as a prototype. This ensures the quality and consistency of the software-concept. Finally a summary, a conclusion, and an outlook will be given.

Inhaltsverzeichnis

Kapitel 1

Einleitung

„In Zeiten von Finanzkrise, Globalisierung, Konkurrenzdruck und immer schneller agierenden Märkten und wechselnden Trends steigen auch die Anforderungen an ein Unternehmen. Wer sich weiterhin Wettbewerbsvorteile sichern möchte, muss auf Marktveränderungen schnell reagieren sowie Chancen erkennen und entsprechend handeln.“ [PB14] Unternehmen müssen über eine hohe Reaktionsfähigkeit verfügen um auf Veränderungen rechtzeitig reagieren zu können und Nachteile zu vermeiden. [BH05] Diese Reaktionsfähigkeit ist ein entscheidendes Kriterium für wirtschaftlichen Erfolg im nationalen, sowie globalen Wettbewerb. Die Änderung der Produktionsanforderungen und -bedingungen können verschiedene Gründe haben:

- Lieferantenwechsel
- Kosteneinsparung
- Qualitätsmängel
- Technische Verbesserungen
- Innovationen am Markt

[Dic15] Die eingesetzten Produktionsprozesse auf die veränderten Produktionsanforderungen und -bedingungen anzupassen bietet ein hohes Nutzenpotential. Auch die Optimierung der Produktionsprozesse in Hinblick auf energetische und ökologische Kennwerte ist von großem Interesse.

Innerhalb von KMUs herrschen gegenüber Großunternehmen weniger formalisierte Organisationsstrukturen. Dies hat eine direkte Auswirkung auf die Reaktionsfähigkeit der Unternehmen. „Eine schnelle Reaktionsfähigkeit ist für KMU überlebensnotwendig, sie können ihr Umfeld nicht wie Großunternehmen beeinflussen, sondern werden in hohem Maße von externen Kräften beeinflusst.“ [Lan09] Die hohe Flexibilität von KMUs muss sich auch in der verwendeten IT-Infrastruktur abbilden. Eine Produktionsplanung und Steuerung auf Grundlage von realen Produktions- und Maschinendaten bieten KMUs die Möglichkeit, ihren Wettbewerbsvorteil weiter auszubauen.

In der industriellen Produktion müssen Pläne und Ziele abgestimmt werden. Aufträge müssen den einzelnen Arbeitsplätzen und/oder Mitarbeitern entsprechend der Ausstattung oder Fähigkeiten zugewiesen werden. Vorhandene Arbeitsplätze und Mitarbeiter optimal auszulasten, aber auch Produktionstermine einzuhalten, erfordert die zeit genaue Abstimmung der einzelnen Produktionsschritte auf unterschiedlichen Arbeitsplätzen. Bei fehlender Abstimmung entstehen einem Unternehmen hohe Kosten durch eine nicht optimale Auslastung ihrer Ressourcen. Es ist im Interesse jedes Unternehmens, diese Cash-to-Cash-Cycle-Time zu reduzieren, aber auch die Arbeitsplätze und Mitarbeiter optimal auszulasten. Die Bereitstellung sehr großer bzw. redundanter Kapazitäten in der Produktion würde zwar die Cash-to-Cash-Cycle-Time erhöhen, es würden aber auch hohe Leerlaufzeiten entstehen. Demgegenüber steht eine maximale Ausnutzung der vorhandenen Kapazitäten in der Produktion, die wiederum in einer hohe Kapitalbindung in Form von unfertigen Produkten resultiert. Die Kapitalbindung minimal zu halten und gleichzeitig die Auslastung der Arbeitsplätze zu maximieren, ist eine komplexe Aufgabenstellung.

Um die Planungsproblematik und die Reaktionsfähigkeit zu verbessern ist es notwendig Daten über den Kundenwunschtermin, den Materialbedarf und dessen Verfügbarkeit, den Auftragsstatus, die Wartungsinformationen, die Maschinenstörungen, und die Maschinenkonfigurationsdaten zu erheben. Diese Daten müssen im benötigten Detaillierungsgrad vereinheitlicht sein und über die verschiedenen Planungs- und Steuerungsebenen hinweg zur Verfügung stehen. Dadurch ist es möglich auf Planungsebene auf Maschinenstörungen reagieren zu können und auf Steuerungsebene den Kundenwunschtermin zu berücksichtigen.

Häufig wird im Laufe eines Produktionsprozesses ein Teil der Fertigungsschritte an Zulieferer bzw. Veredler ausgelagert. In diesem überbetrieblichen Kontext ist die Datenverfügbarkeit verringert. Das Interesse an einer unternehmensübergreifenden Überwachung des Auftragsstatus und die Information über Störungen die einen termingetreuen Ablauf gefährden könnten, ist sehr groß. Der Wunsch besteht darin auch Daten unternehmensübergreifend in die Planungs- und Steuerungsebene einfließen lassen zu können, um die Reaktionsfähigkeit des eigenen Unternehmens zu erhöhen.

Für KMUs sind die Kosten und Komplexität von Product Lifecycle Management (PLM) / Enterprise Ressource Planning (ERP)-Systeme, wie Siemens, Dassault oder SAP, welche komplexe Planungs- und Steuerungsalgorithmen anbieten, in der Regel zu hoch. Daher sind KMUs auf zielgerichtete Softwarelösungen angewiesen, welche die beschriebene Problematik adressieren und in ihren Kosten und der Komplexität handhabbar sind.

Der Bericht Wirtschaft DIGITAL 2017 [DP17] des Bundesministerium für Wirtschaft und Energie (BMWi) zeigt die aktuelle Situation der Digitalisierung der deutschen Wirtschaft auf. „2017 liegt der Digitalisierungsgrad der deutschen Wirtschaft bei 54 von 100 möglichen Indexpunkten. [...] Das verarbeitende Gewerbe erreicht 42 Indexpunkte.“ [DP17] Das verarbeitende Gewerbe liegt mit 42 Indexpunkten unter dem deutschen Durchschnitt von 54 Punkten. Dies zeigt auf, dass noch viel ungenutztes Potential innerhalb dieses Gewerbes liegt, daher sind diese Arbeit und das Forschungsprojekt inMachine[1] notwendig.

[1] www.inMachine.de

1.1 Kontext der Arbeit

Diese Arbeit wird im Kontext der Forschungsprojektes „inMachine - Lokale Intelligenz und vernetzte Planung zur Effizienzsteigerung technischer Produktionsmaschinen in kollaborativen Produktionsverbünden von KMU“ durchgeführt. Das Forschungsprojekt inMachine beschäftigt sich mit der effizienten Produktionsplanung aufgrund von Echtzeitdaten in KMUs und wird von dem Bundesministerium für Bildung und Forschung gefördert. Die Laufzeit des Projektes geht vom 01.06.2016 bis zum 31.05.2019. Folgende Projektpartner sind in dem Projekt vertreten:

- Smart Mechatronics GmbH, Dortmund
- software4production GmbH, München
- Großewinkelmann GmbH & Co. KG, Rietberg-Varensell
- HEERMANN GmbH, Hagen
- Fraunhofer Institut für Materialfluss und Logistik, Dortmund
- Fachhochschule Dortmund

1.2 Zielsetzung der Arbeit

Das Ziel der Arbeit ist es, im beschriebenen Kontext, ein Multiagentensystem (MAS) aufzubauen. Dieses soll es ermöglichen den Produktionsprozess zu überwachen und somit die Reaktionsfähigkeit der KMU zu verbessern. Hierbei ist jedoch kein abgeschlossenes System zu entwickeln. Auf Basis von Recherchen, sowie des aktuellen Forschungsstandes, wird zunächst ein allgemeines Konzept erarbeitet, welches in einer prototypisch Software umgesetzt wird. Die Ergebnisse der Arbeiten bilden ein Grundgerüst im Kontext der Industrie 4.0 und sind wichtiger Bestandteil des Forschungsprojektes.

1.3 Vorgehensweise und Gliederung

Zu Beginn der Arbeit wird in Kapitel 1 die Thematik aufgezeigt und die benötigte Problemlösung motiviert. Anschließend wird in Kapitel 2 eine Anforderungsanalyse durchgeführt, innerhalb dieser wird die Ausgangssituation, der Systemkontext, die resultierenden Anforderungen, und die Abgrenzung der zu erstellenden Software beschrieben.
In Kapitel 3 werden die Grundlagen beschrieben, die zum Verständnis der Arbeit notwendig sind. Darauf folgt das Softwarekonzept in Kapitel 4. Dabei werden die eingesetzten Prinzipien vorgestellt, die Komponenten des Softwarekonzeptes dargelegt, das Hardwarekonzept mit der Integration in das Einsatzszenario vorgestellt, und die Interaktion durch Benutzerschnittstellen beschrieben. Anschließend wird in Kapitel 5 durch die prototypische Implementierung des Softwarekonzeptes dieses validiert. Zusätzlich werden die eingesetzten

Technologien, die Projektstruktur, Besonderheiten der Implementierung, die Erweiterbarkeit, und die Umsetzung der Benutzerschnittstellen vorgestellt.
In Kapitel 6 wird eine allumfassende Zusammenfassung gegeben, ein Fazit über den Themenkomplex gezogen, und ein Ausblick über die Zukunft des Projektes gewährt.

Inhaltliche Aufteilung

An dieser Stelle wird die inhaltliche Aufteilung der Arbeit zwischen den beiden Autoren dargestellt. Außer bei den gemeinsam bearbeiteten Kapitel werden nur die jeweiligen Unterkapitel angegeben, durch die Kombination der Unterkapitel ergibt sich das Gesamtkapitel.

Gemeinsam bearbeitet:
Kapitel 1, Kapitel 2.3, Kapitel 6,

David Grimm:
Kapitel 2.1, Kapitel 2.2, Kapitel 2.4, Kapitel 3.2, Kapitel 3.2.1, Kapitel 3.2.3, Kapitel 3.2.4, Kapitel 3.2.5, Kapitel 3.3, Kapitel 3.4, Kapitel 3.5, Kapitel 3.6, Kapitel 4.1.1, Kapitel 4.1.2, Kapitel 4.1.3, Kapitel 4.1.3.2, Kapitel 4.1.4, Kapitel 4.2.2, Kapitel 4.2.3, Kapitel 4.2.4, Kapitel 4.6, Kapitel 5.1.1, Kapitel 5.3.2, Kapitel 5.3.3, Kapitel 5.5, Kapitel 5.6.3, Kapitel 5.7.2.1

Andreas J. Wojtok:
Kapitel 3.1, Kapitel 3.2.2, Kapitel 3.7, Kapitel 4.1.3.1, Kapitel 4.1.5, Kapitel 4.2.1, Kapitel 4.2.5, Kapitel 4.3, Kapitel 4.4.1, Kapitel 4.4.2, Kapitel 4.5, Kapitel 4.7, Kapitel 5.1.2, Kapitel 5.1.3, Kapitel 5.2, Kapitel 5.3.1, Kapitel 5.4, Kapitel 5.6.1, Kapitel 5.6.2, Kapitel 5.7.1, Kapitel 5.7.2.2, Kapitel 5.8

Kapitel 2

Anforderungsanalyse

In diesem Kapitel werden die Anforderungen an das System analysiert. Dazu wird zunächst der Kontext betrachtet, indem das System eingebettet werden soll. Außerdem werden die einzelnen Rollen identifiziert und deren Anforderungen an das System festgehalten.

2.1 Ausgangssituation

An dieser Stelle wird die Ausgangssituation beschrieben, welche als Grundlage für die Erstellung der Softwarelösung dient. Zunächst wird davon ausgegangen, dass wir uns im Kontext der KMUs aufhalten. KMUs sind durch ihre Agilität und Flexibilität charakterisiert. Folgendes Einsatzszenario dient als Ausgangssituation:

Das Unternehmen, in dem die Software eingesetzt werden soll, verfügt über ein Produktionsplanungs- und Steuerungssystem (PPS-System). Das PPS-System übernimmt die Planung der Arbeitsabläufe und weist Aufträge einzelnen Maschinen oder Arbeitsplätzen zu. Für einen Auftrag, der einem Arbeitsplatz/Maschine zugewiesen wurde stehen alle Materialien bereit. Der Einkauf benötigter Materialien erfolgt vor der Zuweisung eines Auftrages. Das Einsatzszenario beschreibt den Einsatz der Softwarelösung in einem manuellen Produktionsprozess, der einen Mitarbeiter an einem Arbeitsplatz oder einer Maschine vorsieht. Der Mitarbeiter erhält bei Arbeitsbeginn und auch während der Arbeit Aufträge. Ein Auftrag wird vom Mitarbeiter mithilfe eines Laufzettels abgearbeitet, der Status seines Auftrages ist daher nur bei Einsicht des Laufzettels erkennbar. In regelmäßigen Abständen (täglich oder wöchentlich) wird der Status eines Auftrages mithilfe des Laufzettels in das PPS-System übertragen. AB diesem Zeitpunkt kann der Status auch von anderen Mitarbeitern eingesehen werden und eine neu Planung der Aufträge ist möglich. Durch die späte Digitalisierung der Laufzettel verliert das KMU Flexibilität und Agilität. Anfragen von Kunden bezüglich ihres Auftragsstatus können nicht zufriedenstellen und nur ungenau beantwortet werden. Anpassungen der Produktionsplanung geschehen verspätet, wodurch ggf. Lieferzeiten nicht mehr eingehalten werden können.

Neben dem Mitarbeiter existieren noch weitere Rollen, der Techniker und der Meister.

Der Meister kontrolliert, die Arbeit des Mitarbeiters und unterstützt die Abarbeitung der Aufträge durch sein Expertenwissen. Der Techniker führt Wartungen an den Maschinen oder Arbeitsplätzen durch.

Das zu entwickelnde System soll in dem Einsatzszenario genutzt werden können, um die Prozessüberwachung zu verbessern. Das System soll jedoch so allgemein gehalten werden, dass es in einem beliebigen KMU eingesetzt werden kann. In den meisten KMUs sind bereits IT-Systeme vorhanden z.B. PPS-System, Zeiterfassungssysteme oder ähnliches. Die Mitarbeiter sind mit Umgang der jeweiligen Systeme vertraut und die Systeme wurden auf die Bedürfnisse der KMUs angepasst. Es ist nicht Ziel der Arbeit diese Systeme zu ersetzen, sondern so zu erweitern, dass ein Mehrwert für das KMUs geschaffen wird. Da die angestrebte Lösung allgemein gehalten werden soll und keine Individuallösung für ein KMU entwickelt werden soll müssen einige Annahmen getroffen werden:

- Das PPS-System ist in der Lage Statusmeldungen über den Produktionsprozess entgegenzunehmen und Sensordaten, die während der Produktion anfallen zu speichern.
- Kundenaufträge werden in das PPS-System eingetragen und sind dort so gepflegt, dass alle nötigen Arbeitsschritte für die Abarbeitung des Auftrages hinterlegt sind.
- Das PPS-System bietet Schnittstellen, über die ersichtlich ist, welchem Arbeitsplatz welcher Auftrag zugeordnet ist und es ermöglichen die einzelnen Arbeitsschritte des Auftrages abzufragen.
- Das PPS-System bietet die Möglichkeit Informationen einsehen zu können, um beispielsweise Rückfragen von Kunden, z.B. über den aktuellen Status des Auftrages bzw. der Einhaltung des Liefertermins, beantworten zu können.

Sollte eine dieser Annahmen nicht auf ein KMU zutreffen, muss zusätzliche Software bereitgestellt werden, damit die Annahmen Erfüllt sind. Dies wird kurz an einem Beispiel erläutert:

Sollte ein KMU über ein PPS-System verfügen, in welches die gemessenen Daten während des Produktionsprozesses nicht gespeichert werden können, muss ein solcher Service bereitgestellt werden. Dazu würde eine neue Software entwickelt, die die gemessenen Prozessdaten speichert und den Benutzern aufbereitet anzeigt.

2.2 Systemkontext

Aus dem beschriebenen Einsatzszenario und den entsprechenden Annahmen ergibt sich folgender Systemkontext (vgl. Abbildung 2.1):

- **Agent**: Der Agent stellt das zu entwickelnde Softwaresystem dar. Er fungiert als Schnittstelle zwischen **Mitarbeiter** und PPS-System. Der Agent entnimmt über die Schnittstellen des PPS-System die vom Mitarbeiter benötigten Informationen und speichert Statusmeldungen und angefallene Sensordaten in diesem.

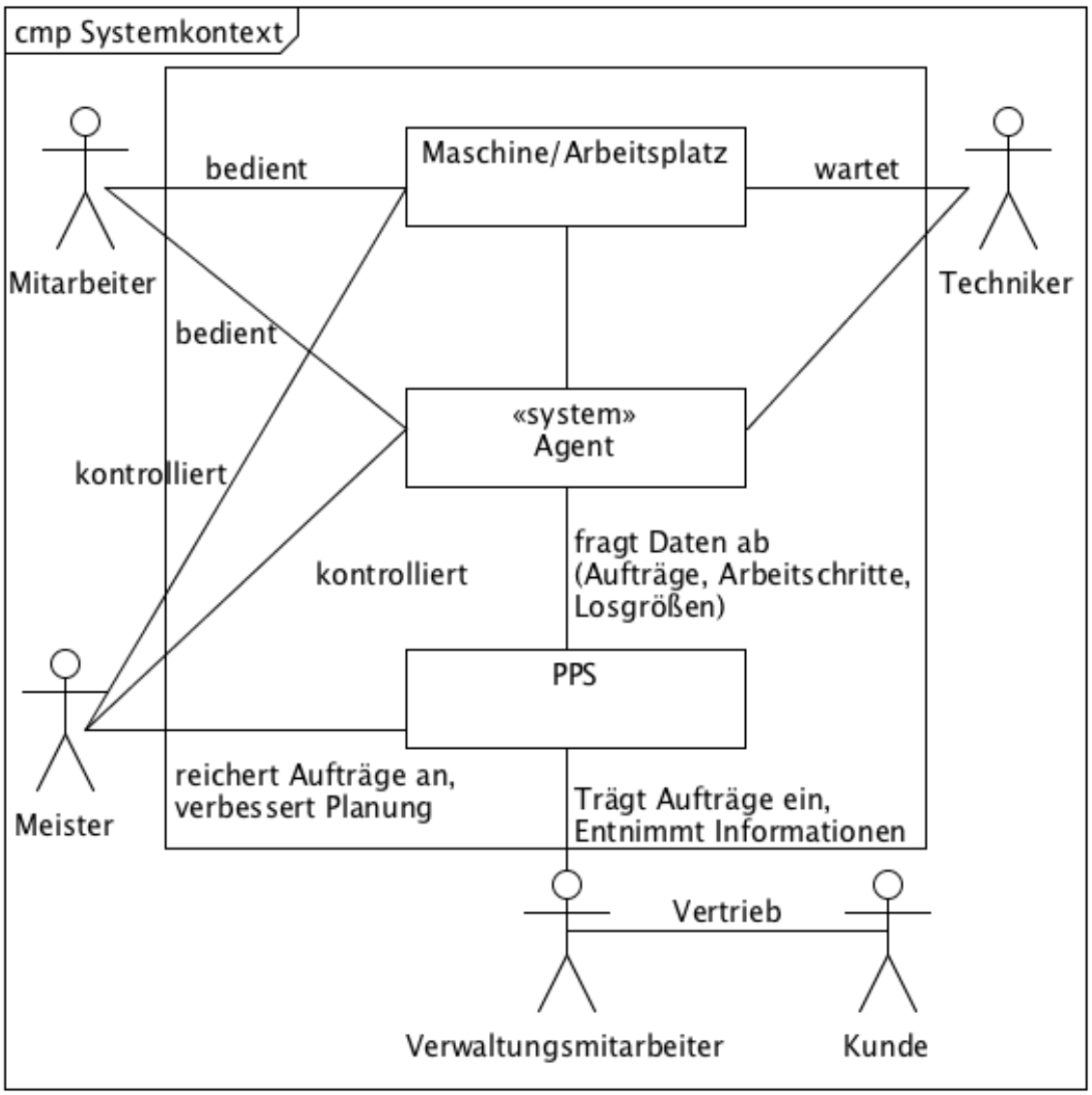


Abbildung 2.1: Systemkontext des zu entwickelnden Systems

- **PPS-System**: Das PPS-System verwaltet alle Aufträge und dient als zentrale Anlaufstelle, um den Produktionsprozess zu überwachen.
- **Mitarbeiter**: Ein Mitarbeiter arbeitet an einem Arbeitsplatz oder einer Maschine. Dieser agiert mit dem zu entwickelnden System (**Agent**) und hat die Möglichkeit darüber Statusmeldungen über den Produktionsprozess anzugeben und seine abzuarbeitenden Aufträge einzusehen.
- **Meister**: Ein Meister ist für die Überwachung des Produktionsprozesses zuständig. Er soll die Möglichkeit haben den Arbeitsfortschritt an einem Arbeitsplatz oder einer Maschine zu überwachen.
- **Techniker**: Ein Techniker wartet Maschinen und gibt über den Agenten Rückmeldungen über die Wartung.

Die verschiedenen Rollen im System können verschwimmen. Der Meister kann beispielsweise ebenfalls die Rolle des Technikers übernehmen.

2.3 Anforderungen

Im Folgenden werden die Anforderungen an das zu entwickelnde Softwaresystem definiert. Die Anforderungen werden entsprechend der verschiedenen Rollen im System getrennt. Zur Definition der Anforderungen wurde eine Schablone verwendet, diese Schablone ist wie folgt aufgebaut:
Nummerierung: Kürzel der Rolle + fortlaufende Nummer
Anforderungsschablone: Als ROLLE möchte ich FUNKTION, damit SINN.

2.3.1 Meister

ME.1 Als Meister muss ich die zugewiesenen Aufträge eines Arbeitsplatzes einsehen können, um die Richtigkeit zu prüfen und ggf. Optimierungen im PPS-System anzustoßen.

ME.2 Als Meister möchte ich den Fortschritt eines Auftrages einsehen können, damit ich die Einhaltung des Produktionsplans kontrollieren kann.

ME.3 Als Meister möchte ich über unvorhergesehene Ereignisse im Produktionsprozess, wie z.B. den Ausfall einer Maschine informiert werden, damit ich geeignete Maßnahmen durchführen kann.

ME.4 Als Meister möchte ich die Messwerte eines Arbeitsplatzes oder einer Maschine einsehen können, um die Ordnungsmäßigkeit kontrollieren zu können.

ME.5 Als Meister möchte ich mein Wissen einfließen lassen können, um Unregelmäßigkeiten im Produktionsprozess aufdecken zu können.

2.3.2 Mitarbeiter

MI.1 Als Mitarbeiter möchte ich die Aufträge einsehen können, die meinem Arbeitsplatz zugewiesen wurden, damit ich diese abarbeiten kann.

MI.2 Als Mitarbeiter möchte ich den Fortschritt der Abarbeitung melden können, damit bei Abweichungen die Planung angepasst werden kann.

MI.3 Als Mitarbeiter möchte ich unvorhergesehene Ereignisse, wie z.B. den Ausfall einer Maschine oder die Produktion von Ausschuss melden können, damit ggf. die Planung angepasst werden kann.

MI.4 Als Mitarbeiter möchte ich über Unregelmäßigkeiten an meinem Arbeitsplatz, wie z.B die erhöhte Temperatur einer Maschine informiert werden, damit ich geeignete Maßnahmen einleiten kann.

2.3.3 Techniker

T.1 Als Techniker möchte ich die Wartungsaufträge einer Maschine einsehen können, um die entsprechende Wartung durchführen zu können.

T.2 Als Techniker möchte ich den Fortschritt einer Wartung melden können, damit die Produktion entsprechend geplant werden kann.

2.4 Abgrenzung

Im Folgenden wird das zu erstellende System abgegrenzt:

- Ziel der Arbeit ist es nicht eine vollumfängliche Prozessüberwachung zu realisieren ferner soll ein Konzept entwickelt werden, das in unterschiedlichsten KMUs eingesetzt werden kann.
- Es soll kein konkretes PPS-System angebunden werden, die Anbindung soll nur exemplarisch gezeigt werden.
- Das System soll nur die Prozessüberwachung verbessern und wird keine Optimierung des Produktionsprozesses durchführen.

Kapitel 3

Grundlagen

Innerhalb dieses Kapitels werden die Grundlagen im Bereich MAS aufgezeigt. Besonders wird auf die Kommunikation und Vernetzung der einzelnen Agenten eingegangen.

3.1 Multiagentensysteme

Im Folgenden wird auf die Eigenschaften eingegangen, die ein MAS ausmachen.
Die Definition des Begriffes Agent ist essentiell zum Verständnis von MAS. Ein Agent ist ein Computer System, welches in der Lage ist autonom auf ein dynamisches und unvorhersehbares Umfeld zu reagieren. Jeder Agent verfolgt ein Ziel und ist in der Lage mit anderen Agenten zu kommunizieren [Ros85] [WJ95] [FLM95] [LMSW05].
Bei einem MAS handelt es sich um einen Zusammenschluss von mehreren Agenten, die ihre Ziele austauschen für eine gemeinsame, vorteilhafte Kooperation [Vei03]. Beim Umsetzen eines MAS besteht die Herausforderung das System so zu strukturieren, dass autonome und kommunikative Agenten integriert werden können [LMSW05].
Die Koordination und Verhandlung zwischen den autonomen Agenten ist dabei essentiell. Es muss sichergestellt werden, dass die Agenten ein einheitliches Ziel verfolgen, auch wenn die einzelnen Ziele der Agenten unterschiedlich sein können. Dabei ist die Herausforderung diese Koordination und Verhandlung so zu automatisieren, dass menschliches Eingreifen unnötig wird [LMSW05].

3.2 Kommunikation in MAS

Kommunikation ist in der Informatik ein Thema von zentraler Bedeutung [Woo09]. Oft ist die Kommunikation zwischen Maschinen ein Remote Procedure Call (RPC). Das bedeutet, dass auf einer entfernten Maschine das Ausführen einer definierten Methode ausgelöst wird.

> „Of course other means and approaches have been used over the years to achieve the lofty goal of seamless exchange of information and knowledge between

> applications. From Remote Procedure Call (RPC) or Remote Method Invocation (RMI), to CORBA and Object Request Brokers (ORB's) the goal has been the same. "[LFP99]

Bei MAS ist die Ausführung einer bestimmten Methode nicht im Interesse des Agenten. Die Kommunikation zwischen Agenten ist komplexer, da nicht nur Objekte ausgetauscht werden, sondern auch Vorschläge, Regeln und Aktionen [LFP99].

> „In general, agents can neither force other agents to perform some action, nor write data onto the internal state of other agents. This does not mean they cannot communicate, however. What they can do is perform actions - communicative actions - in an attempt to influence other agents appropriately." [Woo09]

In MAS dient Kommunikation dazu, Agenten zu beeinflussen und Übereinkünfte zu treffen, daher geht jeglicher Kommunikation eine Intention voraus [Woo09].

3.2.1 Speech Act

Der Philosoph John Austin bemerkte 1962 [AU09], dass einige Äußerungen in unserer Sprache die Charakteristik von Handlungen haben. Diese Äußerungen, die er als Sprechakte bezeichnet, verändern die Umwelt analog zu physischen Handlungen oder bedingen diese. Die Äußerung, die eine Handlung darstellt bzw. bedingt, kann je nach äußerem Umstand und Kontext oder je nach Sprecher oder Hörer, unterschiedliche Bedeutungen haben und gelingen oder missglücken [Aus72]. Austin unterscheidet zwischen drei Aspekten eines Sprechaktes:

- Der lokutionäre Sprechakt beschreibt die Handlung des Sagens, z.B: „Bitte mach mir einen Tee.".
- Der illokutionäre Sprechakt ist die Äußerung, mit der eine Tat vollzogen wird; etwas wird mitgeteilt, versprochen oder befohlen z.B: „Er hat mich gebeten Tee zu machen.".
- Die Perlokutionäre Äußerung stellt eine Erwartungshaltung dar, die eine entsprechende Maßnahmen des Empfängers mit sich bringt: Er wird überzeugt, überrascht oder abgeschreckt z.B. „Er brachte mich dazu Tee zu machen.".

Sprechakte wurden gegen 1970 erstmals eingesetzt, um Systeme zu erstellen, die planen können, wie autonom Ziele erreicht werden können [AHT90]. Sie setzten den Grundstein für verteilte Planung und wurden sukzessiv weiterentwickelt.

3.2.2 Sprachen zur Kommunikation von Agenten

Sprachen zur Kommunikation zwischen Agenten bieten die Möglichkeit, Informationen und Wissen auszutauschen [LFP99]. Finin, Labrou und Mayfield [FLM95] definieren drei Aspekte, die Agenten aufweisen müssen um effektiv interagieren und zusammenarbeiten zu können:

1. Eine gemeinsame Sprache
2. Ein gleiches Verständnis der ausgetauschten Informationen
3. Die Möglichkeit des Austausches von 1. und 2.

Eine verbreitete Sprache zur Kommunikation von Agenten ist die Knowledge Query Manipulation Language (KQML) [LMP04]. Daher soll dies Sprache im Folgenden vorgestellt werden.

KQML Die folgenden Informationen zur KQML sind [FWW+93] und [FLM95] entnommen die zur Spezifikation der KQML dienen. Bei der KQML handelt es sich sowhol um eine Sprache als auch um ein Protokoll, welches es Agenten ermöglicht, andere Agenten zu identifizieren, sich mit ihnen zu verbinden, und Informationen auszutauschen.

Der Aufbau einer KQML-Nachricht kann dem Folgenden entnommen werden.

(<performative>
:sender <word>
:receiver <word>
:reply-with <expression>
:content <expression>
:language <word>
:ontology <word>

Bei *<performative>* wir die Art bzw. der Verwendungszweck der Nachricht angegeben. Bei *:sender* und *:receiver* gibt man die beteiligten Agenten an. *:reply-with* gibt an, dass der Sender der Nachricht eine Antwort erwartet, welche das Feld *:in-reply-to <expression>* und die selbe *expression* enthält. Dies ermöglicht die eindeutige Zuordnung einer Antwort zu einer Anfrage. *:content* enthält den eigentlich Inhalt der Nachricht. Innerhalb von *:language* wird angegeben, um was für eine Art von Sprache es sich bei der in *:content* enthaltenen *expression* handelt. *:ontology* gibt an, um welche Ontologie es sich bei der in *:content* enthaltenen *expression* handelt, anders gesagt, zu welcher Menge von Elementen die verwendeten Konstanten innerhalb von *expression* gehören.

Beispiel:

(ask-one
:sender joe
:receiver stock-server
:reply-with ibm-stock
:content (PRICE IBM ?price)
:language LPROLOG
:ontology NYSE-TICKS)

Bei der Art der Nachricht handelt es sich um *ask one*, es wird ein bestimmter Agent einmal nach Informationen angefragt. Der *:sender* ist ein Agent namens *joe*. Der *:receiver*

Performative	Klasse	Beschreibung
tell	Basic informative	Angefragte Informationen liefern.
deny	Basic informative	Angefragte Aussage ist nicht wahr.
ask-if	Basic query	Anfrage ob die expression innerhalb von :content wahr ist.
ask-one	Basic query	Anfrage nach Informationen.
subscribe	Notification	Sender registriert sich beim Receiver um automatisch bei Änderungen der Informationen informiert zu werden.
broadcast	Networking	Nachricht an alle verbundenen Empfänger schicken.

Tabelle 3.1: Ausschnitt der möglichen Performative der KQML. Für eine komplette Liste siehe [FWW+93].

ist ein Agent namens *stock-server*. Durch *:reply-with ibm-stock* wird der Empfänger aufgefordert seine Antwort mit *:in-reply-to ibm-stock* anzureichern, damit eine Zuordnung von Anfrage und Antwort geschehen kann. *:content* enthält den Ausdruck um den Preis der IBM Aktie abzufragen. Der Ausdruck ist in der Sprache *LPROLOG* geschrieben, welches in *:language* angegeben ist und enthält Elemente der *:ontology NYSE-TICKS*. Die Antwort des stock-server sieht folgendermaßen aus.

(tell
:sender stock-server
:receiver joe
:in-reply-to ibm-stock
:content PRICE IBM 14
:language LPROLOG
:ontology NYSE-TICKS)

Ein Ausschnitt möglicher *performative* kann Tabelle 3.1 entnommen werden.

3.2.3 Treffen einer Übereinkunft

Die Fähigkeit Übereinkünfte zu treffen ist fundamental für intelligente, autonome Agenten, denn diese können unterschiedliche Ziele verfolgen. Übereinkünfte können nur getroffen werden wenn die Agenten über die Möglichkeit zur Verhandlung und Argumentation verfügen. Verhandlungen werden durch ein Protokoll unterstützt, dass die Regeln für die Verhandlung bestimmt. [Woo09]

Ein solches Protokoll sollte folgende Eigenschaften sicherstellen [Woo09]:

- **Garantiert erfolgreich:** Ein Protokoll garantiert den Erfolg, wenn es sicherstellt, dass eine Übereinkunft getroffen werden muss.

- **Maximales soziales Wohl:** Ein Protokoll maximiert das soziale Wohl, wenn es sicherstellt, dass das Ergebnis, die Summe der Nützlichkeit aller Verhandlungsteilnehmer maximiert.
- **Pareto-effizient:** Das Ergebnis einer Verhandlung ist pareto-effizient, wenn es kein anderes Ergebnis gibt, dass für mindestens einen Agenten nützlicher wäre, ohne für einen anderen Agenten nutzloser zu sein.
- **Individuell rational:** Ein Protokoll ist individuell rational, wenn es im Interesse aller Verhandlungsteilnehmer ist, sich an die Regeln des Protokolls zu halten.
- **Stabil:** Ein Protokoll ist stabil, wenn es allen Agenten einen Anreiz gibt, sich in einer bestimmten Art und Weise zu verhalten.
- **Einfach:** Ein einfaches Protokoll macht die geeignete Verhandlungstatregie für die Verhandlungsteilnehmer offensichtlich. Zusätzlich kann ein Verhandlungsteilnehmer leicht die optimale Verhandlungstatregie bestimmen.
- **Verteilt:** Ein Protokoll sollte sicherstellen, dass es keinen „Single-Point of Failure" gibt und idealerweise die nötige Kommunikation zwischen Agenten minimieren.

3.2.4 Auktion

Die einfachste Möglichkeit eine Übereinkunft zu treffen ist eine Auktion. Auktionen wurden durch die Internetplattform ebay [eBa] populär und erhielten Einzug in das Leben vieler Menschen. Bei einer Auktion gibt es zwei Rollen, den Bieter und den Auktionator. Das Ziel des Auktionators ist es, einen Gegenstand an einen Bieter zu verkaufen. Dabei möchte dieser einen möglichst guten Preis erzielen, der Bieter hingegen möchte den Preis minimieren. Der bei einer Auktion erzielte Preis ist abhängig vom Wert des Auktionsgegenstandes und der Art der Auktion. Im Folgenden werden diese einzelnen Punkte näher erläutert.

Der Wert eines Auktionsgegenstandes kann für jeden Bieter unterschiedlich sein, da der Auktionsgegenstand einen ideellen oder subjektiven Wert haben kann. Neben dem ideellen Wert beeinflusst auch der gesellschaftliche Wert die Auktion. Ein Bieter, der ein Gemälde mit der Intention kauft, es irgendwann weiter zu verkaufen, wird nicht mehr zahlen, als er hofft in einer späteren Auktion dafür zu bekommen.

Eine Auktion kann folgende Eigenschaften aufweisen, welche teilweise miteinander kombinierbar sind:

- **sealed-bid**: Die Bieter kennen nicht die Gebote der anderen Bieter.
- **open-cry**: Alle Bieter wissen, was die Anderen geboten haben.
- **single shot**: Es gibt nur eine Runde, in denen die Bieter ihre Gebote abgeben können.
- **ascending**: Der Gebotspreis beginnt niedrig und wird sukzessive durch Gebote erhöht.

- **descending**: Der Gebotspreis beginnt bei einem hohen fiktiven Wert und sinkt in einem bestimmten Zeitintervall.
- **first-price auction**: Der Bieter mit dem höchsten Gebot bekommt den Zuschlag.
- **scond-price auction**: Der Bieter mit dem höchsten Gebot bekommt den Zuschlag, zum Preis des zweithöchsten Gebotes.

Im Folgenden werden gängige Auktionsformen näher erläutert.

Trendverfahren Das Trenderverfahren ist auch unter dem Namen first-price-sealed-bid bekannt. Jeder Bieter gibt verdeckt ein Angebot ab. Das höchste Gebot erhält den Zuschlag. Ein Bieter bezieht bei der Höhe seines Gebotes die Konkurrenten mit ein. Er weiß nicht genau, wer sein direkten Konkurrenten bei der Auktion ist, und welche Gebote er abgeben werden. Meist besteht aber eine ungefähre Vorstellung, in welchem Bereich sich das beste Angebot der Konkurrenz befinden wird.

Vickrey-Auktion Die Vickrey-Auktion oder auch Philatelisten-Auktion ist eine second-price-seald-bid Auktion. Wie beim Trenderverfahren gibt jeder Bieter ein Gebot ab. Es handelt sich um eine verdeckte Auktion. Die Teilnehmer erfahren die anderen Gebote nicht. Der Auktionsgegenstand geht an den höchsten Bieter, nicht zu seinem eigenen, sondern zum Preis des Zweitbieters (second-price). Diese Art der Auktion scheint zunächst nicht sinnvoll zu sein, da der Auktionator einen höheren Preis erzielen könnte. Dieses Verfahren sorgt bei Auschreibungen dafür, dass der Einkäufer keine überhöten Angebote bekommt. Lieferanten mit überhöhten Angeboten stellen sich in einigen Fällen schlechter und niemals besser und bevorzugen deshalb wahre Angebote [DN97].

Englische Auktion Eine englische Auktion beginnt mit einem niedrigen Startpreis. Die Bieter haben die Möglichkeit, sukzessive immer höhere Gebote abzugeben (ascending). Jeder Bieter kann beliebig oft mitbieten. Die Auktion ist beendet, wenn keine weiteren Gebote mehr folgen. Der Bieter mit dem höchsten Gebot erhält den Zuschlag zu dem von ihm gebotenen Preis (first-price). Es handelt sich hierbei um eine offene Auktion (open-cry). Alle Teilnehmer kennen zumindest den aktuellen Preis.

Niederländische Auktion Die niederländische Auktion beginnt im Gegensatz zur englischen Auktion mit einem sehr hohen Startpreis. Dieser Preis wird vom Auktionator kontinuierlich um einen bestimmten Betrag verringert, bis ein Teilnehmer ein Gebot abgibt (descending). Der Bieter erhält den Zuschlag zum Preis seines Gebotes (first-price). Bei der englischen Auktion kann ein Auktionsteilnehmer immer wieder nach bieten. Bei der niederländischen Auktion ist die Auktion mit dem ersten Gebot vorbei. Es handelt sich um eine offene Auktion (open-cry), die Bieter kennen den aktuellen Preis.

3.2.5 Verhandlungen

Neben der Auktion gibt es noch andere Möglichkeiten eine Übereinkunft zu treffen. Eine dieser Möglichkeiten ist die Verhandlung. Zunächst muss definiert werden was genau unter einer Verhandlung verstanden wird. Dabei nutzen wir die Definition von Büttner.

> „Unter einer Verhandlung ist ein iterativer Kommunikations- und Entscheidungsprozess zwischen mindestens zwei sozialen Systemen zu verstehen, die ihre Ziele nicht durch unilaterale Aktionen erfüllen können und zum Erreichen eines Konsens Angebote und Argumente austauschen." [Büt11]

Jede Verhandlung besteht generell aus vier Teilen [Woo09]:

- Ein Satz aus Verhandlungsgegenständen, die als Angebot unterbreitet werden können.
- Ein Protokoll, welches festlegt, wie Angebote unterbreitet werden können.
- Eine Ansammlung an Strategien, jeweils eine für jeden Agenten. Die Strategie, die ein Agent verfolgt, ist nicht für die anderen Teilnehmer der Verhandlung direkt ersichtlich.
- Eine Regelung, die bestimmt, wann eine Übereinkunft getroffen wurde und welche Bestandteile diese Übereinkunft hat.

Verhandlungen bestehen meistens aus mehreren Runden, in denen jeder Agent ein Angebot unterbreitet. Bei Verhandlungen wird nicht nur wie bei einer Auktion der Preis abgestimmt, es können mehrere Attribute zur Verhandlung stehen z.B.: Bei dem Kauf eines Autos kann nicht nur über den Preis verhandelt werden sondern auch über die Länge der Garantie und andere Leistungen. Dadurch können Verhandlungen sehr komplex werden. Die Anzahl der Agenten, die an der Verhandlung teilnehmen und die Art und Weise wie diese miteinander interagieren, kann diese Komplexität noch steigern. Folgende Konstellationen sind gängig:

- **One-to-one Verhandlung**: Bei dieser Verhandlung sind zwei Agenten involviert.
- **y-to-one Verhandlung**: Hier verhandelt ein einziger Agent mit einer Gruppe von Agenten. Auktionen sind ein Besipiel für y-to-one Verhandlungen.
- **Many-to-many Verhandlung**: In dieser Konstellation verhandeln viele Agenten gleichzeitig mit vielen anderen Agenten. Dadurch kommen bis zu $\frac{n*(n-1)}{2}$ gleichzeitige one-to-one Verhandlungen zustande.

3.3 Netztopologie

Eine konsequente Vernetzung des MAS ist entscheidend, da der Ausfall des Netzes einen Produktionstopp nach sich ziehen könnte. Die Art und Weise, wie ein Netzwerk organisiert ist, nennt man Topologie. Im Folgenden werden die gängigsten Topologien näher erläutert. Knoten sind beliebige Endgeräte in der Topologie und Kanten stellen ihre Verbindungen zueinander dar.

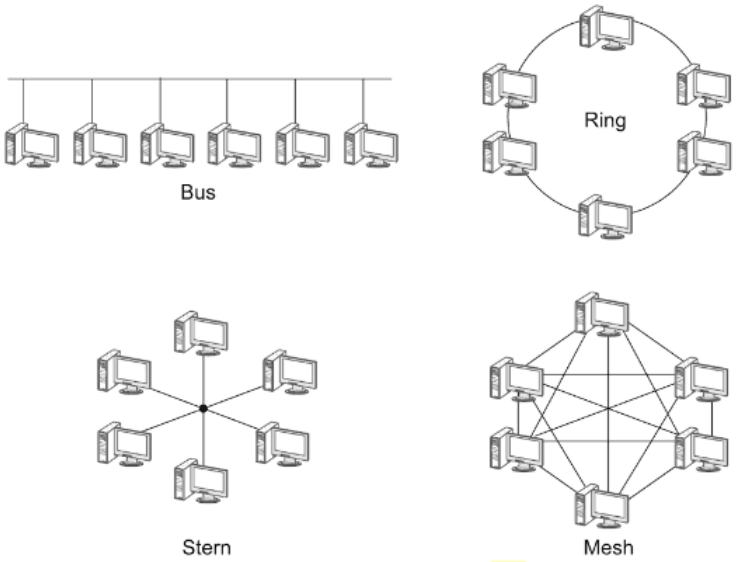


Abbildung 3.1: Im Uhrzeigersinn von oben links: Bus-, Ring-, Mesh- und Stern-Topologie [Mey16]

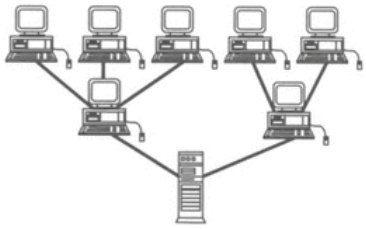


Abbildung 3.2: Abbildung einer Baumtopologie [TV01]

Stern

Bei der Stern-Topologie sind alle Knoten mit einem zentralen Verteiler verbunden [Nol09] (vgl. Abbildung 3.1). Dieser empfängt Signale, wertet diese aus oder leitet diese an den

richtigen Knoten weiter. Der Verteiler ist ein „Single Point of Failure“, jeglicher Datenaustausch zwischen den Knoten wird unmöglich, sobald dieser ausfällt.

Mesh

In einer Mesh-Toloplogie sind Knoten direkt miteinander verbunden (vgl. Abbildung 3.1). Die Knoten sind gleichwertige Kommunikationspartner. Sie leiten Daten an ihre Nachbarknoten weiter, sodass auch Knoten die nicht direkt miteinander verbunden sind kommunizieren können. Sind alle Knoten miteinander verbunden, ist das Netzwerk vollvermascht.

Bus

Bei einer Bus-Topologie sind alle Knoten im Netz an einer Leitung angeschlossen, dem sogenannten Bus-Kabel (vgl. Abbildung 3.1) [Mey16].

Ring

Bei einer Ring-Topologie sind je zwei Knoten des Netzes so miteinander verbunden, dass ein geschlossener Ring entsteht (vgl. Abbildung 3.1) [Mey16].

Ring-, Stern- und Bus-Topologien sind fehleranfällig. Der Ausfall weniger, sogar einzelner Geräte kann zum Gesamtausfall des Netzwerkes führen. Mesh-Topologien sind robuster. Jedoch bräuchte man zur Verkabelung eines vollvermaschten Netzes mit n Knoten $\frac{n*(n-1)}{n}$ Kabel [Mey16]. Ein Netz mit 10 Geräten bräuchte somit 45 Kabel. Daher wird diese Topologie praktisch nie für verkabelte Netzwerke eingesetzt [Mey16]. Der Einsatz in drahtlose Verbindungen oder eine Kombination aus drahtlosen und verkabelten Verbindungen wäre aber denkbar.

Baum

Die Baumtopologie kennzeichnet sich durch einen zentralen Knoten, der die Wurzel bildet und an den weitere Knoten angeschlossen sind, die sich ähnlich wie Äste und Zweige eines Baumes verzweigen [TV01] (vgl. Abbildung 3.2). Der Endknoten an einem Zweig oder Ast wird als „Blatt“ bezeichnet. Durch eine hierarchische Anordnung von Bus-, Stern- oder Ringtopologien mithilfe von Verteilern (Switch oder Hub) kann eine Baumstruktur erreicht werden. Der Vorteil dieser Topologie ist, dass der Ausfall eines Verteilers nur zum Ausfall eines Teilnetzes führt. Der Ausfall von Endkonten bleibt ohne Konsequenzen.

3.4 Sensoren & Sensornetzwerke

> „Ein Sensor ist eine Einrichtung zum Feststellen von physikalischen oder chemischen Eingangsgrößen, die optional eine Messwertzuordnung (Skalierung) der

> Größen treffen kann, sowie ggf. ein digitales bzw. digitalisierbares Ausgangssignal liefern kann." [Sch05]

Beispiele für Sensoren sind Sensoren zur Abstandskontrolle (Infrarot, Sonar), Temperatur- und Drucksensoren, aber auch Mikrofone, Kameras oder Touchscreens gehören zur Kategorie der Sensoren. Sensoren werden zunehmend mit Intelligenz in Form eines Mikrocontrollers u.ä. ausgestattet. Man nennt diese Art von Sensoren dann Smart-Sensoren [Moi].

Wenn viele Sensoren gemeinsam eine große physische Umgebung beaufsichtigen, formen diese ein Sensornetz. [DP10]. Yi Pan und Yang Xiao nennen drei Charakteristiken eines Sensornetzwerkes:

- **Datenzentrierung:** Bei einem Sensornetzwerk stehen Daten im Mittelpunkt. Das Ziel ist es, Daten, die von Sensoren erfasst wurden, dahin zu bringen, wo sie grade benötigt werden. Je nach Anwendungsfall werden Daten in unterschiedlicher Qualität benötigt. Sensornetzwerke dienen nicht dazu, die Erfassung von Rohdaten zu maximieren, sondern dazu, die Anzahl an nützlichen Informationen zu maximieren und den richtigen Stellen mitzuteilen.[PX06]

- **Anwendungsorientiert:** Sensornetzwerke werden erstellt, um spezifische Aufgaben zu erledigen. Das steht im Gengensatz zu normalen Netzwerken, die dazu erstellt werden, eine Vielzahl von Aufgaben zu übernehmen.[PX06]

- **Kollaboration:** Die Zusammenarbeit der Sensoren zur Erreichung eines spezifischen Ziels ist in Sensornetzwerken der wichtigste Aspekt.[PX06]

Abgrenzung zu MAS Der Begriff des Software-Agenten ist mehrdeutig, es existiert keine einheitliche Definition. Ein Sensor in einem Sensornetzwerk kann auch als Agent angesehen werden und ein Sensornetzwerk somit als ein System aus mehreren Agenten einem MAS. Eine Abgrenzung von Sensornetzwerk und MAS ist daher nötig.

Bei Sensornetzwerken stehen Daten im Mittelpunk. MAS hingegen versuchen kooperierend ein Problem zu lösen, wobei die Agenten auch unterschiedliche Ziele verfolgen können [LMSW05]. Dazu kommunizieren die Agenten untereinander und zeichnen sich durch autonomes Handeln aus. Sensornetzwerke kommunizieren nur, um Daten den richtigen Stellen mitzuteilen, die einzelnen Sensoren handeln in der Regel nicht autonom. Würde ein Sensor beispielsweise aufgrund seiner gesammelten Erfahrungen in bestimmten Zeiträumen weniger Messungen durchführen, wäre dies ein autonomes handeln. Dieser Sensor wäre gemäß der in Kapitel 3.1 gegebenen Definition ein Agent und mehrere diese Sensoren würde ein MAS bilden.

3.5 Vergleich Microservices MAS

Microservices sind ein aktueller Trend [Gar16]. Dabei wird eine Softwareanwendung in viele kleine unabhängige Services zerteilt. Diese verteilte Architektur hat parallelen zur MAS,

die im Folgenden herausgearbeitet werden sollen. Microservice-Architekturen sind nicht formal definiert, neben einer verteilten Architektur werden auch organisatorische Vorschläge gemacht. Es lassen sich aber gewisse Charakteristiken erkennen, die typischerweise in allen Microservices-Anwendungen vorhanden sind:

- **Ausrichtung anhand der Fähigkeiten und Merkmale des Unternehmens:** Oft werden Entwicklungen anhand der Technologien getrennt, was zu Oberflächen-Teams, Geschäftslogik-Teams und Datenbank-Teams führt. Änderungen in einem dieser Teams können Auswirkungen auf die anderen Teams haben, was zeit- und geldintensiv sein kann.

  > „Any organization that designs a system (defined more broadly here than just information systems) will inevitably produce a design whose structure is a copy of the organization's communication structure." [Con86]

  Entsprechend dieser These wird die Entwicklungen eines Systems anhand der Unternehmensstruktur geteilt. Jedes Team entwirft für eine der Abteilungen des Unternehmens einen Microservice. Es werden nur die Abteilungen des Unternehmens berücksichtigt, die Interesse an dem zu entwickelnden System haben. Das Team muss somit sowohl das Nutzerinterface, als auch die Persistenzschicht und alle externen Schnittstellen realisieren [Fow14]. Diese Trennung entspricht der Trennung von Zuständigkeiten [BL11, S. 31], richtet sich aber nach der im Unternehmen bestehenden Trennung.

- **Produkte nicht Projekte:** Bei der Microservice-Architektur wird die Idee verfolgt, Produkte zu entwickeln und keine Projekte durchzuführen. Die Teams sind für ihre Produkte so lange verantwortlich, wie diese genutzt werden. Bei Projekten ist es oft so, dass das Team sich nur solange für die Software verantwortlich fühlt, bis diese vom Auftraggeber abgenommen wurde. Bei der Microservice-Architektur kümmern sich die Teams um den Betrieb ihres Systems von der Installation über die Konfiguration bis hin zur Wartung. Dies sorgt für ein besseres Verständnis der Teams bezüglich des Verhaltens ihrer Software und ihrer Benutzer. Dadurch soll langfristig die Qualität und die Identifizierung des Teams mit dem Produkt verbessert werden [Fow14].

- **Smart end points and dump pipes:** Anwendungen, die die Microservice-Architektur nutzen, versuchen gleichzeitig möglichst entkoppelt und so zusammenhängend wie möglich zu sein [Fow14]. Dazu werden

  > „Die Aufgaben des gesamten Systems [...] in einzelne Verarbeitungsstufen zerlegt [...]. Jeder Verarbeitungsschritt wird in Form eines Filters implementiert. Jeweils zwei Filter werden durch eine Pipe verbunden.". [Gol11, S. 883]

- **Dezentrale Steuerung:** Anstatt sich auf einzelne Technologien zu konzentrieren und Standards für diese niederzuschreiben, versuchen Teams, die die Microservice-Architektur verwenden, nützliche Werkzeuge zu entwickeln. Diese Werkzeuge können

von anderen Teams wiederverwendet werden, um ähnliche Probleme zu lösen [Fow14]. Die Werkzeuge entstehen in der Regel während der Implementierung. Netflix und Google sind ein gutes Beispiel für eine solches Vorgehen. Dort ermutigen erprobte Bibliotheken andere Entwickler, ähnliche Probleme in ähnlicher Weise zu lösen. Einige dieser Bibliotheken werden auch der Öffentlichkeit zur Verfügung gestellt.

- **Dezentrales Datenmanagement:** Anders als bei monolithischen Systemen, wo oft eine einzige Datenbank favorisiert wird, bevorzugen Microservices eine eigene Datenbank pro Microservice, wobei verschiedene Arten von Datenbanken zum Einsatz kommen können [Fow14]. Ein Vorgehen, das „Polyglot Persistence“ genannt wird.

- **Automatisierung der Infrastruktur:** Wie auch bei monolithischen Systemen versucht man bei Microservices ein Buildmanagement zu etablieren. Dadurch soll die Anwendung automatisch getestet und veröffentlicht werden. Es werden außerdem Methoden zur kontinuierlichen Auslieferung und Integration genutzt [Fow14].

- **Entworfen für Fehler:** Eine Konsequenz aus der Nutzung der Microservice-Architektur ist, dass die einzelnen Microservices so entworfen werden müssen, dass sie Fehler anderer Microservices tolerieren können. Fehler müssen so schnell wie möglich erkannt und behoben werden können. Dies erfordert das ständige Überprüfen von Metriken der einzelnen Microservices [Fow14] sowie ein zentralisiertes Logging.

- **Evolutionäres Design:** Microservices werden so entworfen, dass Änderungen leicht vorgenommen werden können [Fow14]. Das evolutionäre Design ergibt sich aus den anderen Charakteristiken eines Microservices. Die Nutzung von Pipes und Filtern führt zum Beispiel zu einer sehr flexiblen und einfach erweiterbaren Architektur [Gol11, S. 839 ff.].

Diese Charakteristiken der Microservice Architektur lassen sich auch auf MAS übertragen.

3.6 Abbildung von Geschäftslogik

Geschäftslogik ist ein Begriff der Softwaretechnik, der die Logik der Software beschreibt, die zur Erfüllung einer Aufgabe nötig ist wie z.B. dem verarbeiten von Daten. Im Gegensatz dazu steht die Programmlogik die den Ablauf des Programms steuert. Es gibt verschiedene Ansätze die Geschäftslogik von der Programmlogik zu trennen, diese werden im Folgenden vorgestellt.

3.6.1 Schichtenarchitektur

Die Schichtenarchitektur dient dazu in verschiedenen Schichten die Programmlogik und Geschäftslosik zu trennen. Ziel dieser Architektur ist es, dass jede Schicht ausgetauscht werden kann ohne Änderungen an einer anderen Schicht durchführen zu müssen. Häufig

wird eine 3-Schichten-Architektur verwendet, wonach die Software in 3 Schichten eingeteilt wird: Präsentation und Benutzerinteraktion, Anwendungs- und Geschäftslogik sowie Datenschicht [Fow03].

3.6.2 Regelbasierte Systeme

Ein regelbasiertes System ist ein Softwaresystem, in welchem die Geschäftslogik von der Programmlogik abgekoppelt ist. Geschäftslogik wird in einem regelbasierten System durch Regeln abgebildet und ist nicht fest in eine Schicht einprogrammiert. Eine Regel ist in einer Wenn-Dann-Form und kann unterschiedlich interpretiert werden [LBB17] :

1. **WENN** Situation **DANN** Aktion
2. **WENN** Bedingung **DANN** Ergebnis
3. **WENN** Voraussetzung **DANN** Folgerung

> „Eine derartige Wissensdarstellung hat viele Vorteile. Regeln werden im Tagtäglichen häufig verwendet, sie sind intuitiv verständlich, entsprechen der menschlichen Denkweise und können als Handlungsanleitung oder auch zum Nachweis eines bestimmten Ergebnisses herangezogen werden.“ [LBB17]

So kann das Expertenwissen der Mitarbeiter die Geschäftslogik eines Systems anreichern. Ein regelbasiertes System besteht im Wesentlichen aus drei Komponenten [Win92]:

- **Datenbasis:** In der Datenbasis sind gültige Fakten des Systems gespeichert. Hierbei kann es sich sowohl um Messwerte, als auch um Ergebnisse aus vorherigen Berechnungen handeln.
- **Regelinterpreter:** Der Regelinterpreter bestimmt die ausführbaren Regeln und wendet diese an.
- **Regelbasis:** Ist die Menge der Regeln.

Aus dieser Trennung ergeben sich laut Pullmann folgende Vorteile [Pul00]:

- **Transparenz**: Aufgrund der transparenten Wissensrepräsentation kann das Verhalten von wissensbasierten Systemen in einfacher Weise erklärt und vorhergesagt werden.
- **Flexibilität**: Wissen kann in einfacher Weise hinzugefügt und entfernt werden.
- **Benutzerfreundlichkeit**: Der Umgang mit wissensbasierten Systemen erfordert kein programmiertechnisches Vorwissen.
- **Kompetenz**: Wissensbasierte Systeme verfügen in ihrem Anwendungsbereich über eine hohe Problemlösungsfähigkeit.

3.6.2.1 Erkenntnisgewinn

Der Regelinterpreter kann neben dem Ausführen von Regeln durch die Verkettung von Regeln dazu genutzt werden neue Fakten zu Gewinnen und unbekannte zusammenhänge aufzuzeigen. So kann z.B. herausgefundenen werden, warum Ausschuss produziert wird. Folgende Regeln sind dazu definiert:

1. **WENN** eine hohe Temperatur herrscht **DANN** verformen sich Werkstücke.
2. **WENN** sich Werkstücke verformen **DANN** sind diese Ausschuss.

Regeln können durch zwei Verfahren verkettet werden:

- **Vorwärtsverkettung:** Bei der Vorwärtsverkettung können wir folgern, dass bei einer hohen Temperatur sich Werkstücke verformen und dadurch Ausschuss sind.

  > „It is a useful strategy to use when there are a small number of initial conditions but a large number of possible solutions“ [DeT89]

- **Rückwärtsverkettung:** Bei der Rückwärtsverkettung hingegen versucht man herauszufinden, wie es zu einem erreichten Zustand gekommen ist. In diesem Fall die Produktion von Ausschuss. Ausschuss wird produziert, wenn sich Werkstücke verformen und durch die erste Regel wissen wir außerdem, dass sich Werkstücke verformen, wenn eine hohe Temperatur herrscht. Wir können somit folgern, dass Ausschuss produziert wird, wenn die Temperatur hoch ist.

3.7 Zusammenfassung

Innerhalb dieses Kapitels wurden die Grundlagen von MAS vorgestellt. Dabei wurde besonders auf die Kommunikation und Vernetzung der einzelnen Agenten untereinander eingegangen. Darüber hinaus wurde eine Abgrenzung zu Sensornetzwerken und der Microservice-Architektur vorgenommen. Zusätzlich wurde verschiedene Möglichkeiten zur Abbildung von Geschäftslogik vorgestellt.

Kapitel 4

MAS Softwarekonzept

Innerhalb dieses Kapitels wird das Softwarekonzept aufgezeigt. Dazu werden zunächst die getroffenen Entscheidungen erläutert, die zu dem entwickelten Konzept führten. Im Weiteren Verlauf wird das Konzept und die damit verbundene Aufteilung der Komponenten vorgestellt. Außerdem werden die Aspekte der Verteilung der Software, der Sicherheit und der Vernetzung näher betrachtet.

4.1 Entwurfsentscheidungen

Im Folgenden werden zunächst die getroffenen Entwurfsentscheidungen erläutert.

4.1.1 Verteilung des Systems

In einem KMU sind die Ressourcen zum Produzieren von Gütern räumlich verteilt. Diese räumliche Verteilung und die Tatsache, dass an jeder dieser Ressource Sensordaten erfasst und aufbereitet werden sollen, sodass ein Einblick in den Produktionsprozess ermöglicht werden kann, erfordert ein verteiltes System. Daher soll das zu entwickelnde System aus einer beliebigen Anzahl von Agenten, jedoch mindestens einem, bestehen. Pro Arbeitsplatz wird ein Agent eingesetzt, welcher mit zusätzlicher Sensorik und Interaktionsmöglichkeiten ausgestattet ist. Diese Agenten formen ein MAS. Um den, an das System gestellten Anforderungen (vgl. Kapitel 2.3) gerecht zu werden, müssen diese Agenten miteinander kommunizieren.

4.1.2 Message oriented middleware

Die Kommunikation der Agenten untereinander soll über eine Message oriented Middleware (MOM) erfolgen. Daniel A. Menascé diskutiert in seinem Paper MOM vs. RPC [Men05] die Vor- und Nachteile einer MOM gegenüber der funktionsbasierten Kommunikation. Durch eine MOM ist die Kommunikation asynchron und robuster [Men05], die einzelnen Agenten sind loser gekoppelt und darüber hinaus kann über Publish-Subscribe Mechanismen eine Many-to-Many Kommunikation umgesetzt werden.

4.1.3 Webanwendung

Die Agenten sollen als Benutzeroberfläche jeweils eine Webanwendung bereitstellen. Webanwendungen können von einer Vielzahl von Endgeräten genutzt werden, da diese nur einen Webbrowser benötigt. Dadurch kann das Hardwarekonzept flexibler gestaltet werden. Webanwendungen liegt eine Client-Server Architektur zugrunde. Der Agent repräsentiert den Server, der geeignete REST-Schnittstellen zur Verfügung stellt, um eine Oberfläche bereitzustellen. Das Prinzip von REST wird im Folgenden näher erläutert.

4.1.3.1 Representational State Transfer (REST)

Bei der Entwicklung des MAS wurde auf die Prinzipien des Representational State Transfer (REST) wert gelegt. Diese Prinzipien wurden von Roy T. Fielding in seiner Dissertation *Architectural Styles and the Design of Network-based Software Architectures* [Fie00] im Jahre 2000 definiert. Diese Prinzipien beachten besondere Randbedingungen die im Folgenden erläutert werden.

Trennung von Client & Server Unter dem Stichwort *Separation of concerns* (Trennung von Belangen) sollen die Belange von Client & Server getrennt werden um für den Client die Portabilität auf andere Plattformen und die Skalierbarkeit der Server Komponente zu ermöglichen. Zusätzlich ermöglicht dies die unabhängige Erweiterung der Komponenten.

Zustandslosigkeit Die Kommunikation zwischen Server & Client muss zustandslos sein. Das Bedeutet, dass bei jeder Anfrage alle benötigten Informationen an den Server geschickt werden, damit dieser entsprechend antworten kann. Auf dem Server wird für den Client kein Zustand vorgehalten. Jeglicher Zustand befindet sich auf der Client-Seite.

Cache Daten innerhalb einer Antwort des Servers können mit Cache Informationen versehen werden. Es kann angegeben werden, ob es sich um cachebare Daten handelt oder nicht. Dies ermöglicht dem Client einen Cache aufzubauen da die erhaltenen Daten ggf. wiederverwendbar sind. Dadurch ist es möglich die Effizienz zu erhöhen und die Zugriffe auf Netzwerk-Ressourcen zu minimieren.

Einheitliche Schnittstelle Für den Zugriff auf den Server wird eine einheitliche Schnittstelle bereitgestellt. Dies resultiert in der Entkopplung der bereitgestellten Schnittstelle und der Implementierung des Servers und ermöglicht eine unabhängige Entwicklung. Zu erwähnender Nachteil ist die verringerte Effizienz, da keine spezifische Schnittstellen für spezifische Anfragen bereitgestellt werden.

Hierarchische Schichten Falls notwendig, kann das System in hierarchische Schichten aufgeteilt werden. Dabei ist die Besonderheit, dass Komponenten, die mit einer Schicht kommunizieren, kein Wissen über die Strukturen hinter dieser Schicht haben.

Dies resultiert in einer vereinfachten Kommunikation mit den verschiedenen Schichten. So kann z.B. eine direkte Kommunikation mit veralteten System verhindert oder auch eine Lastverteilung zwischen geschaltet werden.

Code-On-Demand Clients soll es möglich sein, ihre eigene Funktionalität durch das zusätzliche Herunterladen und Ausführen von Code zu erweitern. Ein Beispiel hierfür sind so genannten Single Page Applications (SPA). Bei Single Page Applications handelt es sich in der Regel um in JavaScript geschriebene Web-Anwendungen. Beim Aufruf der Anwendung über den Webbrowser lädt dieser den kompletten Anwendungscode herunter und führt diesen aus.

Der Kernaspekt einer REST Architektur sind Ressourcen. Eine Ressource kann z.B. ein Dokument, ein Bild, eine Person oder ähnliches sein. Eine Ressource kann dabei über verschiedene Repräsentationen dargestellt werden, mögliche Beispielen wären JSON, XML, und HTML. So werden die selben dahinter liegenden Daten in verschiedenen Formen dargestellt. Darüber hinaus sind Ressourcen immer eindeutig adressierbar.

Durch den Einsatz von REST kann die Benutzeroberfläche mit einer beliebigen Technologie umgesetzt werden. Diese muss nur in der Lage sein Hyper Text Transfer Protokoll (HTTP)-Anfragen zu stellen. Die Kommunikation in Webanwendungen ist meist heterogen, der HTTP-Client stellt eine Anfrage, die vom Server beantwortet wird. In einem Ereignis getriebenen Kontext, wie dem eines KMU, ist eine homogene Kommunikation unerlässlich, um einen HTTP-Client über Ereignisse zu informieren. Diese Kommunikation kann über Websockets umgesetzt werden.

4.1.3.2 Websockets

Websockets sind sowohl ein Protokoll als auch eine vom World Wide Web Consortium standardisierte API [Fet11]. Es sind Full-Duplex-Verbindungen auf TCP-Basis. Wie TCP auch, sind Websockets streamorientiert. Es handelt sich bei Websockets jedoch um einen Stream von Nachrichten und nicht von Bytes. Durch Websockets ist es möglich, aus einem Webbrowser eine asynchrone und bidirektionale Verbindung aufzubauen [WSM13]. Der Server kann auf dem selben Port HTTP- und Websocket-Verbindungen anzunehmen, da Websocket einen HTTP-kompatiblen Handshake benutzen.

4.1.4 Regelbasiertes Expertensystem

Aufgrund der Anforderungen Expertenwissen, in die Geschäftslogik des Agenten einfließen lassen zu können und dem Ziel ein System zu schaffen, das in verschiedenen KMU eingesetzt werden soll, soll die Geschäftslogik mithilfe von Regeln abgebildet werden können. Dies ermöglicht das System auf die Gegebenheiten eines Unternehmens anzupassen, ohne den Quelltext verändern zu müssen. Dabei sollen Regeln folgendermaßen aufgebaut sein:

WENN Event UND Bedingung DANN Action

Ereignisse im Produktionskontext, wie z.B. die Meldung des Auftragstatus oder das Messen eines Sensorwertes stellen Events dar. Actions stellen die Ausführung einer gewissen Geschäftslogik dar.

4.1.5 Aufteilung in Komponenten

Ein Kernaspekt des Softwarekonzeptes ist der Aufbau eines flexiblen Gesamtsystem, welches das nachträgliche Hinzufügen von Funktionalitäten ermöglicht. Daher soll das System in verschiedene Komponenten aufgeteilt werden. Ein Agent ist in insgesamt fünf Komponenten aufgeteilt, siehe Abbildung 4.1. Jede Komponente ist dabei für einen bestimmten Teilbereich zuständig. Für jede Komponente sind die möglichen Events, sowie die möglichen Tätigkeiten in Form von Actions, definiert. Nach auslösen eines Events wird in einem (Rule Engine) überprüft, in welcher Form andere Komponenten auf das Event reagieren sollen.

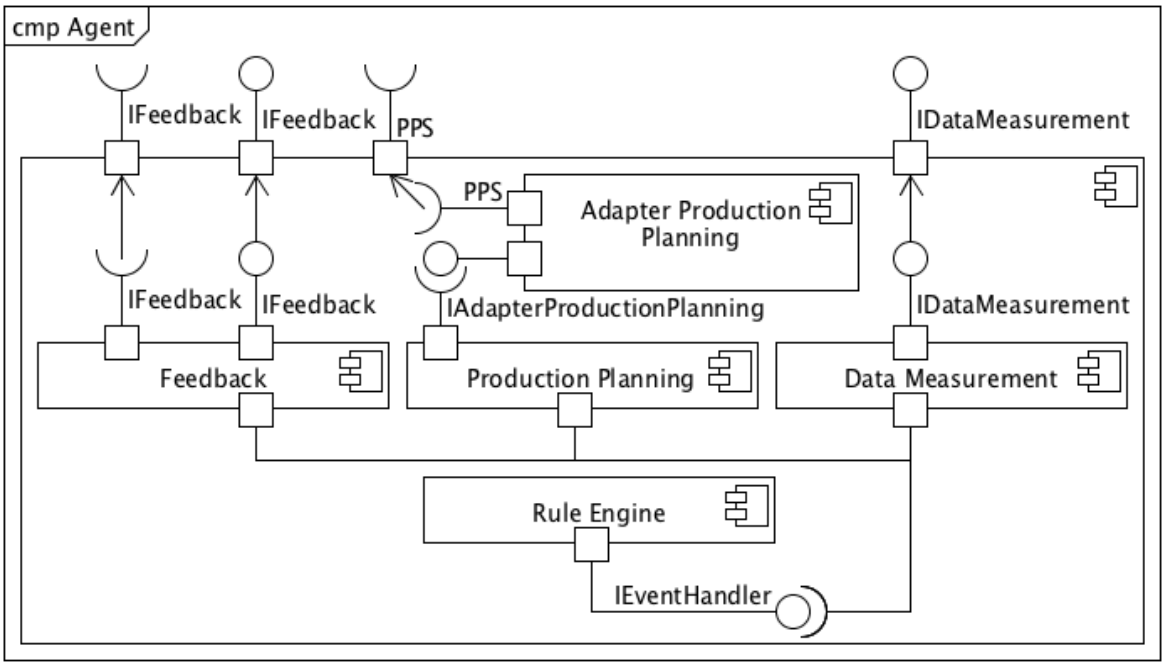


Abbildung 4.1: Komponentendiagramm eines Agenten

4.2 Komponenten

Im Folgenden werden die Zuständigkeiten der einzelnen Komponenten näher Beschreiben mit den zugehörigen Events und Actions.

4.2.1 Data Measurement

Die Data Measurement Komponente ist für das sammeln von Sensordaten während des Produktionsprozesses verantwortlich wie z.B. Stromverbrauch, Temperatur oder Geräusche. Sie bietet für externe Sensoren eine Schnittstelle um Sensordaten zu erhalten.

Events

Das '*dataMeasured*' Event wird ausgelöst, nachdem Sensordaten von externen Sensoren erhalten wurden.

Actions

Keine

Schnittstellen

Zum Erhalten der Sensordaten von externen Sensoren wird eine HTTP-basierte Schnittstelle angeboten. Diese erlaubt es den externen Sensoren über einen POST-Request die gemessenen Daten an die Data Measurement Komponente zu senden. Dabei muss ein Objekt in Form von M2MDevicePortsData übergeben werden. Die Schnittstelle ist über '*/api/dataMeasurement/ports*' zu erreichen. Die Schnittstelle in UML Form kann Abbildung 4.2 entnommen werden.

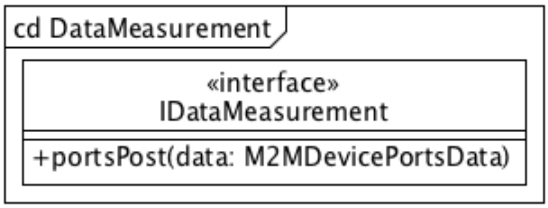


Abbildung 4.2: Data Measurement Schnittstelle

Datenmodell

Das Datenmodell zum Empfangen von Sensordaten der externen Sensoren ist insgesamt in drei Klassen aufgeteilt.

Parameter	Datentyp	Beschreibung
deviceId	String	Enthält den eindeutigen Bezeichner des externen Sensors.
senorName	String	Der Name des Sensors. Ein externen Sensor kann aus mehreren tatsächlichen Sensoren bestehen.
senorPort	Integer	Die Port Nummer an dem der Sensor angeschlossen ist.
mode	M2MDevicePortDataMode	Angabe um welchen Modus es sich bei einem digitalen Port handelt.
timestamp	Timestamp	Der Zeitpunkt an dem die Sensordaten gemessen wurden.
value	Integer	Der tatsächlich gemessene Wert.

Tabelle 4.1: Data Measurement M2MDevicePortData

Parameter	Datentyp	Beschreibung
aports	M2MDevicePortData Array	Enthält die Daten der analogen Ports.
dports	M2MDevicePortData Array	Enthält die Daten der digitalen Ports.

Tabelle 4.2: Data Measurement M2MDevicePortsData

- M2MDevicePortDataMode ist eine Enumeration, welche ausdrückt, um welchen Modus es sich bei einem digitalen Port handelt. Digitale Ports können in zwei unterschiedlichen Modi betrieben werden Digital & Counter. Analoge Ports haben keine unterschiedlichen Betriebsmodi.

- M2MDevicePortData enthält die eigentlichen Informationen vom externen Sensor, siehe Tabelle 4.1.

- M2MDevicePortsData enthält die Daten für die digitalen und analogen Ports, siehe Tabelle 4.2.

Das Datenmodell in UML Form kann der Abbildung 4.3 entnommen werden.

4.2.2 Production Planning

Die Production Planning Komponente hat die Aufgabe der Visualisierung des Produktionsprozesses. Es sollen die Aufträge dargestellt werden, die einer Maschine zugewiesen wurden. Über diese Komponente soll der Status eines Auftrages oder einer Maschine an des PPS-System übermittelt werden.

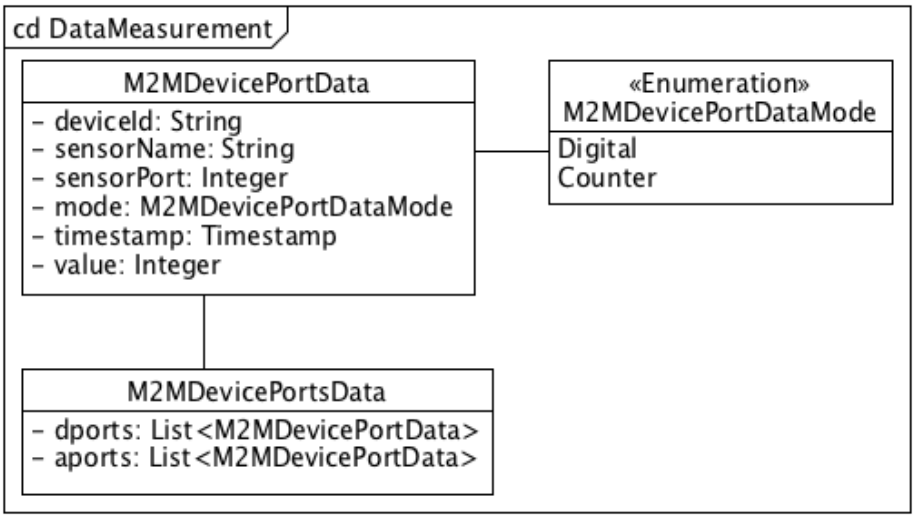


Abbildung 4.3: Data Measurement Datenmodell

Events

- '*jobAssigned*' nach dem ein neuer Auftrag zugewiesen wurde.
- '*jobChanged*' nach dem ein zugewiesener Auftrag sich geändert hat.
- '*jobCanceled*' nach dem ein zugewiesener Auftrag abgebrochen wurde.

Actions

- '*writeData*' schreibe Sensordaten in das PPS-System.
- '*deliverFeedback*' Rückmeldung über den aktuellen Stand des Produktionsprozesses an das PPS-System liefern.

Schnittstellen

Die Komponente definiert die Schnittstelle IAdapterProductionPlanning (vgl. Abbildung 4.4) diese bietet alle nötigen Funktionen um den Produktionsprozess zu visualisieren und Feedback zu senden.

Datenmodell

Die Komponente definiert ein generisches Datenmodell für den Produktionsprozess (vgl. Abbildung 4.5):

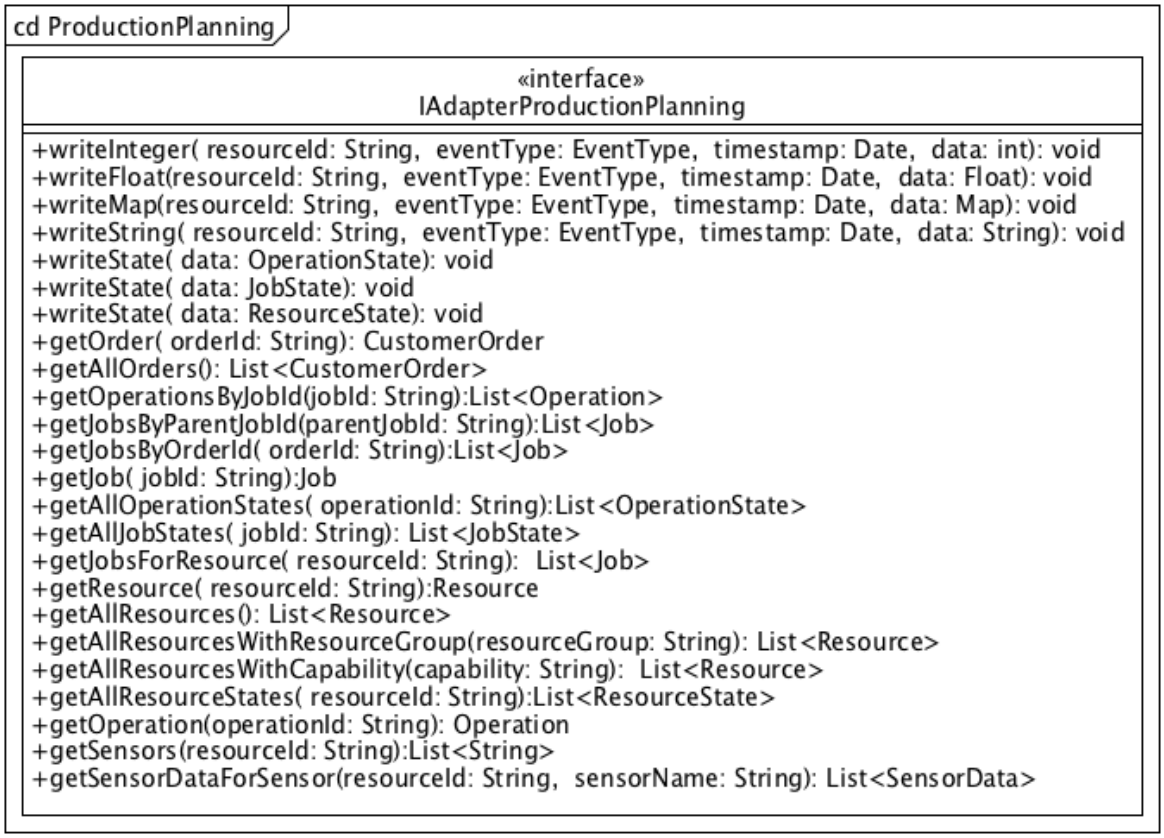


Abbildung 4.4: Interface IAdapterProductionPlanning

- **CustomerOrder**: Die Klasse CustomerOrder stellt einen Kundenauftrag dar. Dieser besteht aus beliebig vielen Fertigungsaufträgen (**Job**).

- **Job**: Ein Job stellt einen Auftrag dar. Diese Klasse ist abstrakt und es existieren zwei Ausprägungen. Der Fertigungsauftrag (**Task**) und die Wartung (**Maintenance**). Ein Job kann aus mehreren Jobs und aus Arbeitsgängen (**Operation**) bestehen. Außerdem können Zeichnungen (**JobDrawing**) hinterlegt werden.

- **Task:** Die Klasse Task stellt einen Fertigungsauftrag dar. Es ist ein Auftrag zum fertigen eines Gutes.

- **Maintenance:** Ein Auftrag mit der Ausprägung Maintenance ist ein Wartungsauftrag. Es wird zwischen einer planmäßigen und unplanmäßigen Wartung unterschieden.

- **Operation**: Ein Arbeitsgang ist Teil eines Auftrages (**Job**). Ein Arbeitsgang besteht unter anderem aus einer Rüstzeit, einer Vorlaufzeit, einer Fertigungszeit sowie einer Fertigungsmenge.

- **JobDrawing**: Ein JobDrawing stellt eine technische Zeichnung eines Auftrages (**Job**) dar.

- **Resource**: Eine Resource stellt ein Arbeitsmittel, wie z.B. eine Maschine oder einen Arbeitsplatz, dar. Eine Resource verfügt über mehrere Fähigkeiten (**Capability**).

- **Capability**: Eine Capability stellt die Fähigkeit eines Arbeitsmittels (**Resource**) dar. Sie gibt an, was mithilfe des Arbeitsmittels gefertigt werden kann.

- **SensorData**: SensorData sind Sensordaten, die während des Produktionsprozesses erfasst wurden, diese können sowohl einer **Resource** als auch einem Fertigungsauftrag (**Job**) zugeordnet werden.

- **ResourceState**: Über die Klasse ResourceState kann der Status eines Arbeitsmittels (**Resource**) propagiert werden. So kann beispielsweise mitgeteilt werden, dass das Arbeitsmittel gerade gewartet wird.

- **OperationState**: OperationState stellt eine Statusmeldung für einen Arbeitsgang (**Operation**) dar. Es können beispielsweise der Fortschritt des Arbeitsgangs, und der produzierte Ausschuss angegeben werden.

- **JobState**: Die Statusmeldung eines Auftrages (**Job**) kann über die Klasse JobState erfolgen. Hierüber können Aufträge beispielsweise in Bearbeitung genommen werden. Es ist unerheblich, ob es sich hierbei um einen Fertigungs- oder Wartungsauftrag handelt.

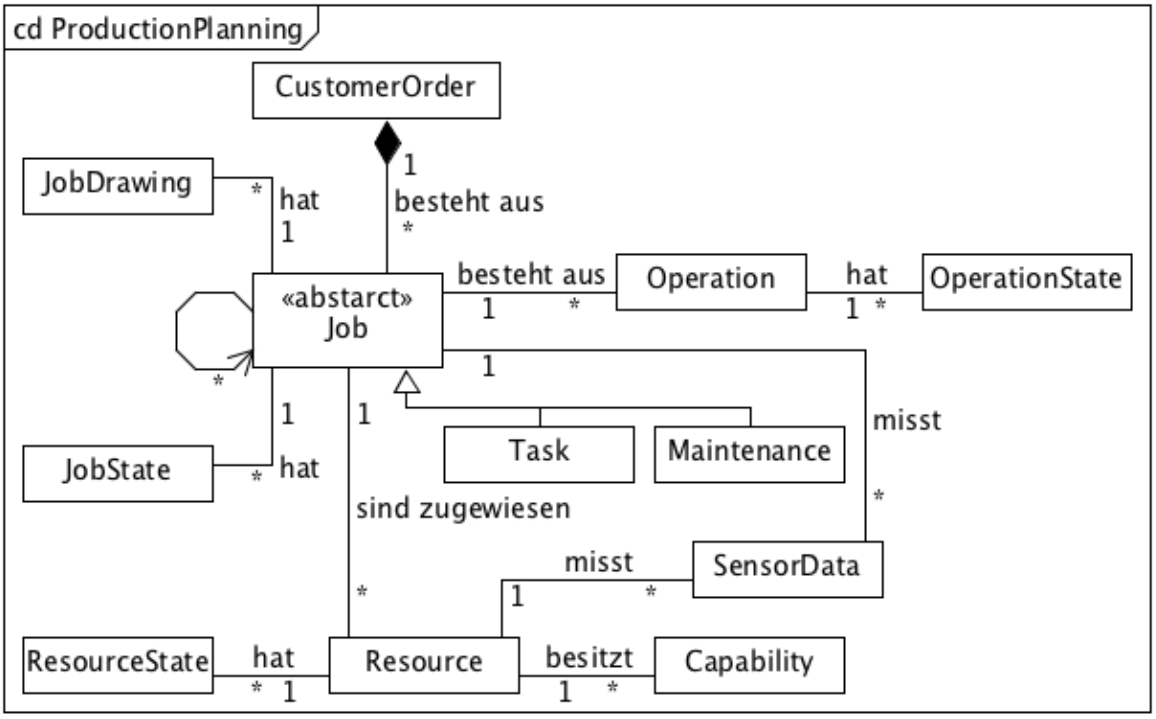


Abbildung 4.5: Klassendiagramm des Datenmodells, das den Produktionsprozess abbildet.

4.2.3 Adapter Production Planning

Die Adapter Production Planning Komponente ist verantwortlich für die Kommunikation mit dem PPS-System. Die Komponente erhält Daten, wie aktuelle Aufträge vom PPS-System und liefert Daten wie den aktuellen Stand des Produktionsprozessen und Sensordaten an das PPS-System. Diese Komponente ist dabei jeweils an das eingesetzt PPS-System angepasst. Dazu wird das von der Production Planning Komponente definierten Interface IAdapterProductionPlanning implementiert.

Events

Keine

Actions

Keine

Schnittstellen

Die Komponente implementiert die Schnittstelle IAdapterProductionPlanning und konsumiert die angebotenen Schnittstellen vom PPS-System.

Datenmodell

Das Datenmodell ist abhängig von dem anzusprechenden PPS-System. In dieser Komponente findet eine Anpassung des Datenmodells des PPS-Systemes auf das Datenmodell der Production Planning Komponente statt.

4.2.4 Feedback

Die Feedback Komponente soll dem Benutzer des Systems die Möglichkeit bieten Feedback zu geben und ihn über Ereignisse zu benachrichtigen. Die Ereignisse sollen auch von anderen Agenten des MAS stammen können. Außerdem soll die Feedback Komponente eine REST-Schnittstelle bereitstellen, über die Ereignisse von außen angestoßen werden können.

Die Komponente übernimmt dazu zwei Funktionen. Zum einen bietet sie Actions an, die es ermöglichen Events an andere Agenten weiterzuleiten und zum anderen bietet sie ein Event an, das angestoßen wird, sobald ein Ereignis über die REST-Schnittstelle oder von einem anderen Agenten empfangen wurde.

Bei der Weiterleitung von Events soll es möglich sein ein Event an einen oder an alle Agenten, deren Benutzer eine bestimmte Rolle einnehmen (Techniker, Meister, Mitarbeiter), weiterzuleiten. Bei der Weiterleitung an einen Agenten, soll das MAS autonom entscheiden, welcher Agent informiert wird. Dazu soll eine Auktion nach dem Trendverfahren (vgl. 3.2.4 stattfinden. Diese Art der Abstimmung wurde gewählt, da sie die am simpelsten

umzusetzende ist. Damit Events an andere Agenten weitergeleitet und von anderen Agenten empfangen werden können, soll mithilfe eines MessageBrokers ein nachrichtenbasierte Kommunikation etabliert werden. Es sollen sogenannte Topics genutzt werden, an die sich alle Agenten registrieren und in denen Nachrichten abgelegt werden können. Die Agenten können diese Nachrichten sofort oder zu einem späteren Zeitpunkt abrufen, so dass diese die Ereignisse erhalten, auch wenn sie nicht permanent mit dem MessageBroker verbunden sind.

Um den Benutzer über Ereignisse informieren zu können, sollte der Server dem Client Nachrichten schicken können. Der eingesetzten Client-Server-Architektur und den REST-Schnittstellen über HTTP liegt eine heterogene Kommunikation zugrunde, da nur der Client Anfragen an den Server stellen kann [Haa08]. Mithilfe von Websockets kann dieses Problem gelöst werden, deshalb wird in der Feedback Komponente eine Web-Socket Schnittstelle vorgesehen.

Die Komponente definiert folgende Actions und Events:

Events

Mögliche Events:

- '*eventReceived*': Dieses Event wird angestoßen, sobald ein Event über die REST-Schnittstelle oder über den MessageBroker empfangen wurde.

Actions

Mögliche Actions:

- '*showInformation*' zeigt, dem Benutzer Benachrichtigungen an. Dabei werden die Informationen eines Events so aufbereitet, dass diese für den Benutzer verständlich sind.
- '*sendEventToAllExperts*' durch diese Action wird ein Event an alle Meister weitergeleitet. Dabei wird immer das Event genutzt, welches die Ausführung der Regel angestoßen hat.
- '*sendEventToAllMaintainers*' sendet ein Event an alle Techniker.
- '*sendEventToAllWorkers*' sendet ein Event an alle Mitarbeiter.
- '*sendEventToOneExpert*' Ein Ereignis wird nach einer vorherigen Auktion an einen Meister weitergeleitet.
- '*sendEventToOneMaintainer*' Ein Ereignis wird nach einer vorherigen Auktion an einen Techniker weitergeleitet.
- '*sendEventToOneWorker*' Ein Ereignis wird nach einer vorherigen Auktion an einen Mitarbeiter weitergeleitet.

Schnittstellen

Die Komponente verfügt über folgende Schnittstellen:

- **MessageBroker:** Der MessageBroker stellt eine Schnittstelle zu den anderen Agenten dar, dabei sollen Topics genutzt werden. Die Topics werden durch einen Namen identifiziert, der sich aus der Rolle und der Anzahl der zu informierenden Agenten zusammensetzt:
  - worker.all
  - worker.one
  - expert.all
  - expert.one
  - maintainer.all
  - maintainer.one

  Wenn nur ein Agent informiert werden soll, findet eine Auktion statt. Die Auktion soll nicht mithilfe einer Topic abgewickelt werden, da es sich um eine 'sealed-bit' Auktion handelt, bleiben die einzelnen Gebote geheim. Es sollen daher diverse temporäre Queues zwischen dem Auktionator, dem Agenten, der ein Event weiterleiten will, und den Bietern, alle Agenten mit einer bestimmten Rolle, genutzt werden.

- **Web-Socket** Die Schnittstelle um den Benutzer über Ereignisse informieren zu können ist mithilfe von Web-Sockets realisiert. Wenn sich der Client für das Topic "/api/-feedback/webSocket/topic/showInformation"registriert, werden ihm alle Ereignisse (Events) gesendet für die eine entsprechende Regel mit '*showInformation*' als Action existiert.

- **REST-Schnittstelle** Über die REST-Schnittstelle können von außen Ereignisse in den Agenten gereicht werden, was notwendig sein kann, wenn externe System angebunden werden sollen. Wenn ein Event an diese Schnittstelle gesendet worden ist, wird das Event '*eventReceived*' ausgeführt.

Datenmodell

Das Datenmodell entspricht dem Datenmodell der Rule Engine Komponente. Das Event wird um eine weiter Ausprägung dem ExternalEvent erweitert. Es verfügt zusätzlich über eine 'ressourceId', die angibt, von welcher Ressource das Event ursprünglich ausging.

4.2.5 Rule Engine

Die Rule Engine Komponente ist verantwortlich zu prüfen ob für die ausgelösten Events Regeln vorhanden sind und löst ggf. die zugeordneten Actions aus. Nach dem innerhalb

eines Agenten ein Event ausgelöst wird, wird die Schnittstelle IEventHandler der RuleEngine angesprochen. Diese Schnittstelle nimmt das Event entgegen und überprüft ob eine zutreffende Regel vorhanden ist und löst eine Action von dem, in der Regel definierten, ActionType aus.

Nach dem Eintreffen eines Events wird überprüft um welchen EventType es sich handelt. Dazu wird der EventType aus dem Event extrahiert. Aufgrund des EventTypes wird überprüft ob Regeln für diesen EventType vorhanden sind. Anschließend wird überprüft ob die mitgesendeten Daten auf die, in den Regeln definierten, Expressions zutreffen. Dabei ist es möglich, dass ein Event auf mehrere Regeln zutrifft und mehrere Actions ausgeführt werden. Sollte nun eine Regel zutreffen wird aufgrund des hinterlegten ActionTypes eine neue Action mit diesem ActionType ausgeführt. Daraus resultiert, dass die verschiedenen Actions der vorhandenen Komponenten ausgeführt werden. Der Ablauf als Sequenzdiagramm kann Abbildung 4.6 entnommen werden.

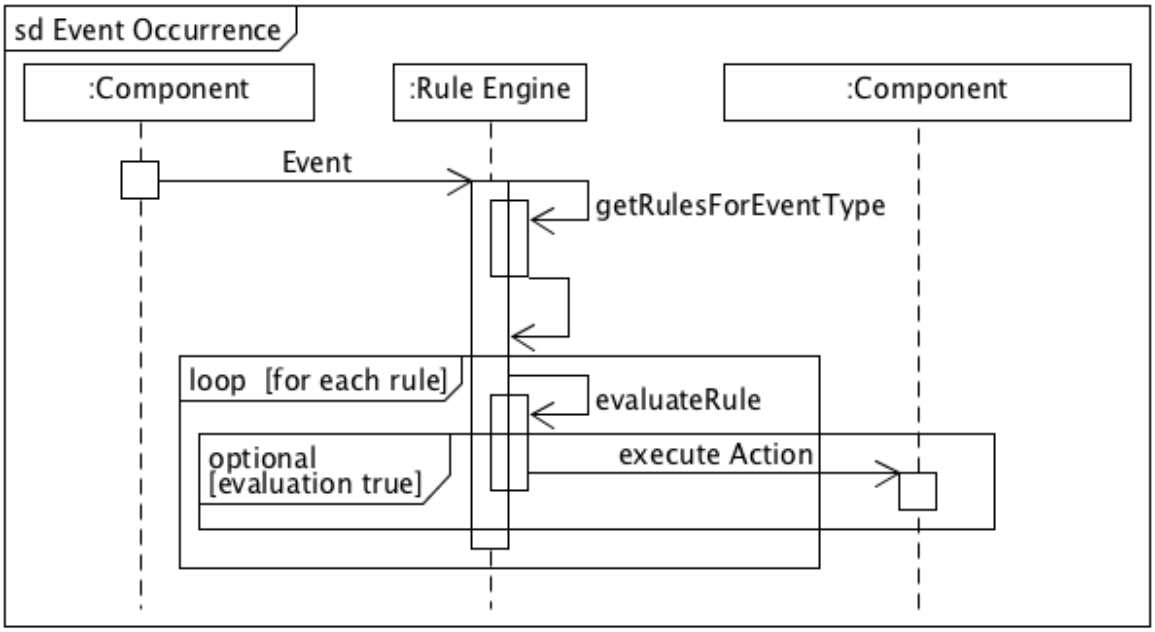


Abbildung 4.6: Rule Engine Ablaufdiagramm

Aufbau einer Regel Expression

Der Aufbau einer Regel Expression ist von hoher Relevanz für die Softwarelösung. Es ist notwendig dem späteren Benutzer die Möglichkeit zu geben das Mitarbeiterwissen in Form von Regeln ausdrücken zu können, daher müssen auch komplexe Sachverhalte abbildbar sein, darüber hinaus darf es jedoch nicht zu kompliziert sein das Wissen abzubilden. Für exakt diesen Anwendungsfall wurde die MVFLEX Expression Language (MVEL) definiert.

Für die Definition der Regel Expressions soll die MVEL eingesetzt werden, daher wird sie im Folgenden näher beschrieben.

Bei der MVEL handelt es sich um eine Sprache zur Definition von Ausdrücken, die innerhalb der Java Plattform einbettbar ist. Ein typischer Anwendungsfall ist die externe Definitionen von Logik durch den Benutzer ohne eine Änderung des Quelltextes vornehmen zu müssen. Die komplette Definition der MVEL kann [Mik16] entnommen werden. Im Folgenden soll exemplarisch der Aufbau und die Verwendung der MVEL dargestellt werden. Zu beachten ist, dass es sich um eine dynamisch typisierte Sprache handelt und auf die Angabe von Datentypen verzichtet wird. Auf die Attribute des „input“-Objektes kann folgendermaßen zugegriffen werden:

```
input.attribute
```

Sollte es sich bei dem Attribut um ein Objekt handeln kann mit „.“ wiederum auf dessen Attribute zugegriffen werden.
Um Attribute mit Werten zu Vergleichen kann man die Operatoren „==“, „>“, „<“, „>=“, „<=“, „!=“ verwenden. Zu beachten ist, dass auf Grund der dynamischen Typisierung der Ausdruck „“123“ == 123“ positiv evaluiert da der Zahlenwert 123 implizit in die Zeichenkette „123“ umgewandelt wird.

```
“123“ == 123; // true
```

Die Verbindung von booleschen Ausdrücken kann mit „&&“ (logisches Und) und „||“ (logisches Oder) realisiert werden. Komplexere boolesche Ausdrücke sind folgendem Beispiel zu entnehmen:

```
input.data.sensorName=="TemperaturSensor1" && (input.data.value > 35 || input.data.value < 10)
```

Diese Expression evaluiert positiv wenn es sich bei den bereitgestellten Daten um den TemperaturSensor1 handelt, welcher einen Temperaturwert größer als 35°C oder kleiner als 10°C liefert. Dies könnten z.B. die vom Hersteller vorgeschriebenen Betriebstemperaturen einer Maschine sein.
Es ist darüber hinaus möglich mehrere Statement zu verwenden und diese jeweils mit einem Semikolon zu trennen. Zur expliziten Ausgabe des Rückgabewertes kann das Schlüsselwort „return“ verwendet werden.

```
    a = input.data.sensorName=="TemperaturSensor1";
b = input.data.value > 35 || input.data.value < 10;
c = a && b;
return c;
```

Die Definition und der Zugriff auf Listen, Maps, und Arrays sieht folgendermaßen aus:

```
    list = ["value1", "value2", "value3"]; // Liste
map = ["TemperaturSensor1": "24", "HumiditySensor": "75"]; // Map
array = {"value1", "value2", "value3"}; // Array

list[1]; // "value2"
map["TemperaturSensor1"]; // "24"
array[1]; // "value2"
```

Zur Flusskontrolle steht ein „if-else-if-else“-Konstrukt zur Verfügung.

```
    if (input.data.userChoice == "yes") {
return true;
}
else if (input.data.userChoice == “accept“) {
return true;
}
else {
return false;
}
```

Events

Keine

Actions

Keine

Schnittstellen

Die Schnittstelle IEventHandler dient dem Entgegennehmen von Events die von Komponenten ausgelöst werden können. Die Schnittstelle in UML Form kann Abbildung 4.7 entnommen werden.

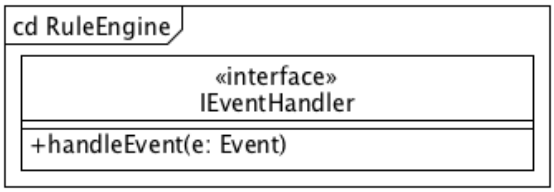


Abbildung 4.7: Rule Engine Schnittstelle

Parameter	Datentyp	Beschreibung
name	String	Name des EventTypes.
eventSource	String	Name der Komponente die diesen EventType auslöst.
dataType	TypeEnum	Angabe um welche Datentyp es sich aus dem TypeEnum handelt.

Tabelle 4.3: Rule Engine EventType

Datenmodell

Für das Zusammenspiel von auftretenden Events, Rules, und den resultierenden Actions wurde folgendes Datenmodell entwickelt. Das TypeEnum ist eine Enumeration und enthält die erlaubten Datentypen die innerhalb eines EventType verwendet werden dürfen. Dies ist notwendig um die maximale Flexibilität beim Absenden von Events zu erreichen jedoch auch die Typensicherheit zu haben wenn sie benötigt wird.

Mit Hilfe von EventTypen kann man Definitionen für auftretende Events angeben, siehe Tabelle 4.3. EventTypen werden benötigt um die Zuordnung zwischen Events und Rules zu ermöglichen.

Ein Event beschreibt das Eintreten eines Events und wird von Komponenten ausgelöst. Innerhalb von Event ist definiert wann dieses Event aufgetreten ist, um welchen EventTypen es sich handelt und die eigentlich Daten des Events, siehe Tabelle 4.4.

Der ActionType gibt an was für ein Typ von Action ausgelöst werden soll, siehe Tabelle 4.6. Die ActionTypes korrespondieren mit den Actions die von den einzelnen Komponenten angeboten werden.

Eine Rule ist definiert über ihren Namen, eine Beschreibung, den EventType auf den sie reagiert, die Expression die zutreffen muss, und dem auszulösenden ActionType, siehe 4.5.

Das Datenmodell in UML Form kann der Abbildung 4.8 entnommen werden.

Parameter	Datentyp	Beschreibung
id	String	Eindeutige Id des Events.
timestamp	Timestamp	Zeitpunkt an dem das Event aufgetreten ist.
eventType	EventType	Angabe um welchen EventType es sich handelt.
data	Object	Die Daten des Events. Der zugehörigen Datentyp ist abhängig von dem EventType.

Tabelle 4.4: Rule Engine Event

Parameter	Datentyp	Beschreibung
name	String	Eindeutiger Name der Regel.
description	String	Beschreibung der Regel.
eventType	EventType	Angabe auf welchen EventType diese Regel reagiert.
expression	String	Zu evaluierender Ausdruck der bestimmt ob diese Regel zutrifft.
actionType	ActionType	Angabe welcher ActionType ausgelöst werden soll wenn die Regel zutrifft.

Tabelle 4.5: Rule Engine Rule

Parameter	Datentyp	Beschreibung
name	String	Eindeutiger Name des ActionType.
actionTarget	String	Name der betroffenen Komponente.

Tabelle 4.6: Rule Engine ActionType

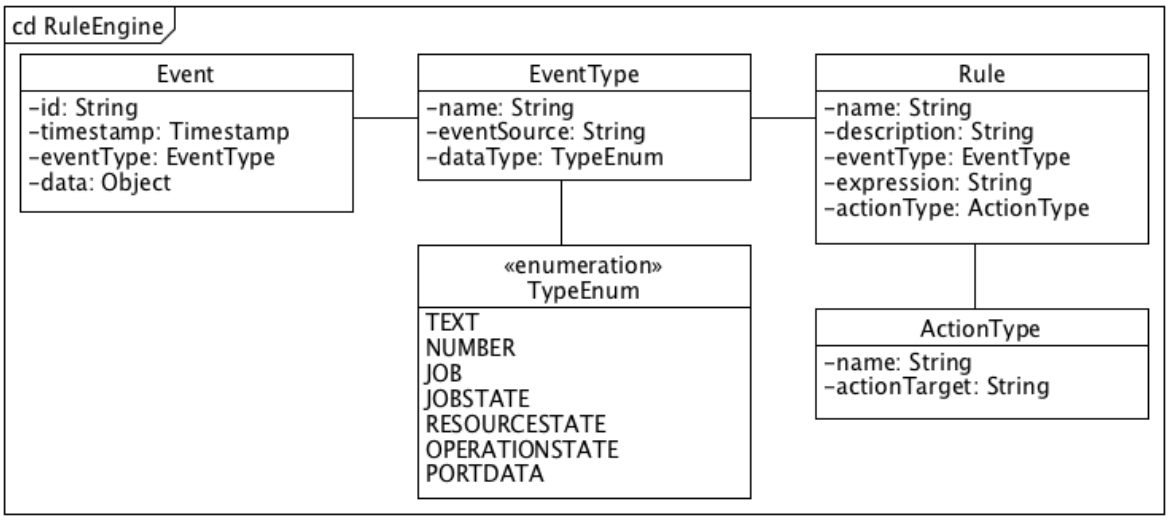


Abbildung 4.8: Rule Engine Datenmodell

4.3 Hardwarekonzept

Das vorgestellte Hardwarekonzept richtet sich nach der vorgestellten Ausgangssituation, siehe Kapitel 2.1, und dem beschriebenen Systemkontext, siehe Kapitel 2.2. Die Ausprägung des Hardwarekonzeptes kann Abbildung 4.9 entnommen werden. Die Ausprägung sieht die Integration der Agenten-Software mit einem manuellen Feedback Mechanismus, der einen Mitarbeiter vorsieht, vor.

Dabei wird der vorhandene Arbeitsplatz mit einem HMI ausgestattet um die Interaktion mit dem System zu ermöglichen. Zusätzlich wird der Arbeitsplatz mit externen Sensoren versehen, die je nach Anwendungsfall, verschiedene Daten aufnehmen.

Die Wahl des verwendeten HMI ist dabei von dem Einsatzszenario abhängig. Eine Möglichkeit wäre die Installation von einem Arbeitsplatz mit Monitor, Maus, und Tastatur, die es dem Mitarbeiter ermöglicht Eingaben zu tätigen. Darüber hinaus wäre auch die Verwendung von einem Tablett denkbar. Durch den modularen Aufbau des Agenten und der Event basierten Kommunikation sind jegliche Kombinationen möglich.

4.4 Benutzeroberfläche

Innerhalb dieses Kapitels werden grafische Vorlagen der Benutzeroberfläche aufgezeigt.

4.4.1 Mitarbeiter Interaktion über HMI

Das Konzept der Interaktion mit dem Mitarbeiter kann den Abbildungen 4.10, 4.11, 4.12, und 4.13 entnommen werden.

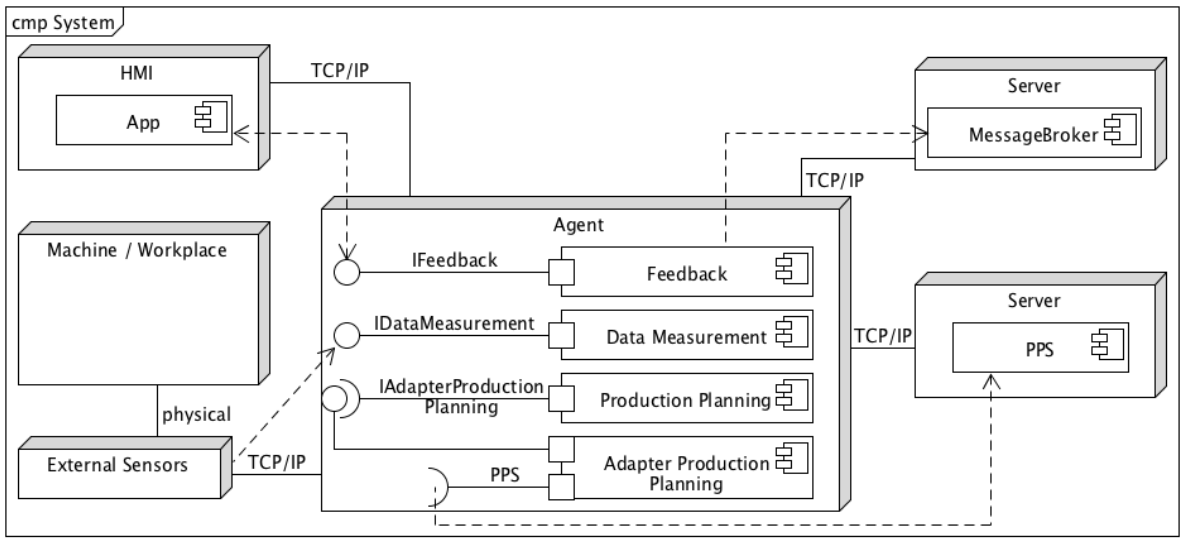


Abbildung 4.9: Komponentendiagramm des Gesamtsystems

Abbildung 4.10 kann die Übersicht der Jobs die dem Arbeitsplatz zugeordnet sind entnommen werden. Dabei hat der Mitarbeiter die Möglichkeit eine Auswahl der aktuell zugeordneten Jobs zu treffen. Dies erlaubt dem Mitarbeit eine gewisse Autonomie in der Gestaltung seiner Arbeitsabläufe. Sollte diese Autonomie nicht gewünscht sein, ist es möglich ausschließlich einen Job dem Arbeitsplatz zuzuordnen.

Durch die Auswahl eines Tabelleneintrags wird die Detailansicht eines Jobs aufgerufen, siehe Abbildung 4.11. Der Mitarbeiter kann nun die Details eines Job einsehen wie die Auftrags Nummer, die Beschreibung, optionale Kommentare, zusätzlich beigefügte Dateien, und eventuelle Unterjobs die dem Job zugeordnet sind.

Die Auswahl der beigefügten Dateien zeigt diese an. Eine Auswahl der Unterjobs führt zu einer Detailansicht dieser. Über „Accept Job“ hat der Mitarbeiter die Möglichkeit den aktuellen Job anzunehmen. Die Metapher den Job zu akzeptieren ist bewusst gewählt um die Autonomie des Mitarbeiters zu fördern.

Nach dem ein Job akzeptiert wurde ändert sich die Ansicht, siehe Abbildung 4.12. Der Job befindet sich nun in Bearbeitung, diese Information wird an das PPS-System zurück gespielt. Verfügt ein Job über Unterjobs müssen zunächst diese in der dargestellten Reihenfolge abgearbeitet werden.

Durch die Auswahl eines Unterjobs und der Betätigung von „Start Job“ signalisiert der Mitarbeiter, dass er mit der Bearbeitung des Unterjobs begonnen hat. Durch Betätigung von „Finish Job“ signalisiert er den Abschluss des Jobs.

Diese Informationen werden jeweils an das PPS-System gesendet um den Einblick in den Produktionsprozesses zu ermöglichen. Es ist nun möglich den aktuellen Status der Jobs mit ihren Unterjobs einzusehen. Sollten alle Unterjobs und der eigentliche Job abgeschlossen sein, wird erneut die Job Übersicht angezeigt und der Mitarbeiter hat die Möglichkeit einen

neuen Job auszuwählen.

Kontinuierlich, sowie während der Bearbeitung der Jobs, werden die erfassten Sensordaten an das PPS-System gesendet. Darüber hinaus hat der Mitarbeit die Möglichkeit die aktuell erfassten Daten über den Reiter „Workplace“ einzusehen, siehe Abbildung 4.13.

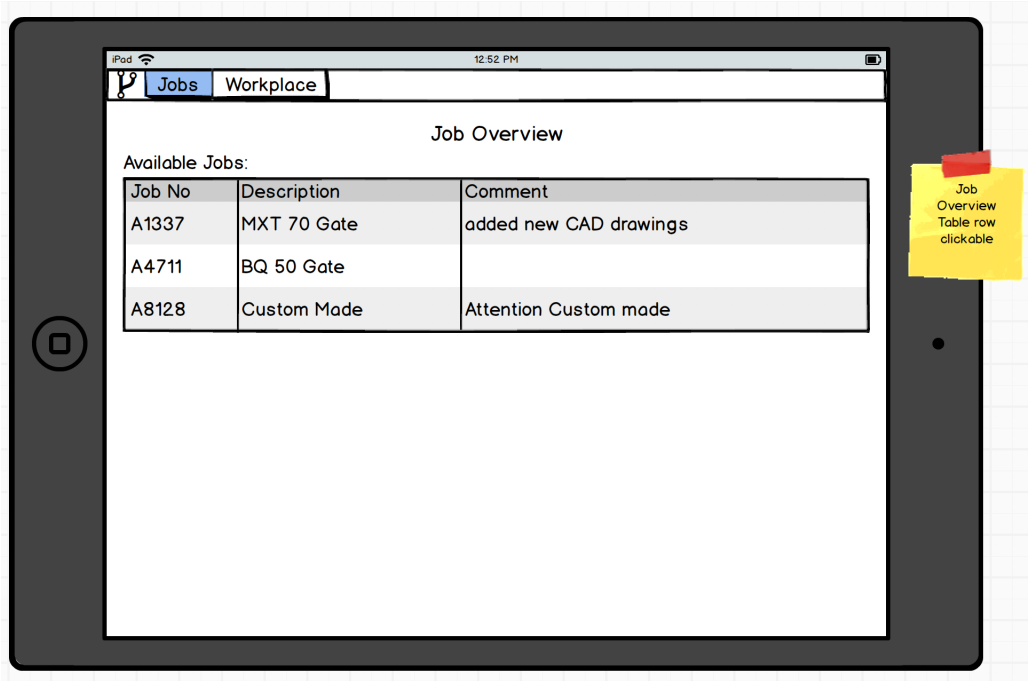


Abbildung 4.10: Übersicht über alle Jobs

4.4.2 Expertensystem

Ein Kernaspekt der Softwarelösung ist der Transfer des Mitarbeiterwissens in die vorhandenen Agenten. Dieser Wissenstransfer soll durch das Anlegen von Regeln realisiert werden. Abbildungen 4.14 und 4.15 kann dabei die grafische Oberfläche entnommen werden.

In Abbildung 4.14 sieht man alle vorhandenen Regeln. Abbildung 4.15 stellt die grafische Oberfläche zum Anlegen neuer Regeln dar. Zunächst wird der Regel ein Name und eine Beschreibung vergeben. Anschließend gibt man an auf welche Art von Event die Regel reagieren soll. An dieser Stelle werden die Events angeboten die von den einzelnen Komponenten ausgelöst werden können. Nun muss der Mitarbeiter eine Expression festlegen die evaluiert wird nach dem das Event eingetreten ist. Danach muss definiert werden, welche Action ausgeführt werden soll, nachdem das Event eingetreten ist und die Expression positiv evaluiert wurde. Bei bereits angelegten Regeln ist es möglich die letzten Events einzusehen die diese Regel ausgelöst haben. Ein einfaches Beispiel für eine Regel wäre, dass nachdem das „DataMeasured“-Event ausgeführt wurde alle Sensordaten in das PPS-System über die Action „WriteData“ geschrieben werden.

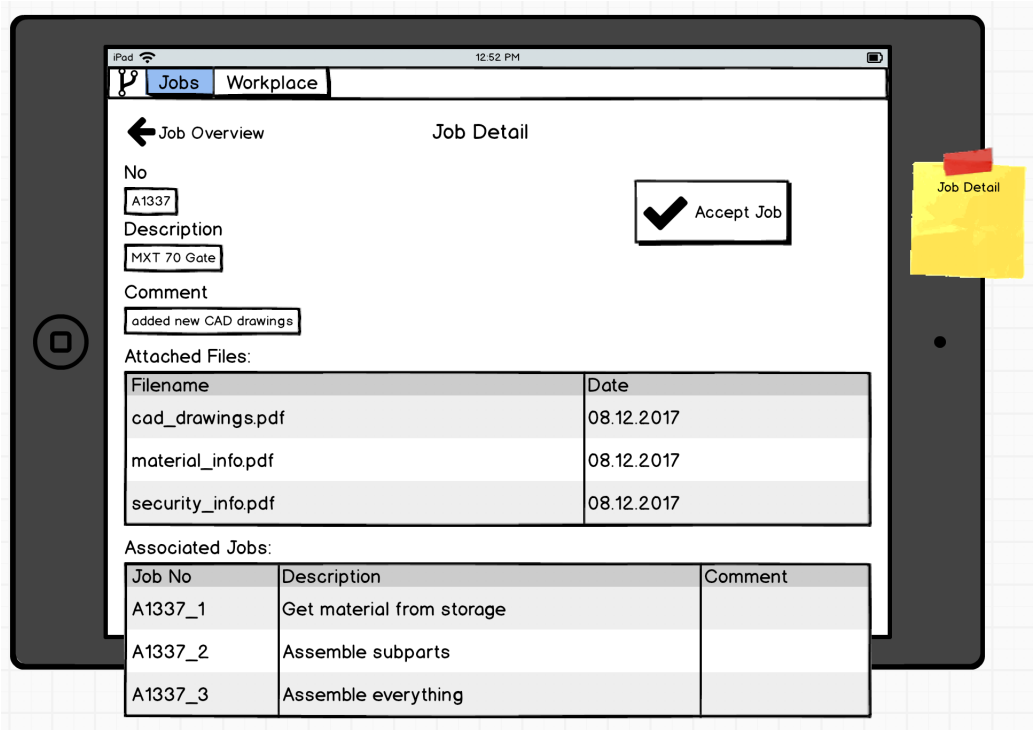


Abbildung 4.11: Detailansicht eines Jobs

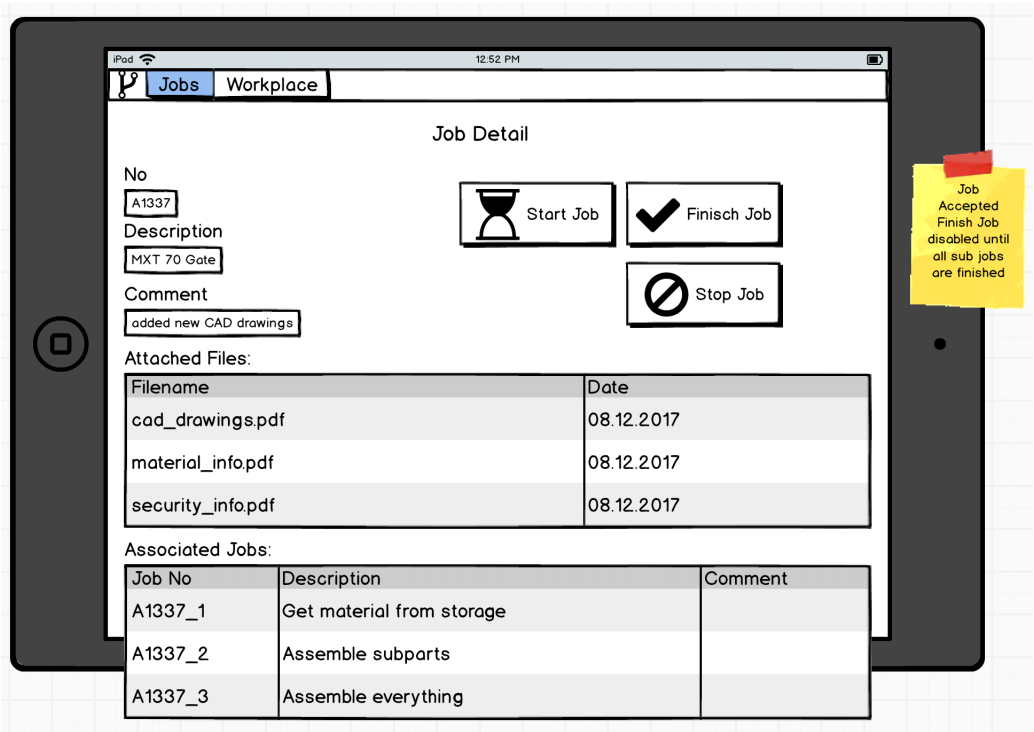


Abbildung 4.12: Aktueller Job in Bearbeitung

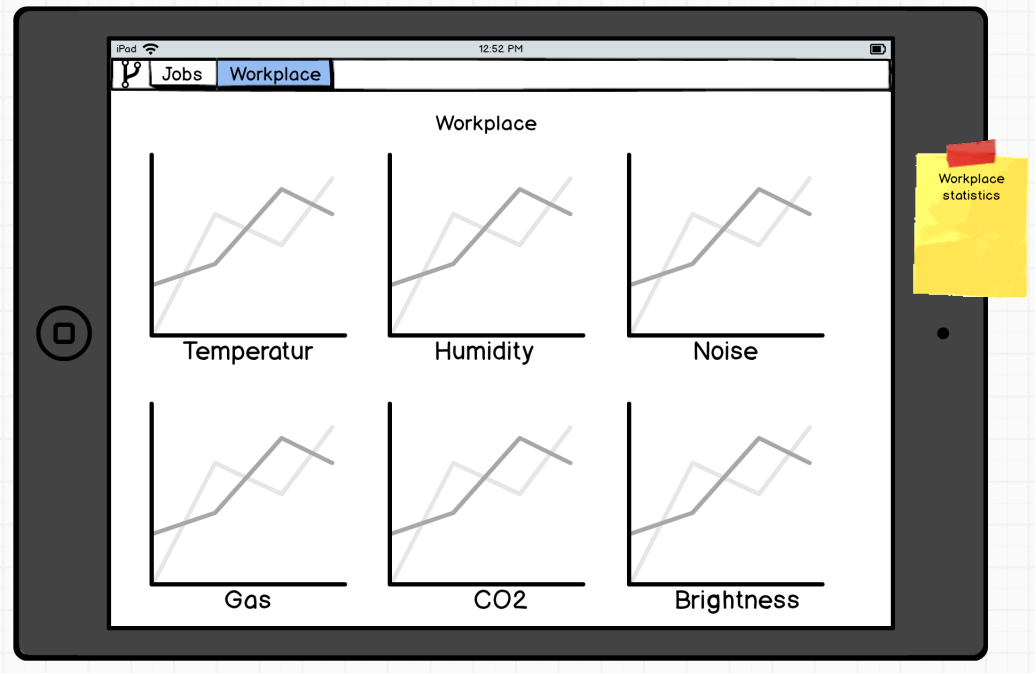


Abbildung 4.13: Übersicht der Sensordaten am Arbeitsplatz

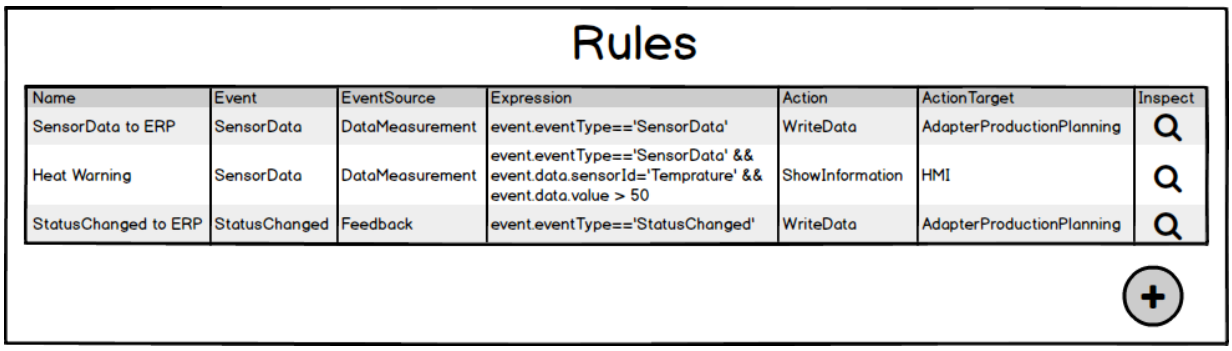


Abbildung 4.14: Übersicht über alle Regeln

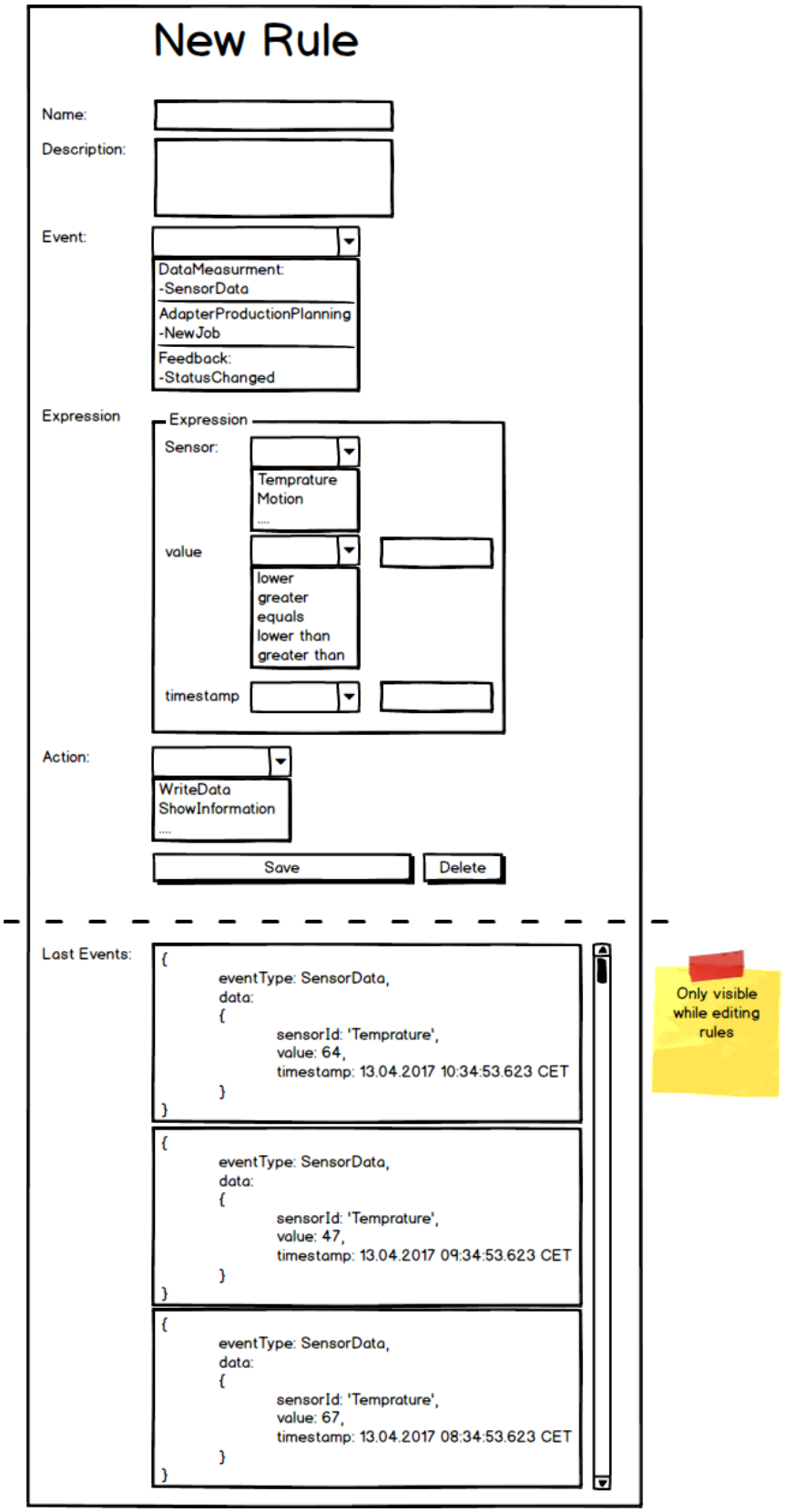


Abbildung 4.15: Neue Regel anlegen

4.5 Sicherheitskonzept

Bei der Entwicklung von Softwarelösungen ist das Thema Sicherheit von essentieller Bedeutung. Es soll das Softwaresystem selbst, sowie die anfallenden Daten vor Missbrauch durch Dritte geschützt werden. Zur Absicherung der Kommunikation zwischen Komponenten besteht die Möglichkeit des Einsatzes von HTTPS [Res00]. Durch das Bereitstellen von Zertifikaten sind die Komponenten in der Lage ihre Daten vor der Übertragung zu verschlüsseln und können nur von dem vorgesehen Empfänger entschlüsselt werden. Durch den Einsatz von HTTPS ist es mit moderaten Mitteln möglich die Kommunikation abzusichern und die anfallenden Daten vor dem Missbrauch durch Dritte zu schützen.

Zusätzlich ist es notwendig den Zugriff auf die Daten regulieren zu können. In der Regel sollte es nicht jedem Mitarbeiter erlaubt sein die Daten einsehen und ändern zu können. Die Lösung dieser Problematik ist die Einführung eines rollenbasierten Benutzerkonzeptes. Für die verschiedenen Funktionen des Systems werden diverse Rollen definiert. Ein Benutzer muss nun über die hinterlegte Rolle verfügen um die Funktion nutzen zu dürfen. Jeder Mitarbeiter der Zugriff auf das System erhalten soll erhält einen Benutzeraccount mit den für ihn vorgesehenen Rollen.

In vielen KMUs sind Systeme zur Verwaltung von Benutzerdaten bereits vorhanden, z.B. durch den Einsatz von Zeiterfassungssystemen oder einer Zugangskontrolle durch Token. Solche Systeme bieten eine optimale Grundlage da es essentiell ist, dass sie über aktuelle Daten verfügen und kontinuierlich gepflegt werden.

4.6 Vernetzung

Ein weiterer Aspekt der betrachtet werden muss ist die Vernetzung der einzelnen Geräte. In der industriellen Produktion gibt es besondere Herausforderungen:

- Störquellen, z.B. elektromagnetische Wellenstrahlung beim Schweißen.
- Überwindung großer Distanzen in Produktionshallen.
- Beweglichkeit der Mitarbeiter bei der Datenerfassung an den Handarbeitsplätzen.

Da der Ausfall des Netzwerkes möglicherweise einen Produktionsstopp nach sich zieht, was zu hohen finanziellen Einbußen führen kann, sollte das Netzwerk robust sein. Ring-, Stern- und Bus-Topologien sind fehleranfällig. Der Ausfall weniger, sogar einzelner Geräte, kann zum Gesamtausfall des Netzwerkes führen. Vorteil der Baum-Topologie ist, dass nur der Ausfall eines Verteilers zum Ausfall eines Teilnetzes führt. Der Ausfall von Endkonten bleibt ohne Konsequenzen. Mesh-Topologien sind noch robuster. Jedoch bräuchte man zur Verkabelung eines vollvermaschten Netzes mit n Knoten $\frac{n*(n-1)}{2}$ Kabel [Fou16]. Ein Netz mit 10 Geräten bräuchte somit 45 Kabel. Daher wird diese Topologie praktisch nie für verkabelte Netzwerke eingesetzt [Mey10].

Kabelgebundene Netzwerke zeichnen sich durch eine höhere Übertragungsrate aus. Sie sind unanfälliger für Störungen und sicherer als ein drahtloses Netzwerk. Ein kabelgebundenes Netzwerk zu etablieren, ist jedoch ein langwierigerer und teurer Prozess und zudem nicht immer praktikabel [AGN+10]. Drahtlose Netzwerke sind flexibler, es lassen sich leichter und schneller neue Geräte in das Netzwerk aufnehmen.

Die Spezifikationen IEEE 802.11 (WLAN) und IEEE 802.15 (WPAN) zur drahtlosen Vernetzung mit unterschiedlichen Reichweiten haben sich gegenüber anderen Standards wie z.B. „High Performance Radio Local Area Network“ (HiperLAN) durchgesetzt und sind in vielen Geräten des täglichen Gebrauchs verbaut. Bis ins Jahr 2012 gab es keinen Standard zum Aufspannen vermaschter Netze mit WLAN. Daher entstanden viele proprietären Implementierungen für Protokolle, die maßgeblich auf dem Basisstandard Wireless Distribution System (WDS) aufbauen, der jedoch viele Fragen offen lässt [Sch]. Der Standard IEEE 802.11s spezifiziert einen neuen Betriebsmodi den Mesh Basic Service Set (MBSS), über den sich ein Mesh-Netzwerk erstellen lässt.

Dieser Standard wird in verschiedenen kommerziellen Produkten, wie zum Beispiel Wifi[1] von Google genutzt. Verschiedene WLAN Router spannen untereinander ein 802.11s Netzwerk auf. Endgeräte können sich über ein 802.11 a/b/g/n/ac Netzwerk mit diesen verbinden. Dadurch kann ein robustes drahtlos Netzwerk entstehen. Der Ausfall eines Routers kann bei geschickter Positionierung durch andere Router aufgefangen werden. Außerdem kann es mehrere Zugangspunkte ins Internet geben. Ein solches Netzwerk in einer industriellen Produktion zu nutzen, um die Agenten miteinander zu vernetzen wäre denkbar, muss jedoch noch näher untersucht werden.

4.7 Zusammenfassung und Fazit

Die Erstellung des Softwarekonzeptes benötigte diverse Entwurfsentscheidungen die im Folgenden dargelegt werden. Zunächst fiel die Entscheidung ein verteiltes System zu realisieren. Dies ist notwendig da im verarbeitendem Gewerbe die Ressourcen zum Produzieren von Gütern räumlich verteilt sind. Durch ein zentrales System ist die Möglichkeit der Erfassung von Sensordaten und die Rückmeldung des Produktionsprozesses an einzelnen Arbeitsplätzen nicht möglich.

Beim Softwarekonzept wurde besonders auf die lose Koppelung der Komponenten und der flexiblen Kommunikation geachtet, dadurch ist es möglich das Softwarekonzept zukunftssicher aufzustellen und das nachträgliche Erweitern der Software zu ermöglichen.

Um die flexible Kommunikation zu ermöglichen wird eine Event-basierte Kommunikation verwendet. Die diversen Komponenten können Events auslösen, diese werden von dem regelbasierten Expertensystem verarbeitet, und können in Anweisungen an andere Komponenten resultieren.

Um das Expertenwissen der Mitarbeiter in das Gesamtsystem einfließen lassen zu können wurde ein regelbasiertes Expertensystem verwendet. Ein regelbasiertes Expertensystem

[1] https://madeby.google.com/wifi/

bietet den Vorteil, dass die benötigte Geschäftslogik durch die Mitarbeiter eingepflegt oder verändert werden kann, ohne die Notwendigkeit den Quelltext anpassen zu müssen.

Damit eine lose Koppelung und die flexible Erweiterbarkeit der Gesamtlösung möglich ist, wurden die diversen Funktionalitäten in Komponenten aufgeteilt. Die Event-basierte Kommunikation ermöglicht es nachträglich Komponenten in das Gesamtsystem einzufügen. Darüber hinaus erhält man die Möglichkeit der Konfigurierbarkeit, es ist möglich den Agenten nur mit den Komponenten auszustatten die für das Einsatzszenario notwendig sind.

Zusätzlich wurden beim Softwarekonzept die Prinzipien von Webanwendungen, ins besondere Representational State Transfer (REST), berücksichtigt. Es wurde auf die Trennung von Client & Server geachtet, die Zustandslosigkeit der Kommunikation, die Cachebarkeit von Informationen, das Bereitstellen von einheitlichen Schnittstellen, die Möglichkeit der Abbildung in hierarchischen Schichten, und die Funktionalität von Code-On-Demand. Darüber hinaus werden Websockets eingesetzt um eine asynchrone und bidirektionale Verbindung zwischen Client und Server aufbauen zu können. Durch Websockets erhält man eine aufrechterhaltene Verbindung wodurch eine verzögerungsfreiere Kommunikation möglich ist.

Ein Agent kann aus folgenden Komponenten zusammengesetzt werden. Die Data Measurement Komponente ist verantwortlich während des Produktionsprozesses Sensordaten zu sammeln, dafür bietet sie eine Schnittelle um von externen Sensoren Daten zu erhalten. Die Production Planning Komponente hat die Aufgabe der Visualisierung des Produktionsprozesses, es werden die Aufträge dargestellt die einem Arbeitsplatz zugeordnet sind. Zusätzlich werden Rückmeldungen und Statusänderungen an das PPS-System gemeldet. Die Adapter Production Planning Komponente ist verantwortlich für die Kommunikation mit dem PPS-System, sie ist das Bindeglied zwischen der Production Planning Komponente und dem eingesetzten PPS-System, daher muss sie abhängig von dem eingesetzten PPS-System angepasst werden. Die Feedback Komponente bietet dem Benutzer die Möglichkeit Feedback zu geben und über Ereignisse benachrichtigt zu werden. Die Rule Engine Komponente deckt das regelbasierte Expertensystem ab. Sie prüft ob für ausgelöste Events Regeln vorhanden sind und löst ggf. die zugeordneten Anweisungen aus.

Die Integration der Softwarelösung in einen bestehenden Arbeitsplatz kann Abbildung 4.9 entnommen werden. Es ist vorgesehen jeden Arbeitsplatz mit einem eigenen Agenten auszustatten, dieser Agent verfügt darüber hinaus noch über, am Arbeitsplatz angebrachte externe Sensoren zum Erfassen von Sensordaten, und einem HMI zur Interaktion mit dem Mitarbeiter. Bei der Gestaltung der Benutzeroberflächen wurde darauf geachtet diese übersichtlich und leicht verständlich zu gestalten.

Das Thema Sicherheit ist von hoher Relevanz. Um die Kommunikation zwischen den Komponenten und diversen Drittsystem abzusichern wird HTTPS eingesetzt, dadurch werden jegliche übertragenen Daten verschlüsselt und sind nicht für Dritte einsehbar. Zusätzlich ist ein rollenbasiertes Benutzerkonzept notwendig. Jeder Funktionalität innerhalb des System wird mit der Angabe über die benötigte Rolle versehen. Anschließend werden Mitarbeitern diese Rollen zugeordnet, dadurch werden Funktionen nur für die berechtigten Mitarbeiter nutzbar.

Kapitel 5

Proof of Concept

Innerhalb dieses Kapitels wird die Implementierung des in Kapitel 4 erstellte Softwarekonzept vorgestellt. Dabei wird zusätzlich auf die eingesetzten Technologien eingegangen, außerdem wird die Erweiterbarkeit des Softwarekonzeptes diskutiert, und die Performance der prototypischen Umsetzung analysiert. Abschließend wird die Einhaltung der Anforderungen validiert und das Vorgehen kritisch reflektiert.

5.1 Eingesetzt Technologien

Innerhalb dieses Kapitels werden die verwendeten Technologien beschrieben.

5.1.1 Spring Boot

Bei dem Spring Framework handelt es sich um ein Framework für die Java Plattform. Ziel des Frameworks ist es, die Infrastruktur und Abhängigkeiten innerhalb eines Projekts zu verwalten, damit die Umsetzung der Business Logik im Vordergrund stehen kann [Spr17]. Spring Boot ist ein Unterprojekt des Spring Frameworks und dient dem vereinfachten Anlegen von neuen Projekten, basierend auf dem Spring Framework. Spring Boot bietet durch den Spring Container und der damit verbundenen Inversion of Control / Depencendy Injection [Fow04] viele Vorteile. Objekte können ihre Abhängigkeiten zu anderen Objekten definieren und der Container stellt ihnen diese bereit. Darüber hinaus lassen sich Spring Boot Anwendungen, die als jar-Datei gepackt wurden, unter Linux als Service registrieren.

5.1.2 MongoDB

MongoDB ist eine OpenSource NoSQL-Datenbank. Zum einen arbeitet MongoDb nicht Tabellen sondern Dokumenten orientiert und verwenden nicht die Structured Query Language (SQL) zur Abfrage ihrer Daten. Dies resultiert in einer vereinfachten Speicherung des gewünschten Datenmodels und somit einem flexibleren Umgang mit den Daten sowie einer erhöhten Skalierbarkeit [Mon17]. Bei dem Einsatz von SQL basierten Datenbanken

in der objektorientierten Programmierung war es in der Regel umständlich das objektbasierte Datenmodell in Tabellen abspeichern zu können, es war notwendig die Objekte in festgelegte Schemata abzulegen. Der dokumentenorientierte Ansatz von MongoDB erlaubt es Objekte schemalos in der Datenbank abzulegen.

5.1.3 Open API

Die saubere Definition von guten Schnittstellen ist ein entscheidender Faktor für erfolgreiche Softwarelösungen. Besonders die Dokumentation der Schnittstelle und die Implementierung müssen unbedingt konsistent gehalten werden um Fehlverhalten verhindern zu können. Zusätzlich ist die Wahl der Beschreibungsart entscheidend. Auf der einen Seite möchte man sich nicht auf eine hersteller- oder produktspezifische Sprache festlegen, auf der anderen Seite möchte man bei der Erstellung der Schnittstelle durch Tool Unterstützung, Validierung, und Generierung des Quelltextes unterstützt werden.

Um die beschriebene Problematik zu lösen wurde die Open API Initiative [ope17] gegründet. Unter dem Dach der Linux Foundation haben sich diverse Hersteller vereinigt zur Erstellung, Pflege, und Förderung eines herstellerunabhängigen Beschreibungsformat. Das ursprüngliche Beschreibungsformat ist aus dem Swagger Projekt [swa17] entstanden und ist in der Open API aufgegangen. Das Swagger Projekt liefert die notwendigen Tools zum Erstellen, Validieren, und der Generierung des Quelltextes. Der große Benefit in der Beschreibung der Schnittstellen mit Open API liegt in der Möglichkeit sowohl den Quelltext als auch die Dokumentation aus der Schnittstellenbeschreibung generieren zu können, dadurch ist die Problematik der Konsistenz zwischen Implementierung und Dokumentation gelöst. Ein weiterer Vorteil beim Einsatz von Open API mit den Tools von Swagger ist die Möglichkeit der Quelltextgenerierung in ganz unterschiedlichen Sprachen und Technologien. Ein Auszug der aktuell unterstützten Sprachen und Technologien:
Clients: C#(.net), C++(QT5, Tizen), Java(Jersey, Feign), JavaScript(Angular), PHP, Python, Ruby, Swift, Typescript(Angular)
Server: C#(.net), C++(Restbed), Java(Spring, Play Framework), JavaScript(Node.js), PHP(Lumen), Python(Flask)

5.2 Projektstruktur

Wie bereits in Kapitel 4 beschrieben wurde, wird besonderen Wert auf die Modularität der Anwendung gelegt. Dies resultiert in einer starken Entkoppelung der Komponenten. Dadurch erhält man die Möglichkeit die Anwendung um Komponenten zu erweitern die aktuell noch nicht zur Verfügung stehen. Um dem Softwarekonzept in Kapitel 4 genüge zu tun wurden die einzelnen Komponenten als eigenständige Projekte umgesetzt. Dadurch kann jede Komponente eigenständig arbeiten woraus eine ideale Aufteilung der Zuständigkeiten (Separation of Concerns) resultiert, darüber hinaus ist es jedoch auch Möglich verschiedene Agenten zu konfigurieren und nur die Komponenten einzusetzen, welche benötigt werden. Im Folgenden werden die einzelnen Projekte vorgestellt.

- Das Projekt „Agent“ dient der Zusammenfassung aller benötigten Komponenten um den gewünschten Agenten zusammenzusetzen. Dies bietet die Möglichkeit abhängig von dem Zielszenario nur die benötigten Komponenten auszuwählen und einen Agenten zu konfigurieren der ideal auf die gewünschte Situation passt.

- Das Projekt „Common“ dient der Definition und Bereitstellung von allgemeingültigen Objektdefinitionen für alle Projekte die sie benötigten. Dadurch werden mehrdeutige Objektdefinitionen und das Problem der Inkonsistenz vermieden.

- Das Projekt „Data Measurement“ entspricht der Definition innerhalb des Softwarekonzeptes und ist für das entgegennehmen und verarbeiten von externen Sensordaten zuständig.

- Das Projekt „Production Planning“ dient der abstrakten Kommunikation mit dem PPS-System. Zusätzlich dient es als lokales PPS-System für die Verwaltung der zugeordneten Jobs. Für die konkrete Kommunikation dient das Projekt „Adapter Production Planning“, welches abhängig von dem tatsächlich eingesetzten PPS-System implementiert ist.

- Das Projekt „Adapter Production Planning“ dient der konkreten Kommunikation mit dem PPS-System und ist abhängig von dem eingesetzten PPS-System implementiert.

- Das Projekt „Feedback“ entspricht der Definition innerhalb des Softwarekonzeptes und ist für das Bereitstellen einer Schnittstelle zur Rückmeldung des Produktionsprozesses zuständig.

- Das Projekt „Rule Engine“ entspricht der Definition innerhalb des Softwarekonzeptes und ist für das entgegennehmen von Events, der Verarbeitung von Regeln, und dem ausführen von Actions zuständig.

5.3 Implementierungsdetails

Innerhalb dieses Kapitels sollen Besonderheiten in der Implementierung einzelner Komponenten und deren Konzepte herausgestellt werden.

5.3.1 Rule Engine

In dem Softwareentwurf wurde festgelegt, dass zur Definition von Regel Expressions die MVEL verwendet werden soll. Bei der Entwicklung von Software ist es üblich nicht alle Teile der Software selbstständig zu entwickeln. In der Regel verwendet man Software Bibliotheken die eine gewisse Grundfunktionalität bereitstellen um sich auf den eigentlich fachlichen Kern der zu entwickelnden Software konzentrieren zu können. Zur Evaluierung der Expressions wurde sich einer bereits erprobten Rule Engine bedient, welche die MVEL

verwendet zur Definition von Regeln [Kut17]. Diese Rule Engine ermöglicht es eine beliebige Anzahl von Regeln in Abhängigkeit von einem Eingabeobjekt prüfen zu lassen. Als Ergebnis der Evaluation erhält man die positiv evaluierten Regeln.

Wie bereits im Softwareentwurf beschrieben, siehe Kapitel 4.2.5, stützt sich die Rule Engine auf EventTypes und ActionTypes. Nach dem ein Event ausgelöst wird überprüft die Rule Engine anhand des EventTypes, innerhalb des Events, ob Regeln für diesen EventType vorhanden sind. Sollte dies der Fall sein wird anhand der übergebenen Daten, innerhalb des Events, überprüft ob die Expression innerhalb der Regel zutrifft. Wird die Expression positiv evaluiert wird eine neue Action ausgelöst von dem ActionType der in der Regel hinterlegt ist. Beim ActionType handelt es sich um die Angabe des ActionType Names und dem ActionTarget, welcher den Namen der Zielkomponente enthält. Um nun die konkrete Action-Implementierung zu erhalten wird sich der Funktionalität der Dependency Injection, welche vom Spring Framework bereitgestellt wird, bedient. Bei Dependency Injection handelt es sich um ein Entwurfsmuster, welches ein zusätzliches Objekt, den „assembler“, dafür nutzt Felder innerhalb einer Klassen mit konkreten Implementierungen des angegeben Interfaces zu versehen [Fow04]. Dabei prüft der „assembler“, welche Implementierungen des angegeben Interfaces im Klassenpfad zur Verfügung stehen und fügt alle gefunden Implementierungen dem Feld hinzu. Der Softwareentwickler muss sich nicht um das Auffinden der Implementierungen und das Instanziieren der Objekte kümmern.
Ein konkretes Beispiel von Dependency Injection kann dem Folgenden entnommen werden:

```
    @Autowired
private List<IAction> actions;
```

Über die Annotation „@Autowired“ gibt man an, dass man die Funktionalität der Dependency Injection für dieses Feld verwenden möchte. Durch „List<IAction> actions“ gibt man an, dass man alle Implementierungen des Interfaces „IAction“ in dem Feld mit dem Namen „actions“ erhalten möchte. Nach dem Start der Anwendung erfolgt die Prüfung der Klassen für das Schlüsselwort der Dependency Injection „@Autowired“. Der Klassenpfad wird auf jegliche Implementierungen des angegeben Interfaces geprüft und die gefundenen Implementierungen werden in dem Feld abgelegt. Durch die Verwendung von Dependency Injection ist es möglich eine lose Koppelung der Komponenten zu realisieren. Wie bereits im Kapitel 4 Softwarekonzept erwähnt wurde, wurde bei dem Entwurf der Software sehr stark auf die lose Koppelung und die Konfiguration von den benötigten Komponenten geachtet. Die Verwendung der Dependency Injection innerhalb der Rule Engine ermöglicht diese lose Koppelung. Sollte durch das Hinzufügen von neuen Komponenten Implementierungen von „IAction“ hinzugefügt werden, findet die Dependency Injection diese und dadurch sind sie in der Rule Engine verwendbar ohne den Quelltext der Rule Engine verändern zu müssen.

5.3.2 Production Planning

Im Folgenden wird näher auf die Implementierungsdetails der Komponente Production Planning eingegangen. Die Komponente stellt folgende Funktionen bereit:

- **Webanwendung**: Es wird eine Webanwendung zur Verfügung gestellt, mit der die Benutzer interagieren können. Für jede Rolle gibt es verschiedene Ansichten. Der Meister und der Techniker können beispielsweise alle Ressourcen und deren Aufträge einsehen, der Mitarbeiter hingegen sieht nur die Ressource und die Aufträge, welche dieser Ressource zugewiesen sind. Die Webanwendung wurde mit Hyper Text Markup Language (HTML) und Javascript (JS) umgesetzt. Mithilfe von JS wurden Services bereitgestellt, die es ermöglichen Daten an den entsprechenden REST-Controllern der Komponente abzufragen (vgl. Abbildung 5.1). Diese Services werden in den HTML-Seiten genutzt.

- **Events**: Neben der Webanwendung werden durch die Komponente auch folgende Events gefeuert.

  – '*jobAssigned*'
  – '*jobChanged*'
  – '*jobDeleted*'

  Die Komponente synchronisiert und speichert alle Aufträge, die ihr zugewiesen sind in regelmäßigen Abständen. Dafür wird das Interface `IAdapterProductionPlanning` (vgl. Abbidlung 4.4) genutzt. Der Ablauf der Synchronisation wird in Abbildung 5.2 dargestellt. Während der Synchronisation wird festgestellt, ob ein Auftrag geändert, entfernt, oder neu zugewiesen wurde. Anschließend wird das entsprechende Event ausgelöst.

- **Actions**: Um die in dem Softwarekonzept definierten Actions *writeData* und *deliverFeedback* umzusetzen wurde das Interface `IAdapterProductionPlanning` (vgl. Abbidlung 4.4) genutzt. Dies ermöglicht es, Sensordaten und Statusmeldungen von Jobs, Ressourcen und Operations an das PPS-System zu senden.

5.3.2.1 Production Planning Mock

Für die prototypische Umsetzung der Software wurde ein beispielhaftes PPS-System implementiert. Dieses generiert CustomerOrders und alle anderen damit verbundenen Objekte. Außerdem weißt das beispielhafte PPS-System einzelnen Ressourcen Jobs zu. Die Schnittstellen dieses Systems sind mit der bereits erwähnten Open-Api dokumentiert. Ein Ausschnitt dieser Dokumentation ist in Abbildung 5.3 zu sehen, dieser beschreibt in YAML die Schnittstelle um alle Aufträge einer Ressource abzufragen. Mithilfe dieser Dokumentation werden Klassen und Interfaces generiert, sodass nur noch die Geschäftslogik selbständig hinzugefügt werden muss. Durch die Generierung und die Tatsache, dass die Generierung

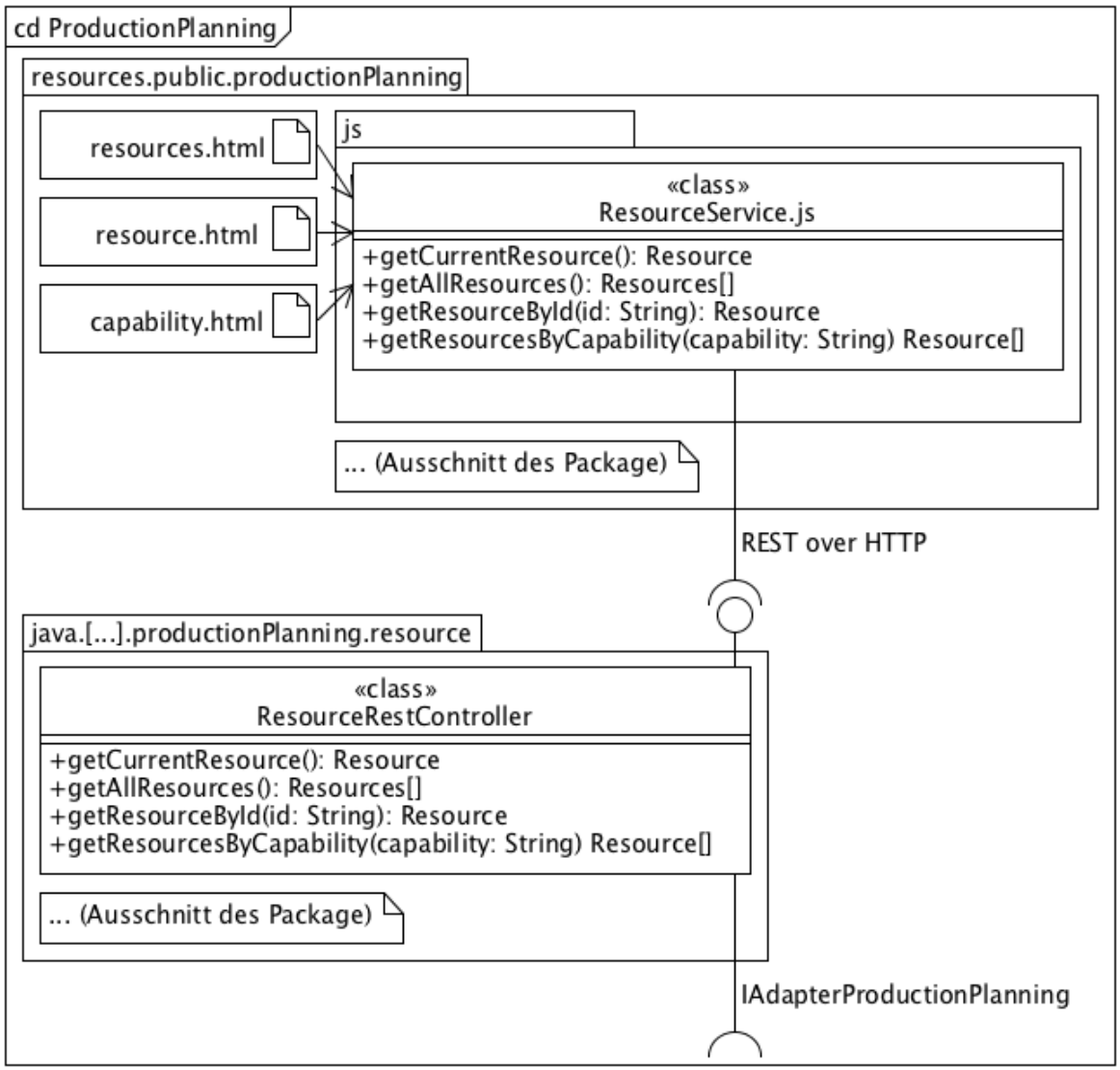


Abbildung 5.1: Verbindung zwischen Javascript Services und REST-Controller

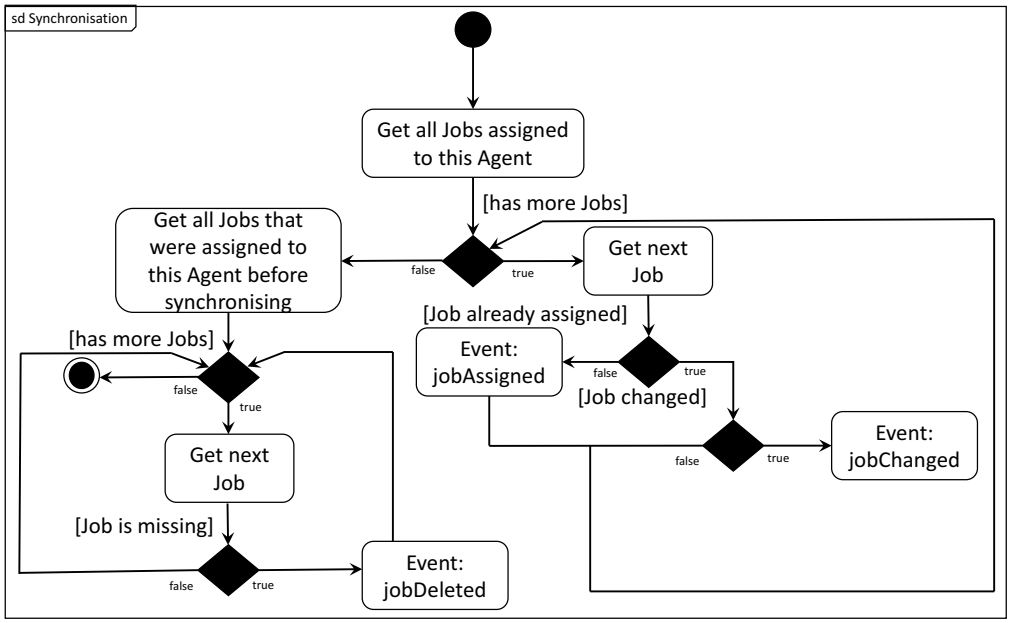


Abbildung 5.2: Aktivitätsdiagramm zur Synchronisierung von Aufträgen

Teil des Buildprozesses ist, entspricht das System immer der Dokumentation. Zur besseren Lesbarkeit der Dokumentation kann diese in HTML generiert werden (vgl. Abbildung 5.4).

5.3.2.2 Adapter Production Planning Mock

Um das beispielhafte PPS-System anzusprechen wurde die Komponente AdapterProductionPlanningMock implementiert. Diese nutzt die durch die Open-Api dokumentierten Schnittstellen und generiert daraus Services, die die Kommunikation zu diesen übernehmen. Diese Services werden in der Implementierung des Interfaces *IAdapterProductionPlanning* genutzt. In Abbildung 5.5 werden die einzelnen Systeme mit einigen Komponenten und ihre Verbindungen dargestellt. Um den Kommunikationsfluss zwischen den Systemen durch die Komponenten zu verdeutlichen wurde zusätzlich ein Sequenzdiagramm erstellt (vgl. Abbildung 5.6). Aus diesem geht hervor, dass wenn der *ResourceService* eine Anfrage an den *ResourceRestController* in der Komponente productionPlanning stellt, ruft diese über das Interface *IAdapterProductionPlanning* bzw. die Implementierung *AdapterProductionPlanningImpl* den *ResourceApiClient* auf. Die Klasse *ResourceApiClient* stellt wiederum eine HTTP-Anfrage an den *ResourceApiRestController* im ProductionPlanningMock, auf die mit der entsprechenden Ressource reagiert wird. Die Ressource wird jetzt entsprechend der Reihenfolge zurückgegeben, bis diese beim *ResourceService* ankommt. Der *ResourceService* läuft im Browser des Client und stellt Anfragen an den Agent, dieser fragt mithilfe des AdapterProductionPlanningMock die entsprechenden Ressource am ProductionPlanningMock an.

```
...
  '/resources/{resourceId}/jobs':
    parameters:
      - $ref: '#/parameters/resourceId'
    get:
      operationId: getAllJobsByResourceId
      tags:
        - job
        - resource
      summary: get all job assigned to this resource
      parameters:
        - $ref: '#/parameters/pageSize'
        - $ref: '#/parameters/pageNumber'
      responses:
        '200':
          description: ''
          schema:
            type: array
            items:
              $ref: '#/definitions/Job'
...
```

Abbildung 5.3: Ausschnitt der Schnittstellendokumentation von dem beispielhaften PPS-System

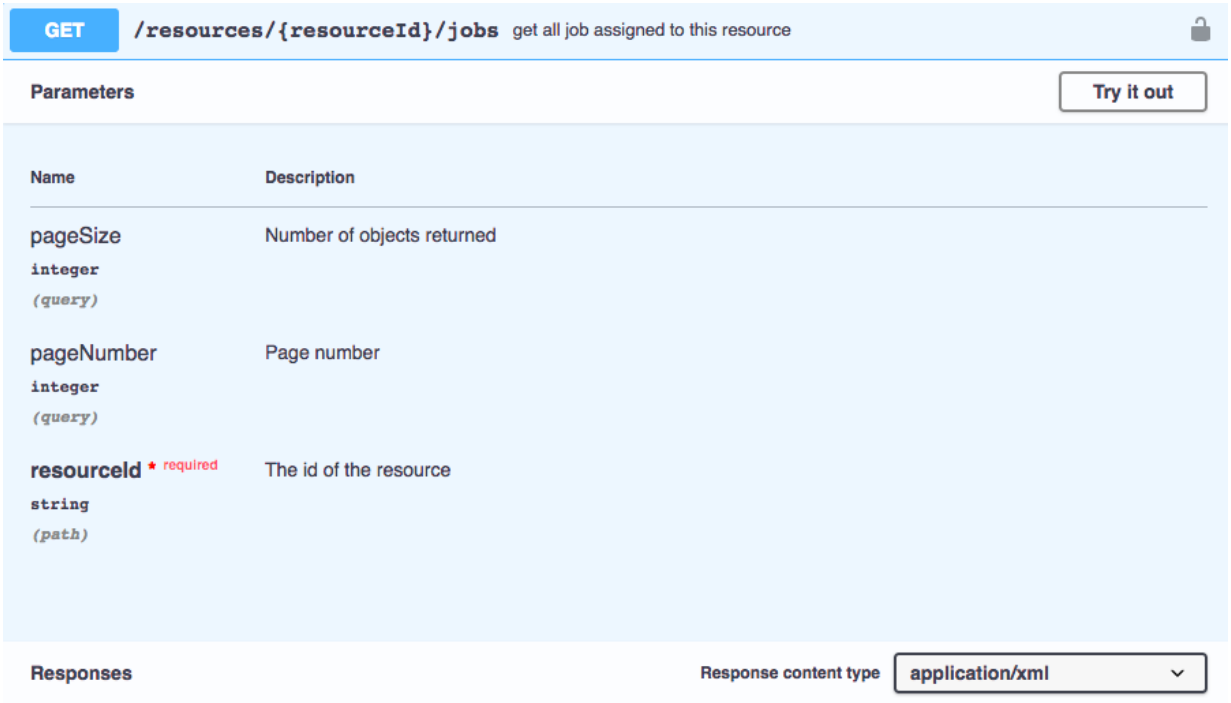


Abbildung 5.4: Generierte HTML-Dokumentation aus der Open-Api-Schnittstelle.

5.3.3 Feedback

Die Feedback Komponente ermöglicht es unter anderem Ereignisse an andere Agenten weiterzuleiten. Dazu wird ein MessageBroker genutzt, als MessageBroker für die prototypische Implementierung wurde Apache ActiveMQ [1] genutzt. Spring Boot unterstützt Java Message Service (JMS) und ermöglicht über ein entsprechendes Projekt eine einfache Konfiguration von JMS über ActiveMQ. Durch den Einsatz von JMS können alle Agenten einer Rolle über ein Ereignis informiert werden. Dabei wird das Ereignis im JSON-Format in den Body einer Nachricht eingebettet und an die entsprechende Topic geschickt.

Soll jedoch nur ein Agent über ein Ereignis informiert werden findet eine Auktion statt. Der Ablauf der Auktion ist in dem Sequenzdiagramm in Abbildung 5.7 dargestellt. Ein Agent, der ein Ereignis an einen Agenten weiterleiten möchte ist der Auktionator, er ruft dazu auf Angebote abzugeben. Dies tut er indem er eine Nachricht auf die entsprechende Topic legt und in der Nachricht eine Antwort Queue hinzufügt. Die anderen Agenten sind die Bieter. Sie nutzen diese Queue um ein Angebot abzugeben. Bei der Abgabe des Angebots geben die Bieter ebenefalls eine Queue an, über die sie benachrichtigt werden können. Die Höhe des Angebotes richtet sich nach der Arbeitsbelastung des Agenten. Zur Berechnung dieser wird ein Interface bereitgestellt, dass alle Komponenten implementieren können. In der prototypischen Anwendung wird die Arbeitsbelastung durch die Dauer der zugewiesenen Aufträge bestimmt. Der Auktionator sammelt eine gewisse Zeit die Gebote,

[1] http://activemq.apache.org/

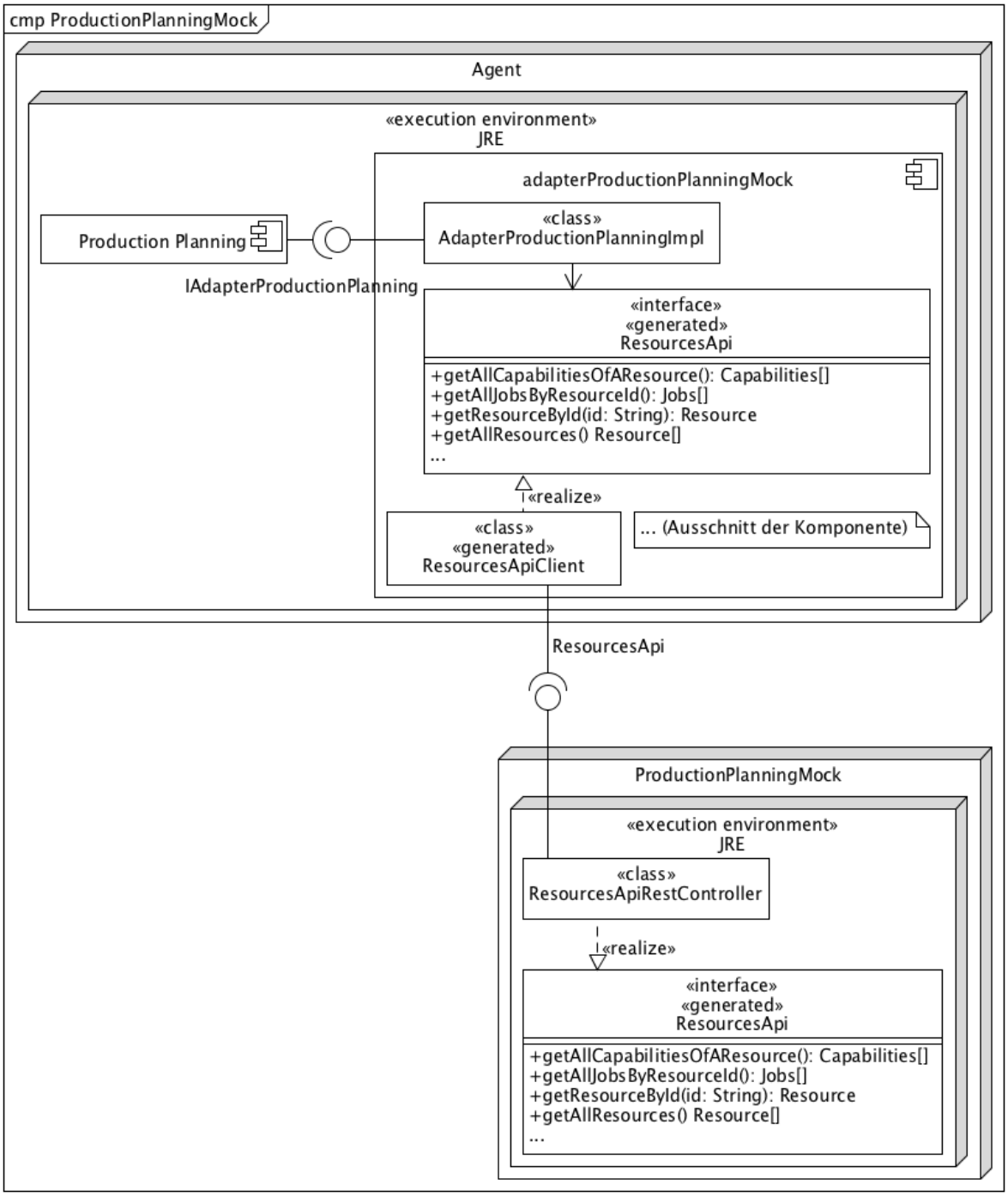


Abbildung 5.5: Production Planning Mock Komponentendiagramm

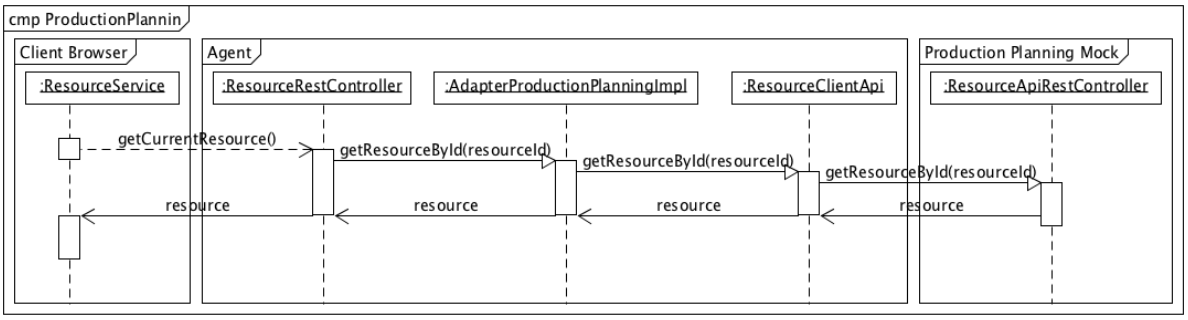


Abbildung 5.6: Sequenzdiagramm, dass den Ablauf vom Anfragen einer Ressource über alle Komponenten beschreibt.

um schließlich den Agenten mit der niedrigsten Arbeitsbelastung über den Zuschlag zu informieren. Allen anderen Bietern erteilt er eine Absage.

5.4 Erweiterbarkeit

Zur Überprüfung ob das Softwarekonzept die gewünschte Flexibilität der Erweiterbarkeit ermöglicht wird exemplarisch das Einbetten einer neuen Komponente veranschaulicht.

Als Beispiel für die Erweiterbarkeit wird eine Komponente zum Ansteuern von Aktorik (Actuator) umgesetzt. Das Beispielszenario sieht vor, dass die Arbeitsplätze zusätzlich mit einer Ampel ausgestattet werden um den aktuellen Zustand des Arbeitsplatzes optisch zu signalisieren. Wenn die Ampel grün aufleuchtet sind keine besonderen Vorkommnisse am Arbeitsplatz vorhanden, es wird der normal Zustand signalisiert. Sollte die Ampel gelb aufleuchten wird eine Beeinträchtigung des Arbeitsablaufes signalisiert, es ist notwendig, dass der Meister sich den Arbeitsplatz anschaut und etwaige Beeinträchtigungen auflöst. Leuchtet die Ampel rot auf besteht eine starke Beeinträchtigung des Arbeitsablaufes, es sind umgehend Gegenmaßnahmen durch den Meister einzuleiten. Abbildung 5.8 kann die Integration der Actuator Komponente in den Agenten und dem Arbeitsplatz entnommen werden.

Zunächst muss eine Schnittstelle zwischen der Actuator Komponente und dem eigentlichen Aktor definiert werden. Dabei wird, wie bei den externen Sensoren, von der Möglichkeit einer TCP/IP basierten Kommunikation ausgegangen. Es werden die Aufrufe definiert die das Schalten der Ampel in die verschiedenen Stati festhält. Um die neue Actuator Komponente innerhalb des Agenten nutzbar zu machen werden neue ActionTypes definiert. Insgesamt werden drei ActionTypes definiert:

- „switchTrafficLightGreen“ - schaltet die Ampel auf grün.
- „switchTrafficLightYellow“ - schaltet die Ampel auf gelb.
- „switchTrafficLightRed“ - schaltet die Ampel auf rot.

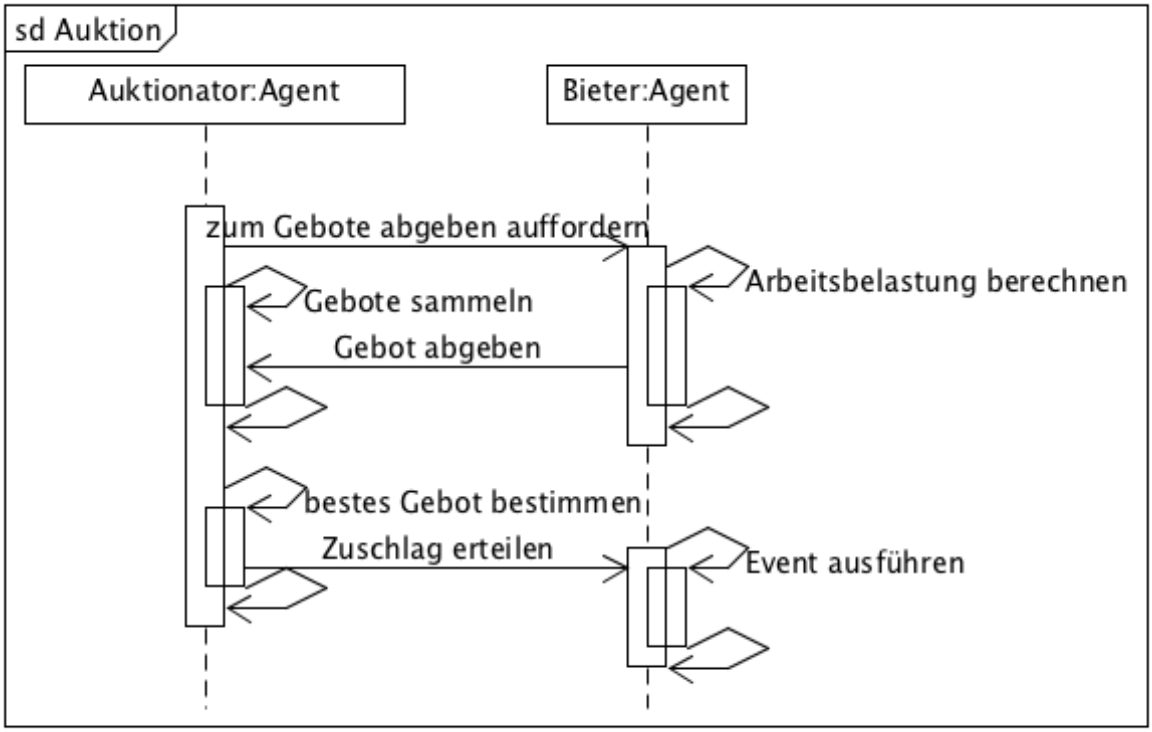


Abbildung 5.7: Sequenzdiagramm, dass den Ablauf einer Auktion aufzeigt

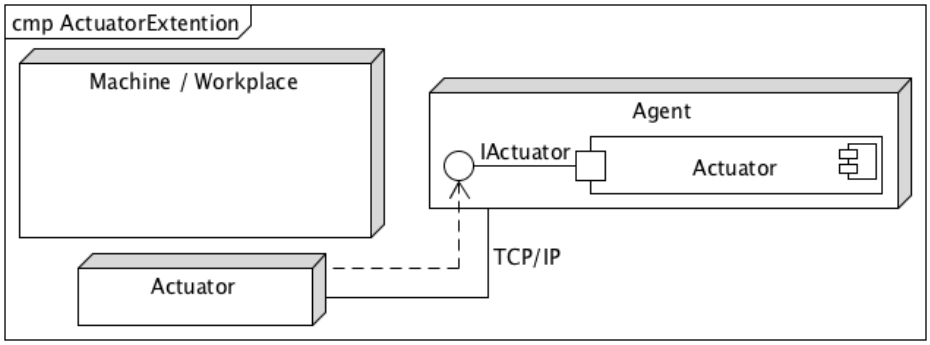


Abbildung 5.8: Erweiterung Komponente Actuator

Innerhalb der Actuator Komponente werden konkrete Implementierungen der beschrieben Actions angeboten, welche die Schnittstelle zum Actuator aufrufen und das tatsächliche Schalten der Ampel initiieren. Der letzte Schritt besteht darin innerhalb des „Agent“ Projektes die Abhängigkeit zu der neu definierten Actuator Komponente hinzuzufügen. Dadurch wird die Actuator Komponente in den Klassenpfad aufgenommen und die Implementierungen der neu definierten Actions können von der Rule Engine gefunden und ausgeführt werden.

Um das Schalten der Ampel produktiv einsetzen zu können ist es notwendig Regeln zu definieren die das Schalten der Ampel veranlassen. Eine mögliche Umsetzung von Regeln soll an einem Beispiel erläutert werden.

Ein Arbeitsplatz ist erst nutzbar nachdem ihm ein Job zugeordnet wurde. Daher wird eine Regel benötigt die nach dem Zuordnen eines Jobs die Ampel auf grün stellt. Eine neue Regel wird angelegt, welche auf den EventType „jobAssigned“ reagiert und eine Action vom ActionType „switchTrafficLightGreen“ auslöst. Eine besondere Regel Expression ist an dieser Stelle nicht notwendig da aktuell auf keine besonderen Eigenschaften des neuen Jobs geprüft werden soll.
Eine Beeinträchtigung des Arbeitsablaufes könnte drohen durch die Übersteigung einer gewissen Temperatur. Daher wird eine neue Regel definiert, welche auf den EventType „dataMeasured“ reagiert und eine Action vom ActionType „switchTrafficLightYellow“ auslöst. Mit folgender Regel Expression soll geprüft werden ob die Temperatur über 35°C gestiegen ist.

```
input.data.sensorName == "TemperaturSensor1" && input.data.value > 35;
```

Eine starke Beeinträchtigung der Arbeitsabläufe kann durch fehlen materieller Betriebsmittel ausgelöst werden. Am Beispiel eines manuellen Produktionsprozesses mit Handarbeitsplatz könnte eine leere Gasflasche das Schweißen verhindern und somit in einer starken Beeinträchtigung des Arbeitsablaufes resultieren. Um diesen Fall abzudecken kann eine Regel definiert werden die auf den EventType „dataMeasured“ reagiert und eine Action vom ActionType „switchTrafficLightRed“ auslöst. Die folgende Expression prüft ob der Sensor an der Gasflasche den Zustand einer leeren Gasflasche ausgibt.

```
input.data.sensorName == "GasCanister" && input.data.value == “empty“;
```

Dieses Beispiel soll illustrieren wie eine neue Komponente hinzugefügt werden kann und sinnvoll in die bestehende Softwarelösung eingebettet wird. Dabei ist dieses Beispiel exemplarisch zu verstehen und soll den möglichen Ablauf darstellen.

Der beschriebene Ablauf stellt dar, dass das Softwarekonzept über die gewünschte und benötigte Erweiterbarkeit verfügt um mit neuen Komponenten ausgestattet zu werden.

5.5 Performance

Die zentrale Komponente der implementierten Architektur ist die RuleEngine. Alle Ereignisse laufen über diese und resultieren durch Regeln in der Ausführung von bestimmter Geschäftslogik, daher ist die benötigte Zeit zur Auswertung der Regeln entscheidend für die Performance des Gesamtsystems. Die Auswertung von Regeln in der RuleEngine darf auch bei vielen auszuwertenden Regeln nicht überproportional ansteigen, um die Performance des Gesamtsystems nicht negativ zu beeinflussen. Im Folgenden werden daher verschiedene Messungen durchgeführt, die überprüfen sollen, wie sich die RuleEngine bei einer Vielzahl von Regeln verhält.

Die Dauer der Auswertung von Regeln ist von mehreren Faktoren abhängig:

- **Der Bedingung**: Die Bedingung, die mithilfe von MVEL definiert wurde, muss zunächst geparst und ausgewertet werden. Die Komplexität der Bedingung, also die Anzahl der Vergleiche kann sich somit auf die Ausführungszeit auswirken.
- **Der Anzahl von Regeln**: Die Anzahl der Regeln ist ein wichtiger Faktor, je mehr Regeln es gibt, desto mehr Bedingungen müssen überprüft werden und desto länger dauert das Abfragen der Regeln aus der Datenbank.
- **Der Aktion**: Die Ausführung einer Aktion beansprucht Rechenzeit, je nach Regelsatz kann ein Ereignis mehrere Aktionen ausführen, die nacheinander abgearbeitet werden. Die Ausführungszeit der Aktion wird jedoch außer Acht gelassen, da diese je nach Aktion unterschiedlich ist.

Abhängig von den zwei Faktoren 'Anzahl der Regeln' und 'Anzahl der Vergleiche', wurde ein Testplan aufgestellt (vgl. Abbildung 5.9). Dabei werden logarithmisch von eins bis 1000 Werte für die zwei Faktoren angenommen. Ein Agent mit mehr als 1000 Regeln wäre schwer zu handhaben und der Ablauf der Regeln wäre schwer nachzuvollziehen. Regeln mit mehr als 1000 Vergleichen sollten ebenfalls nicht auftreten, da die Nachvollziehbarkeit dieser nicht mehr gegeben ist. Aus den angenommen Werten ergeben sich 16 Testszenarien. Für jedes Szenario werden drei Messungen durchgeführt und ein Mittelwert gebildet. Dies soll Messfehler zu reduzieren, die dadurch auftreten könnten, dass andere Programme zeitweise mehr Rechenleistung benötigen. Die Testszenarien werden auf einem Computer mit einem 3,3 GHz Intel Core i7 und 16 GB 2133 MHz DDR3 RAM durchgeführt.

Anzahl Regeln \ Anzahl Vergleiche	1	10	100	1000
1	T1	T2	T3	T4
10	T5	T6	T7	T8
100	T9	T10	T11	T12
1000	T13	T14	T15	T16

Abbildung 5.9: Testplan für die Messungen

Auswertung Die Tests wurden entsprechend der Planung durchgeführt. Dabei wurde festgestellt, dass bei ansteigender Anzahl der Regeln auch die Dauer linear ansteigt (vgl. 5.10). Die Dauer zur Auswertung einer Regel bleibt dabei konstant, bzw. nimmt sogar ab. 1000 Regeln mit 10 Vergleichen auszuwerten dauert durchschnittlich 660ms (vgl. Abbildung 5.12). Dadurch kommt es auch bei einer großen Regelbasis nicht zu unerwünschten Verzögerungen bei der Ausführung der Geschäftslogik. Darüber hinaus wurde die RuleEngine so implementiert, dass immer nur die Regeln aus der Datenbank geladen und ausgewertet werden, die auf ein ausgelöstes Event reagieren. Somit kann die Regelbasis über 1000 Regeln pro Event enthalten, bis es zu Verzögerungen von durchschnittlich zu 660ms kommt.

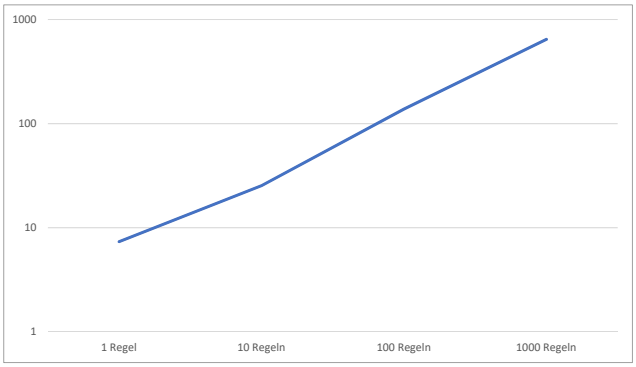


Abbildung 5.10: Der Graph zeigt die Dauer der Auswertung in Millisekunden, abhängig von der Anzahl der Regeln. Die Auswertunsgsdauer steigt mit der Anzahl der Regeln linear an. Eine Regel auszuwerten dauert bei einer Regel 7,3 ms und bei 1000 6,4 ms.

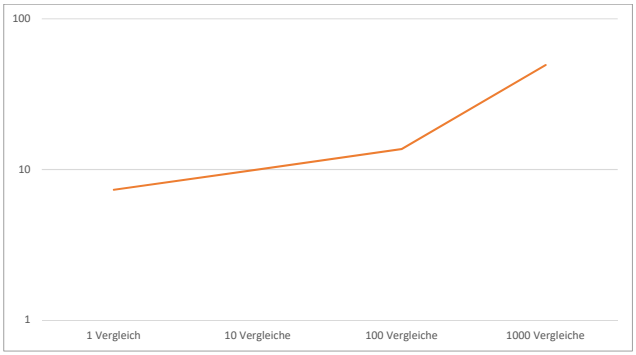


Abbildung 5.11: Der Graph zeigt die Dauer der Auswertung in Millisekunden, abhängig von der Anzahl der Vergleiche. Die Anzahl der Vergleiche steigt linear an, Vergleiche beeinflussen die Dauer der Auswertung aber nicht so lange, wie die Anzahl der Regeln.

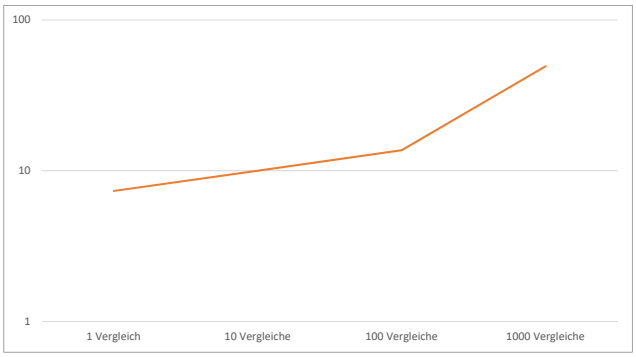


Abbildung 5.12: Der Graph zeigt die Dauer der Auswertung in Millisekunden, abhängig von der Anzahl der Regeln. Jede Linie stellt hierbei Regeln mit verschiedenen Vergleichen dar.

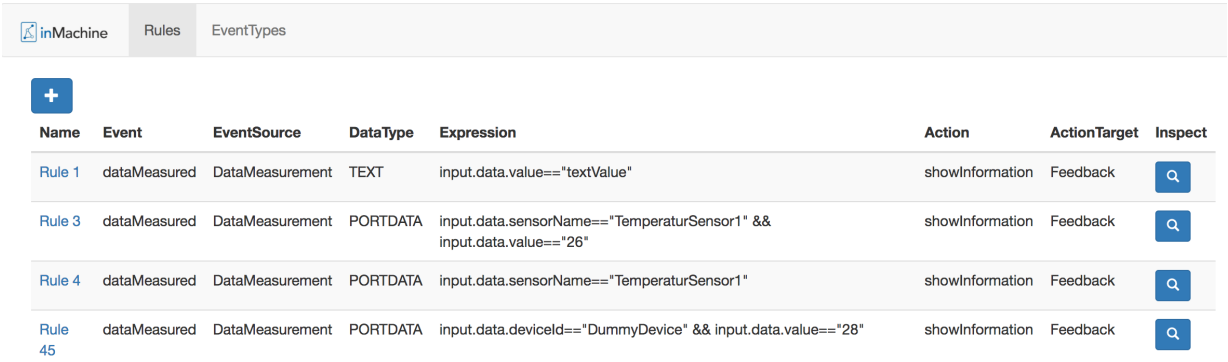
5.6 Umsetzung Benutzeroberfläche

In diesem Kapitel werden die Umsetzungen der Benutzeroberflächen beschrieben.

5.6.1 Regeln

Den Abbildungen 5.13 und 5.14 kann die Benutzeroberfläche zur Übersicht und Detailansicht der Regeln entnommen werden. Durch die Übersicht der Regeln besteht die Möglichkeit alle aktuell angelegten Regeln einzusehen, neue Regeln anzulegen, und durch die Auswahl einer Regel diese zu bearbeiten oder zu löschen. Abbildung 5.14 ist die Detailansicht einer Regel zu entnehmen. Beim Anlegen einer Regel ist es möglich diverse Attribute zu vergeben:

- Name der Regel.
- Beschreibung der Regel.
- Auswahl des EventTypen auf den die Regel reagieren soll. Abhängig von dem ausgewählten EventType und dem hinterlegten DatenTyp werden unterhalb von Expression alle möglichen Attribute aufgelistet. Dies erleichtert das Anlegen neuer Regeln da alle zur Verfügung stehenden Attribute dem Benutzer ersichtlich sind und er Werte für die relevanten Attribute hinterlegen kann.
- Expression zeigt die resultierende Regel durch das Vergeben von Werten an den möglichen Attributen.
- ActionType ermöglicht die Auswahl eines ActionTypes der ausgelöst wird, sollte die Expression positiv evaluiert werden.



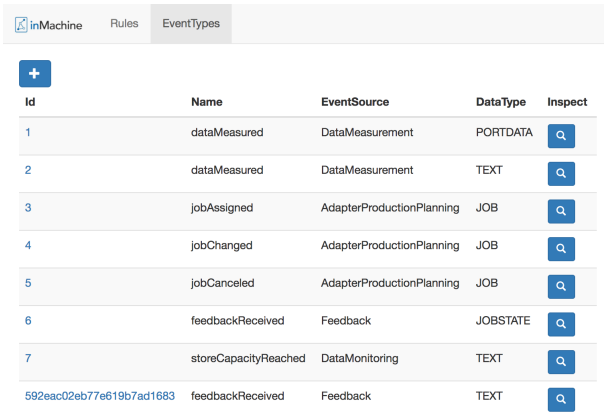
Name	Event	EventSource	DataType	Expression	Action	ActionTarget	Inspect
Rule 1	dataMeasured	DataMeasurement	TEXT	input.data.value=="textValue"	showInformation	Feedback	
Rule 3	dataMeasured	DataMeasurement	PORTDATA	input.data.sensorName=="TemperaturSensor1" && input.data.value=="26"	showInformation	Feedback	
Rule 4	dataMeasured	DataMeasurement	PORTDATA	input.data.sensorName=="TemperaturSensor1"	showInformation	Feedback	
Rule 45	dataMeasured	DataMeasurement	PORTDATA	input.data.deviceId=="DummyDevice" && input.data.value=="28"	showInformation	Feedback	

Abbildung 5.13: Benutzeroberfläche Regeln Übersicht

5.6.2 EventType

Den Abbildungen 5.15 und 5.16 kann die Benutzeroberfläche zur Übersicht und Detailansicht der EventTypen entnommen werden. Durch die Übersicht der EventTypen besteht die Möglichkeit alle aktuell angelegten EventTypen einzusehen, neue EventTypen anzulegen, und durch die Auswahl eines EventTypen diesen zu bearbeiten oder zu löschen. Durch das nachträgliche Anlegen von EventTypen besteht die Möglichkeit neue EventTypen zu definieren die z.B. von zusätzlichen Komponenten ausgelöst werden die bei der initialen Konfiguration noch nicht zur Verfügung standen. Dies bietet eine maximale Flexibilität die Softwarelösung nachträglich anpassen zu können ohne Eingriff in den Quelltext. Abbildung 5.16 ist die Detailansicht eines EventTypen zu entnehmen. Beim Anlegen eines EventTypen ist es möglich diverse Attribute zu vergeben:

- Der Name des EventTypen.
- Der EventSource, also von welcher Komponente der EventTyp ausgelöst wird.
- Der DatenTyp zur Definition welche Art von Daten hinterlegt sind.



Id	Name	EventSource	DataType	Inspect
1	dataMeasured	DataMeasurement	PORTDATA	
2	dataMeasured	DataMeasurement	TEXT	
3	jobAssigned	AdapterProductionPlanning	JOB	
4	jobChanged	AdapterProductionPlanning	JOB	
5	jobCanceled	AdapterProductionPlanning	JOB	
6	feedbackReceived	Feedback	JOBSTATE	
7	storeCapacityReached	DataMonitoring	TEXT	
592eac02eb77e619b7ad1683	feedbackReceived	Feedback	TEXT	

Abbildung 5.15: Benutzeroberfläche EventType Übersicht

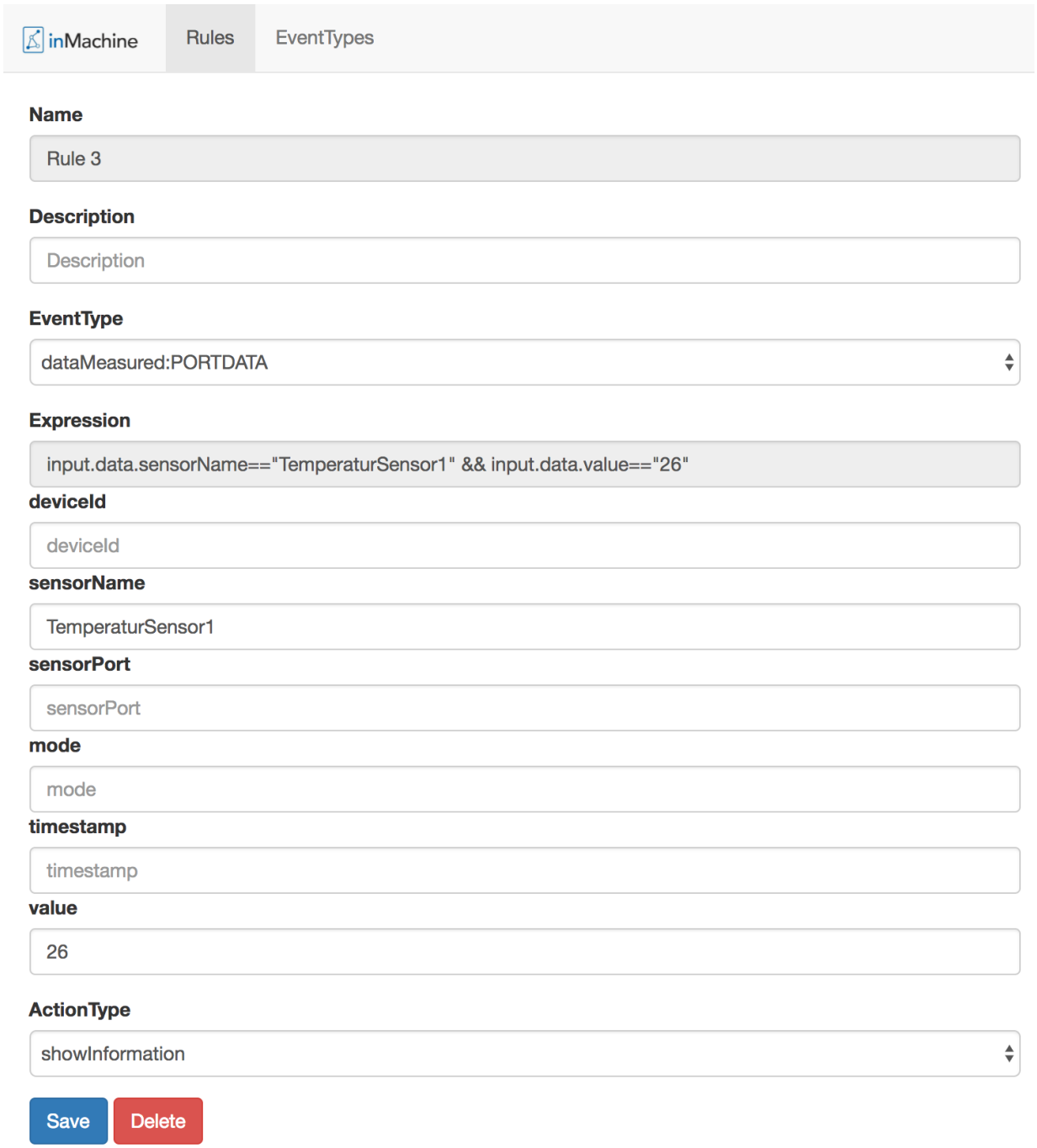


Abbildung 5.14: Benutzeroberfläche Regeln Detailansicht

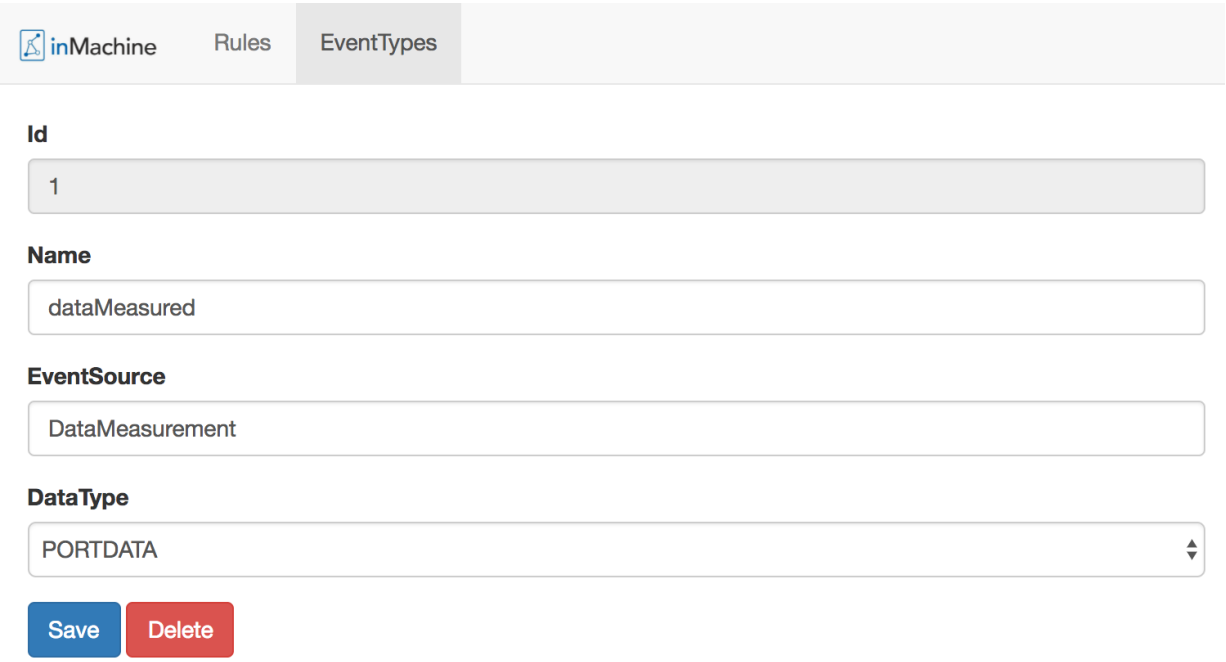


Abbildung 5.16: Benutzeroberfläche EventType Detailansicht

5.6.3 Production Planning

Die Benutzeroberfläche der ProductionPlanning Komponente ist je nach Rolle unterschiedlich. In Abbildung 5.17 ist die Benutzeroberfläche für den Meister und Techniker zu sehen. Diese können alle Ressourcen und deren Effizienz-Faktor, Stundensätze, Fähigkeiten, und Zustand (farbliche Hervorhebung) einsehen. Ferner ist es diesen Rollen möglich, genauere Informationen einer Ressource einzusehen (vgl. Abbildung 5.20). Dadurch kann nachvollzogen werden, in welchem Status sich eine Ressource befindet und welche Aufträge dieser zugewiesen sind. Der Meister und Techniker können außerdem alle Ressourcen mit einer bestimmten Fähigkeit einsehen (vgl. Abbildung 5.18) und die Ressourcen, die einer bestimmten Gruppe zugeordnet sind (vgl. Abbildung 5.19). Der Meister und der Techniker können darüber hinaus Bestellungen (vgl. Abbildung 5.21) und die einzelnen Fertigungsaufträge einsehen.

Abbildung 5.17: Übersicht des Meisters/Technikers über alle Ressourcen

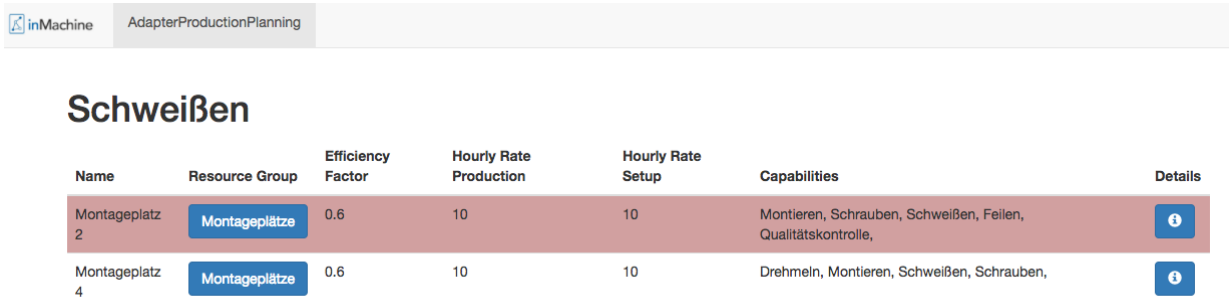


Abbildung 5.18: Übersicht des Meisters/Technikers über alle Ressourcen, die eine bestimmte Fähigkeit besitzen.

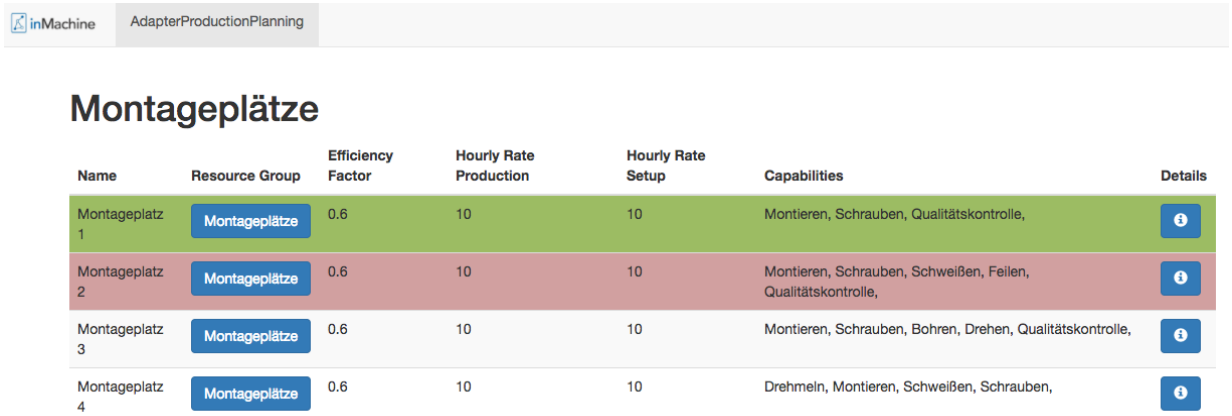


Abbildung 5.19: Übersicht des Meisters/Technikers über alle Ressourcen, die einer bestimmten Gruppe zugeordnet sind.

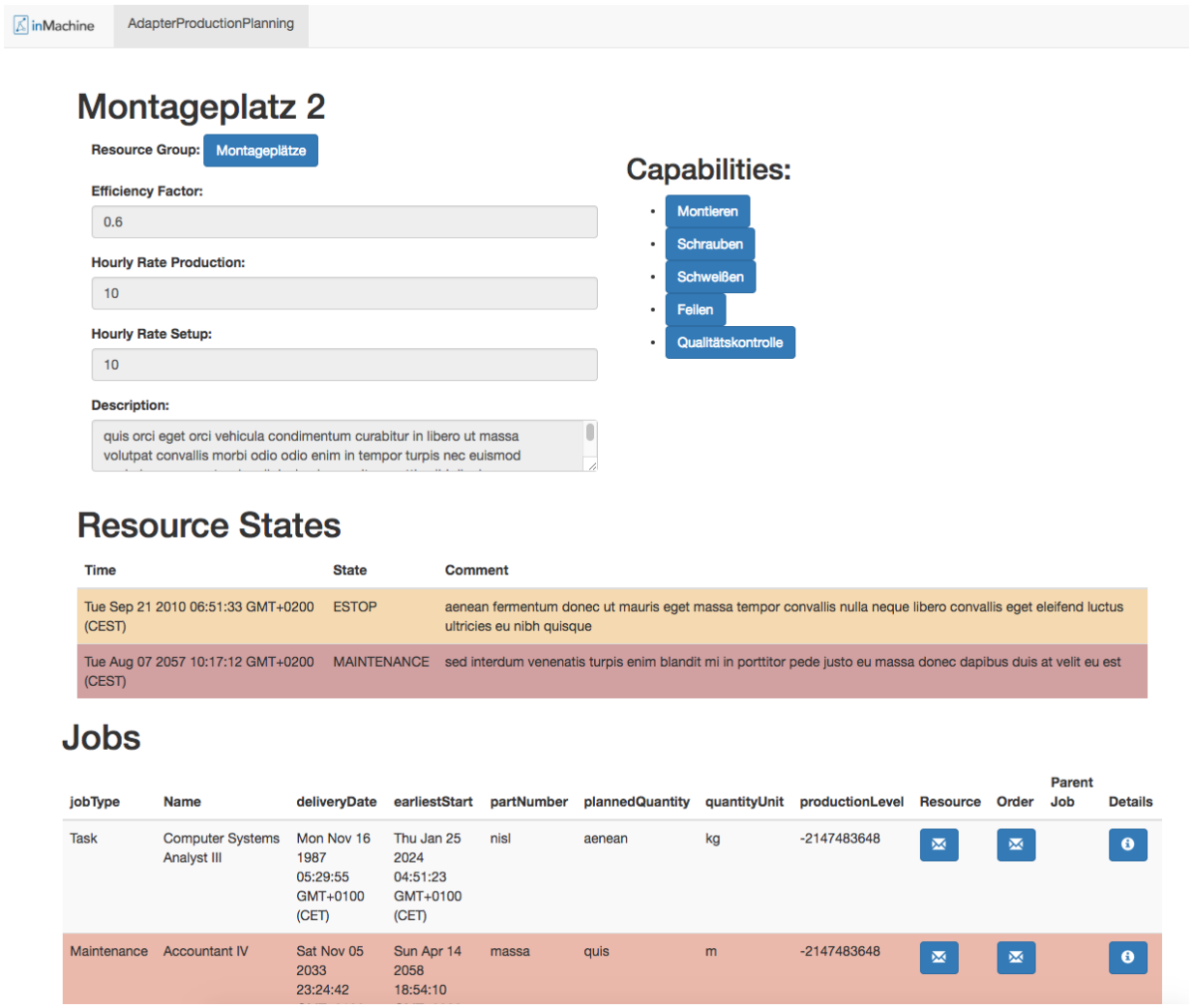


Abbildung 5.20: Übersicht des Meisters/Technikers über eine bestimmte Ressource

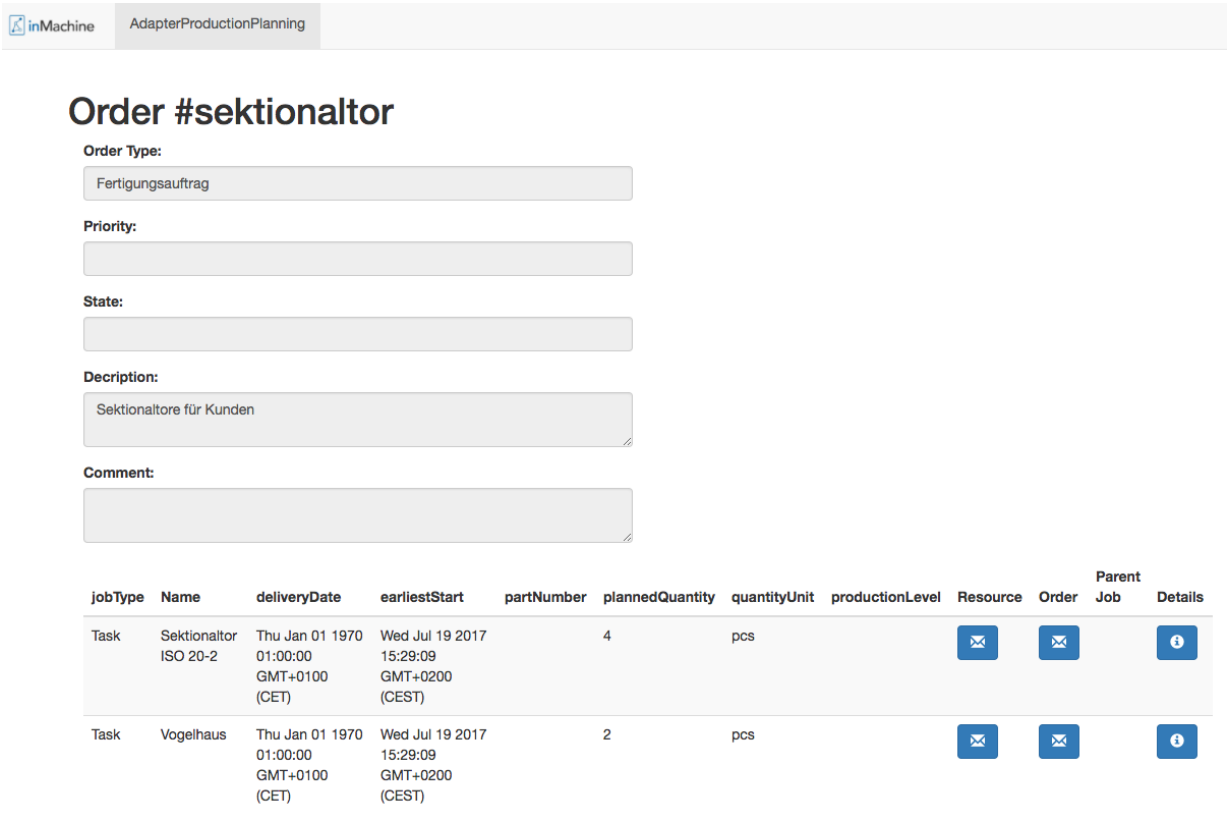


Abbildung 5.21: Übersicht des Meisters/Technikers über eine Bestellung

Der Benutzer mit der Rolle 'Arbeiter' hat eine eingeschränktere Sicht auf die Produktion. Er kann nur die Aufträge einsehen, die seiner Ressource zugeordnet sind (vgl. Abbildung 5.22). Der Benutzer kann sich die Details der einzelnen Aufträge und deren Arbeitsgänge ansehen und diese in Bearbeitung nehmen (vgl. Abbildung 5.23). Einem Auftrag können Dateien angefügt werden, die der Benutzer zur Produktion der Güter nutzen kann. Ein begonnener Auftrag kann abgebrochen oder abgeschlossen werden.

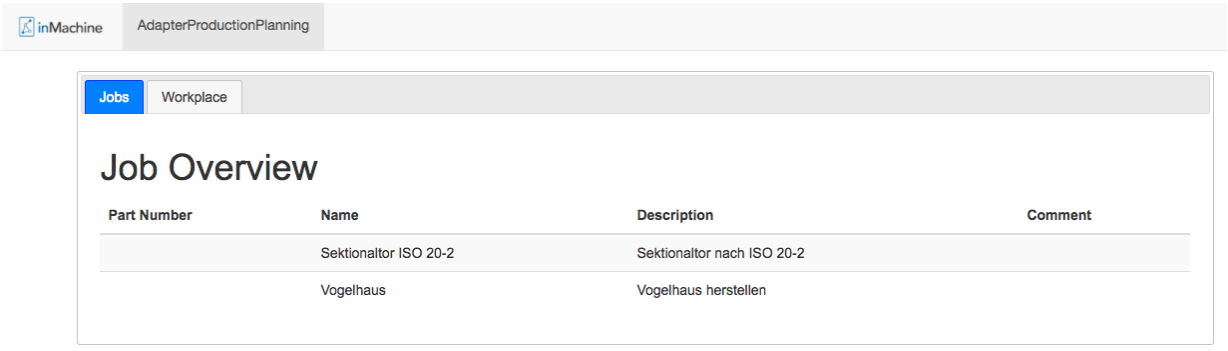


Abbildung 5.22: Übersicht des Arbeiters über alle Aufträge, die der Ressource an der dieser arbeitet zugewiesen sind.

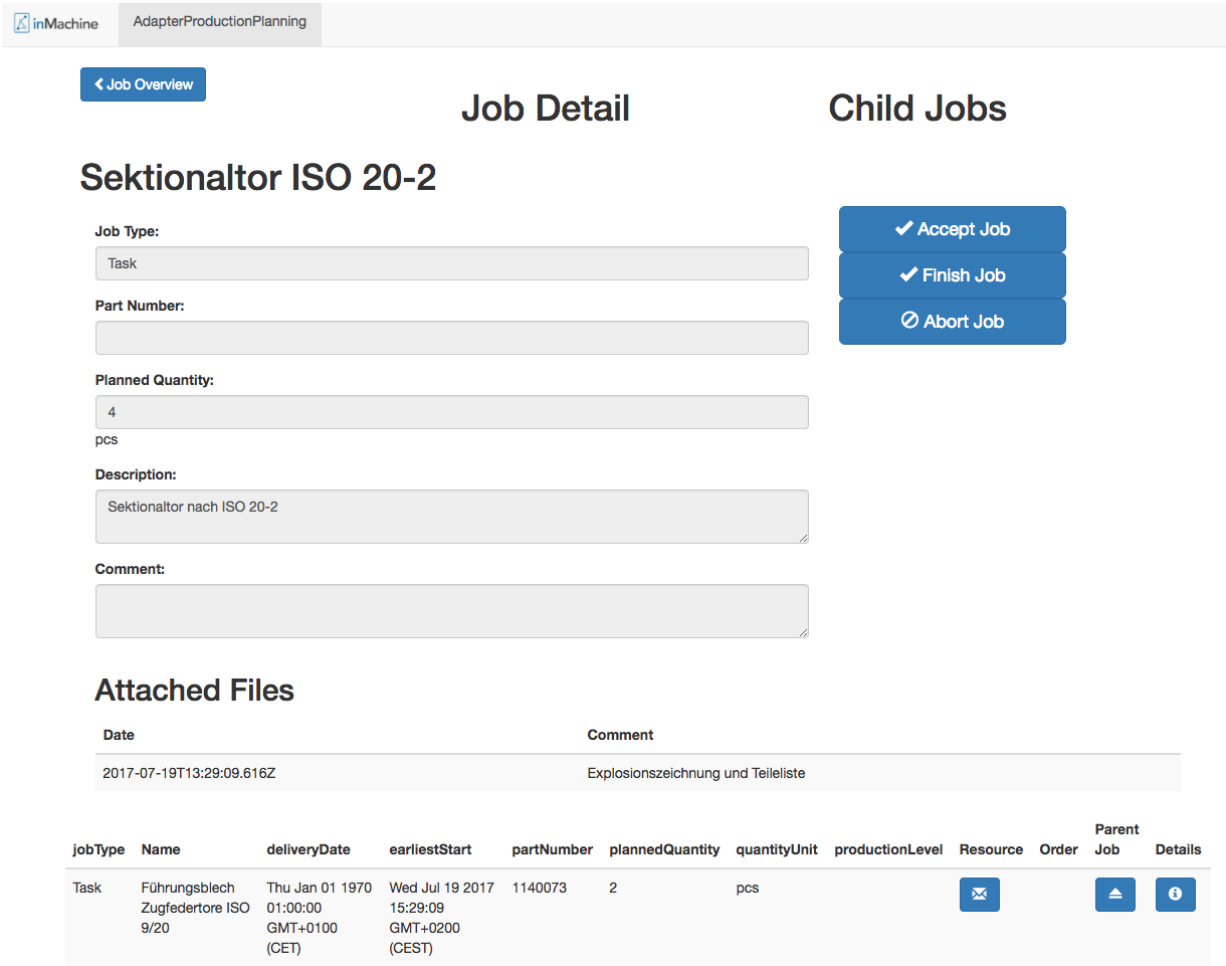


Abbildung 5.23: Übersicht des Arbeiters über einen Auftrag. Der Arbeiter kann einen Auftrag in Bearbeitung nehmen, abschließen oder abbrechen.

huge

5.7 Projektergebnis

Innerhalb dieses Kapitels wird die Einhaltung der Anforderungen validiert, sowie das Vorgehen kritisch reflektiert.

5.7.1 Validierung der Anforderungen

In Kapitel 2.3 wurden die Anforderungen an das Gesamtsystem definiert. Im Folgenden wird überprüft ob die Anforderungen umgesetzt wurden, siehe Tabelle 5.1.

Nummer	Beschreibung	Umsetzung
ME.1	Als Meister muss ich die Zugewiesenen Aufträge eines Arbeitsplatzes einsehen können, um die Richtigkeit zu prüfen und ggf. Optimierungen durchzuführen.	Die Komponente ProductionPlanning stellt hierzu eine Weboberfläche bereit, diese nutzt angebotene Rest-Services, die wiederum das Interface IAdapterProductionPlanning nutzen, um die benötigten Informationen abzufragen (vgl. Kapitel 5.3.2). Optimierungen können über das PPS-System angestoßen werden.)
ME.2	Als Meister möchte ich den Fortschritt eines Auftrages einsehen können, damit ich die Einhaltung des Produktionsplans kontrollieren kann.	Über die Komponente ProductionPlanning kann der Fortschritt eines Auftrages an das eingesetzt PPS-System gemeldet werden. Dadurch kann der Meister die benötigten Informationen über das PPS-System abfragen, aber auch über die Weboberfläche, die ihm die ProductionPlanning bereitstellt.
ME.3	Als Meister möchte ich über unvorhergesehene Ereignisse im Produktionsprozess, wie z.B. den Ausfall einer Maschine informiert werden, damit ich geeignete Maßnahmen durchführen kann.	Die Feedback Komponente bietet die Möglichkeit Mitarbeiter, Techniker, und Meister über unvorhergesehene Ereignisse zu informieren.
ME.4	Als Meister möchte ich die Messwerte eines Arbeitsplatzes oder einer Maschine einsehen können, um die Ordnungsmäßigkeit kontrollieren zu können.	Die Data Measurement Komponente ist in der Lage Sensordaten von externen Sensoren entgegen zu nehmen. Diese Daten werden durch die Schnittstelle der Production Planning Komponente dargestellt.

Nummer	Beschreibung	Umsetzung
ME.5	Als Meister möchte ich mein Wissen einfließen lassen können, um Unregelmäßigkeiten im Produktionsprozess aufdecken zu können.	Die Rule Engine Komponente bietet eine Schnittstelle die es dem Meister ermöglicht sein Expertenwissen über die Definition von Regeln einzubringen.
MI.1	Als Mitarbeiter möchte ich die Aufträge einsehen können, die meinem Arbeitsplatz zugewiesen wurden, damit ich diese abarbeiten kann.	Die Production Planning Komponenten bietet die Visualisierung der zugewiesenen Aufträge, welche durch das HMI am Arbeitsplatz eingesehen werden können.
MI.2	Als Mitarbeiter möchte ich den Fortschritt der Abarbeitung melden können, damit bei Abweichungen die Planung angepasst werden kann.	Die Production Planning Komponente bietet dem Mitarbeiter die Möglichkeit über das HMI den Fortschritt der Abarbeitung melden zu können.
MI.3	Als Mitarbeiter möchte ich unvorhergesehene Ereignisse, wie z.B. den Ausfall einer Maschine oder die Produktion von Ausschuss melden können, damit ggf. die Planung angepasst werden kann.	Über die Feedback Komponente können unvorhergesehene Ereignisse gemeldet werden.
MI.4	Als Mitarbeiter möchte ich über Unregelmäßigkeiten an meinem Arbeitsplatz, wie z.B die erhöhte Temperatur einer Maschine informiert werden, damit ich geeignete Maßnahmen einleiten kann.	Die Production Planning Komponente bietet über das HMI den Einblick in die erfassten Daten der externen Sensoren.
T.1	Als Techniker möchte ich die Wartungsaufträge einer Maschine einsehen können, um die entsprechende Wartung durchführen zu können.	Die Komponente ProductionPlanning stellt hierzu dem Techniker eine Oberfläche bereit.
T.2	Als Techniker möchte ich den Fortschritt einer Wartung melden können, damit die Produktion entsprechend geplant werden kann.	Die Production Planning Komponente bietet dem Techniker die Möglichkeit über das HMI den Fortschritt der Wartung melden zu können.

Tabelle 5.1: Validierung der Anforderungen

5.7.2 Kritische Reflektion

Im Folgenden wird das Vorgehen und die getroffenen Entscheidungen kritisch reflektiert.

5.7.2.1 Rule Engine Erweiterung

Das erstellte System sieht Regeln der folgenden Form vor:

> WENN Event [UND Bedingung] DANN Action

Dadurch ist keine Verkettung von Regeln möglich und die RuleEngine ist in seiner Erweiterbarkeit beschränkt. Würde zusätzlich die Möglichkeit geboten, dass eine Regel in einem Zustand resultiert und der Wechsel des Zustandes ein Event darstellt, würden sich Möglichkeiten zur Verkettung von Regeln ergeben. Im Folgenden wird dies an einem Beispiel dargelegt:

> 1. WENN dataMeasured UND Temperatur 10 Minuten lang > 50 DANN Zustand: Temperatur erhöht
> 2. WENN Zustand eintritt: Temperatur erhöht DANN showInformation
> 3. WENN Zustand eintritt: Temperatur erhöht DANN sendEventToAllMaintainers
> 4. WENN dataMeasured UND Zustand ist: Temperatur erhöht UND Temperatur 10 Minuten lang > 50 DANN Zustand: Wartung

In einem Agenten sind fünf Regeln definiert. Wenn 10 Minuten lang die gemessene Temperatur über 50 Grad liegt, dann wechselt der Agent in den Zustand 'Temperatur erhöht' (Regel 1). Der Wechsel des Zustandes hat zur Folge, dass dem Mitarbeiter, der mit dem Agenten arbeitet, diese Information angezeigt wird (Regel 2) und dass alle Techniker über den Zustandswechsel informiert werden (Regel 3). Dies wäre eine Verkettung von Regeln, mit dem bisherigen System ließe sich dasselbe Ergebnis erreichen, aber die Regeln würden dann redundante Bedingungen erhalten, was die Wartung erschweren würde und lange Verkettungen würden in sehr langen Bedingungen resultieren.

Zudem wäre es wünschenswert, wenn sich Regeln definieren ließen, deren Bedingung den aktuellen Zustand enthalten. In dem Beispiel wechselt der Agent vom Zustand 'Temperatur erhöht' in den Zustand 'Wartung', wenn die Temperatur weitere 10 Minuten über 50 Grad liegt (Regel 4). So ließe sich ein Mehrwert schaffen, es können Zustände für einen Agenten definiert werden und Regeln angegeben werden, die nur in einem bestimmten Zustand des Agenten ausgeführt werden können.

Das Problem, das dabei auftreten kann ist, dass Regeln unerwünschter weise einen Zustand überschreiben können. Wenn in unserem Beispiel der Agent in dem Zustand 'Temperatur erhöht' ist und 10 Minuten lang ein Sensorwert gemessen wird der über 50 Grad liegt werden sowohl Regel 1 als auch Regel 4 ausgeführt. Die Reihenfolge der Ausführung bestimmt den Endzustand des Agenten. Es kann zu unerwünschtem Verhalten kommen.

Die Nutzer müssen daher die Möglichkeit haben ein Zustandsautomaten zu definieren und Regeln für die einzelnen Zustandsübergänge angeben können.

Dieser Zustandsautomat könnte als Petri-Netz modelliert werden, es wäre somit möglich zu prüfen, ob Zustände jemals erreicht und verlassen werden können.

Die Erweiterung der RuleEngine um Zustände würde einen Mehrwert schaffen wirft aber auch viele Probleme auf. Es müsste daher genauer untersucht werden, wie Nützlich diese Erweiterung wäre, dies war aus Zeitgründen innerhalb dieser Arbeit nicht möglich.

5.7.2.2 Konfiguration von Actions

Aktuell sieht der Ablauf, zusammengefasst, folgendermaßen aus. Es geschieht ein Event, dieses Event wird an die Rule Engine weitergeleitet. Die Rule Engine überprüft ob es für dieses Event eine Regel gibt, sollte dabei die Bedingung zutreffen wird eine Action ausgelöst. Die ausgelöste Action erhält dabei alle Daten die innerhalb des Events zur Verfügung standen. Die übergebenen Daten können innerhalb der Action verwendet werden.

Jedoch ist eine zusätzliche Konfiguration der Action aktuell nicht möglich. Daraus kann resultieren, dass es Actions gibt die prinzipiell sehr ähnlich sind, sich nur sehr fein unterscheiden, und daher im schlimmsten Fall fast exakt doppelt implementiert sind. Innerhalb der Softwareentwicklung möchte man doppelten Quelltext vermeiden. Sollte in dem Quelltext ein Fehler sein muss dieser korrigiert werden, ist der Quelltext jedoch doppelt vorhanden muss auch die zweite Stelle korrigiert werden. Dies ist in der Regel sehr fehleranfällig, weil die zweite Stelle übersehen werden kann. Eine mögliche Lösung ist die Generalisierung des Quelltextes und das Verwenden in den jeweiligen Actions, dadurch resultiert eine Implementierung und die jeweiligen Feinheiten sind in den einzelnen Actions definiert. Dieses Vorgehen bekämpft jedoch nur die Symptome und nicht die Ursachen.

Im Folgenden wird die Erweiterung der Actions um ihre Konfigurierbarkeit vorgestellt. Abbildung 5.24 kann die Erweiterung der Action Interfaces entnommen werden. Durch die verwendete Rule Engine, siehe Kapitel 5.3.1, sind die Interfaces *IExecutableAction* und *IAction* bereits vorgegeben. *IExecutableAction* ermöglicht die Ausführung einer Action. *IAction* erlaubt die Identifizierung der Action durch ihren Namen. Die Erweiterung der Actions um ihre Konfigurierbarkeit geschieht in *IActionConfigurable*. Durch die Methode *getConfigurations():String[]* muss eine Action alle ihre Konfigurationsmöglichkeiten als String-Array herausgeben. Die Methode *setChoosenConfigurations(configuration:String[])* erlaubt es die ausgewählte Konfiguration für die Action zu setzen. Bei der Implementierung der Action kann nun folgendermaßen vorgegangen werden. Die Action gibt ihre möglichen Konfigurationen nach außen hin bekannt und implementiert die möglichen Fälle. Beim Ausführen einer Action wird nun vorher die gewünschte Konfiguration übergeben und es werden nur die Fälle ausgeführt, die in der übergebenen Konfiguration ausgewählt sind.
Ein geeignetes Beispiel kann Kapitel 5.4 entnommen werden. Für die Schaltung einer Ampel werden drei ActionTypes definiert, die in ausführbaren Actions resultieren.

- „switchTrafficLightGreen“ - schaltet die Ampel auf grün.
- „switchTrafficLightYellow“ - schaltet die Ampel auf gelb.

- „switchTrafficLightRed“ - schaltet die Ampel auf rot.

Mit Hilfe der Erweiterung *IActionConfigurable* ist es nun möglich diese drei ActionTypen zu einer *switchTrafficLight* zusammenzufassen. Die Methode *getConfigurations():String[]* gibt nun die drei Möglichkeiten „Green“, „Yellow“, und „Red“ zurück. Bevor nun die konkrete Action aufgerufen wird kann man durch *setChoosenConfigurations(configuration:String[])* festlegen welchen Status die Ampel darstellen soll.

Um die gewünschte Konfiguration speichern zu können muss zusätzlich noch das Datenmodell der Regel angepasst werden. Ein String-Array mit dem Namen „actionTypeConfiguration“muss *Rule* hinzugefügt werden.

Abschließend muss die neuen Funktionalität dem späteren Benutzer zugänglich gemacht werden. Dazu wird die Benutzeroberfläche für das Erstellen von Regeln erweitert. Nach dem der gewünschte ActionType ausgewählt wurde, werden dem Benutzer die zur Verfügung stehenden Konfigurationen in Form von Auswahlboxen angezeigt. Nun kann der Benutzer die gewünschte Konfiguration auswählen und diese wird innerhalb der Regel gespeichert. Trifft nun ein neues Event bei der Rule Engine ein, wird überprüft ob Regeln vorhanden sind und ihre Bedingungen geprüft. Werden die Bedingungen positiv evaluiert wird eine Action vom hinterlegten ActionType mit der gespeicherten Konfiguration ausgeführt.

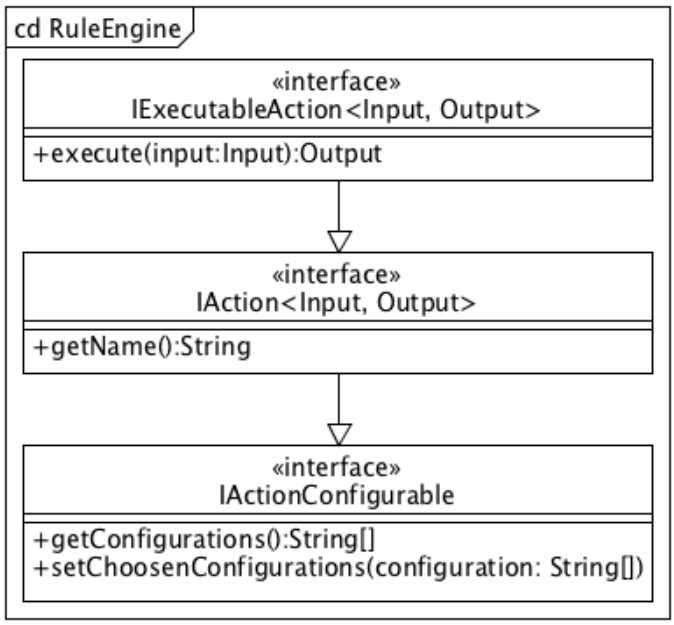


Abbildung 5.24: Erweiterung Actions

5.7.2.3 Allgemeines Modell für das PPS System

Im Zuge der Arbeit wurde ein allgemein gehaltenes Modell erstellt, das den Produktionsprozess abbilden können soll, um in verschiedenen KMUs eingesetzt werden zu können. In

der Retrospektive scheint das Modell nicht generisch genug zu sein. Die Zustände, die eine Ressource, Job oder Operation annehmen kann sind zum Beispiel vorgegeben und nicht änderbar. Diese können aber in jedem Unternehmen unterschiedlich sein, daher hätte dass Modell hier flexibler sein müssen. Die Zustände müssten frei definiert werden können, um das System auf die Bedürfnisse der KMUs anpassen zu können.

5.8 Zusammenfassung und Fazit

Nach der Erstellung des Softwarekonzeptes wurde dieses prototypisch implementiert. Dieses Vorgehen sichert die Qualität und Konsistenz des Softwarekonzeptes ab. Zunächst wurden die eingesetzten Technologien ausgewählt. Für das Grundgerüst fiel die Entscheidung auf Spring Boot, dies ermöglicht einen soliden Unterbau der Software, welcher bereits die gewünschten Prinzipien von REST einhält und eine Verteilung des Systems ermöglicht. Für die Persistenz wurde die NoSQL-Datenbank MongoDB ausgewählt, sie ermöglicht durch den dokumentenorientierten Ansatz eine vereinfachte Speicherung von Objekten. Zur Definition der Schnittstellen wurde Open API herangezogen. Open API ist ein herstellerunabhängiges Beschreibungsformat zur Definition und Dokumentation von Schnittstellen. Open API bietet den Vorteil die Dokumentation und den Quelltext für diverse Plattformen und Sprachen generieren zu können. Um das Softwarekonzept umsetzen zu können wurde folgende Projektstruktur verwendet.

- Das Projekt „Agent“ dient der Zusammenfassung aller benötigten Komponenten, um den gewünschten Agenten zusammenzusetzen. Dies bietet die Möglichkeit abhängig von dem Zielszenario nur die benötigten Komponenten auszuwählen und einen Agenten zu konfigurieren, der ideal auf die gewünschte Situation passt.

- Das Projekt „Common“ dient der Definition und Bereitstellung von allgemeingültigen Objektdefinitionen für alle Projekte, die sie benötigen. Dadurch werden mehrdeutige Objektdefinitionen und das Problem der Inkonsistenz vermieden.

- Das Projekt „Data Measurement“ entspricht der Definition innerhalb des Softwarekonzeptes und ist für das Entgegennehmen und Verarbeiten von externen Sensordaten zuständig.

- Das Projekt „Production Planning“ dient der abstrakten Kommunikation mit dem PPS-System. Zusätzlich dient es als lokales PPS-System für die Verwaltung der zugeordneten Jobs. Für die konkrete Kommunikation dient das Projekt „Adapter Production Planning“, welches abhängig von dem tatsächlich eingesetzten PPS-System implementiert ist.

- Das Projekt „Adapter Production Planning“ dient der konkreten Kommunikation mit dem PPS-System und ist abhängig von dem eingesetzten PPS-System implementiert.

- Das Projekt „Feedback“ entspricht der Definition innerhalb des Softwarekonzeptes und ist für das Bereitstellen einer Schnittstelle zur Rückmeldung des Produktionsprozesses zuständig.
- Das Projekt „Rule Engine“ entspricht der Definition innerhalb des Softwarekonzeptes und ist für das Entgegennehmen von Events, der Verarbeitung von Regeln, und dem Ausführen von Actions zuständig.

Zusätzlich wurde ein Beispiel für die Erweiterbarkeit des Softwarekonzeptes gegeben, dies soll aufzeigen, dass das Softwarekonzept über die gewünschte Erweiterbarkeit verfügt. Anschließend wurden die, anhand der im Softwarekonzept beschriebenen, Benutzerschnittstellen umgesetzt und aufgezeigt. Abschließend wurde die Umsetzung kritisch reflektiert. Dies resultiert darin, dass die aktuelle Umsetzung der Actions erweiterungswürdig wäre. Die Actions verfügen aktuell über keine Konfigurierbarkeit, das Erweitern um diese Funktionalität im Bereich der Actions und Regeln würde in der Vermeidung von doppeltem Quelltext und einer geringeren Anzahl von ActionTypes resultieren.

Kapitel 6

Abschluss

Innerhalb dieses Kapitels wird eine Zusammenfassung der gesamten Arbeit gegeben. Anschließend wird ein Fazit gezogen. Abschließend wird ein Ausblick gewährt und das weitere Vorgehen dargelegt.

6.1 Zusammenfassung

Unternehmen stehen auf Grund von Globalisierung, Konkurrenzdruck und immer schneller agierenden Märkten vor der Herausforderung auf diese Veränderungen flexibel reagieren zu müssen. Unternehmen die sich schnell auf Marktveränderungen einstellen können haben einen Wettbewerbsvorteil der beibehalten werden muss. Beim verarbeitenden Gewerbe resultiert das in einer Optimierung der Produktionsplanung- und Steuerung. Um eine Optimierung der Produktionsplanung und -steuerung vornehmen zu können muss zunächst Einblick in diese zur Verfügung stehen. KMUs sind in der Regel nicht in der Lage die Kosten und Komplexität von Softwarelösungen von Herstellern wie Siemens, Dassault, oder SAP zu handhaben. Aufgrund dessen ist es notwendig Softwarelösungen anzubieten, die genau auf das Einsatzszenario in KMUs zugeschnitten sind.

Ziel dieser Arbeit war das Erstellen einer Softwarelösung, um eine Optimierung der Produktionsplanung- und Steuerung zu ermöglichen. Um dieses Ziel zu erreichen wurden zunächst Anforderungen an das Gesamtsystem gestellt. Diese Anforderungen fließen in das zu erstellende Softwarekonzept ein. Durch die Anforderung an Unternehmen flexibel auf Marktveränderungen reagieren zu können, resultiert auch die Anforderung an die eingesetzte Software, flexibel auf neue Gegebenheiten angepasst werden zu können.

Das Erstellen eines Softwarekonzeptes benötigt das Treffen von diversen Entwurfsentscheidungen, die das Gesamtsystem ausschlaggebend beeinflussen. Zunächst fiel die Entscheidung ein verteiltes System zu realisieren. Dies ist notwendig da im verarbeitendem Gewerbe die Ressourcen zum Produzieren von Gütern räumlich verteilt sind. Durch ein zentrales System ist die Möglichkeit der Erfassung von Sensordaten und die Rückmeldung des Produktionsprozesses an einzelnen Arbeitsplätzen nicht möglich. Beim Softwarekonzept wurde besonders auf die lose Koppelung der Komponenten und die flexiblen Kommunika-

tion geachtet. Dadurch ist es möglich das Softwarekonzept zukunftssicher aufzustellen und das nachträgliche Erweitern der Software zu ermöglichen. Zur flexiblen Kommunikation der Komponenten wurde eine Event-basierte Kommunikation verwendet. Die diversen Komponenten können Events auslösen, diese werden von dem regelbasierten Expertensystem verarbeitet und können in Anweisungen an andere Komponenten resultieren.

Ein entscheidender Punkt ist die Möglichkeit das Expertenwissen der Mitarbeiter in das Gesamtsystem einfließen lassen zu können, daher wurde ein regelbasiertes Expertensystem eingerichtet. Ein regelbasiertes Expertensystem bietet den Vorteil, dass die benötigte Geschäftslogik durch die Mitarbeiter eingepflegt oder verändert werden kann, ohne die Notwendigkeit den Quelltext anpassen zu müssen.

Die diversen Funktionalitäten wurden in Komponenten aufgeteilt, um eine lose Koppelung und die flexible Erweiterbarkeit der Gesamtlösung zu ermöglichen. Die Event-basierte Kommunikation ermöglicht es, nachträglich Komponenten in das Gesamtsystem einzufügen. Darüber hinaus erhält man die Möglichkeit der Konfigurierbarkeit. Es ist möglich den Agenten nur mit den Komponenten auszustatten die für das Einsatzszenario notwendig sind.

Zusätzlich wurden beim Softwarekonzept die Prinzipien von Webanwendungen, ins besondere Representational State Transfer (REST), berücksichtigt. Es wurde auf die Trennung von Client & Server geachtet, die Zustandslosigkeit der Kommunikation, die Cachebarkeit von Informationen, das Bereitstellen von einheitlichen Schnittstellen, die Möglichkeit der Abbildung in hierarchischen Schichten, und die Funktionalität von Code-On-Demand. Darüber hinaus werden Websockets eingesetzt um eine asynchrone und bidirektionale Verbindung zwischen Client und Server aufbauen zu können. Durch Websockets erhält man eine aufrechterhaltene Verbindung wodurch eine verzögerungsarme Kommunikation möglich ist.

Ein Agent kann aus folgenden Komponenten zusammengesetzt werden. Die Data Measurement Komponente ist dafür verantwortlich während des Produktionsprozesses Sensordaten zu sammeln. Sie bietet dafür eine Schnittelle an, um von externen Sensoren Daten zu erhalten. Die Production Planning Komponente hat die Aufgabe der Visualisierung des Produktionsprozesses. Es werden die Aufträge dargestellt die einem Arbeitsplatz zugeordnet sind. Zusätzlich werden Rückmeldungen und Statusänderungen an das PPS-System gemeldet. Die Adapter Production Planning Komponente ist verantwortlich für die Kommunikation mit dem PPS-System, sie ist das Bindeglied zwischen der Production Planning Komponente und dem eingesetzten PPS-System, daher muss sie abhängig von dem eingesetzten PPS-System angepasst werden. Die Feedback Komponente bietet dem Benutzer die Möglichkeit Feedback zu geben und über Ereignisse benachrichtigt zu werden. Die Rule Engine Komponente deckt das regelbasierte Expertensystem ab. Sie prüft, ob für ausgelöste Events Regeln vorhanden sind und löst ggf. die zugeordneten Anweisungen aus.

Die Integration der Softwarelösung in einen bestehenden Arbeitsplatz kann Abbildung 4.9 entnommen werden. Es ist vorgesehen jeden Arbeitsplatz mit einem eigenen Agenten auszustatten. Dieser Agent verfügt zum einen über, am Arbeitsplatz angebrachte, externe Sensoren, zum Erfassen von Sensordaten, und zum anderen ein HMI zur Interaktion mit dem Mitarbeiter.

Bei der Gestaltung der Benutzeroberflächen wurde auf Übersichtlichkeit und Verständlichkeit geachtet.

Das Thema Sicherheit ist von hoher Relevanz. Um die Kommunikation zwischen den Komponenten und diversen Drittsystem abzusichern wird HTTPS eingesetzt, dadurch werden jegliche übertragenen Daten verschlüsselt und sind nicht für Dritte einsehbar. Zusätzlich ist ein rollenbasiertes Benutzerkonzept notwendig. Jeder Funktionalität innerhalb des System wird mit der Angabe über die benötigte Rolle versehen. Anschließend werden Mitarbeitern diese Rollen zugeordnet, dadurch werden Funktionen nur für die berechtigten Mitarbeiter nutzbar sind.

Nach der Konzeptionierung des Softwarekonzeptes wurde eine prototypische Implementierung durchgeführt. Die prototypische Implementierung sichert die Qualität und die Konsistenz des Softwarekonzeptes ab. Es wurden zunächst die eingesetzten Technologien vorgestellt. Als Grundlage wurde Spring Boot eingesetzt. Spring Boot ermöglicht unter anderem eine vereinfachte Implementierung von Schnittstellen, die dem REST Programmierparadigma genügen. Für die Persistenz wurde die NoSQL-Datenbank MongoDB ausgewählt, sie ermöglicht durch den dokumentenorientierten Ansatz eine vereinfachte Speicherung von Objekten. Zur Definition der Schnittstellen wurde Open API herangezogen. Open API ist ein herstellerunabhängiges Beschreibungsformat zur Definition und Dokumentation von Schnittstellen. Open API bietet den Vorteil die Dokumentation und den Quelltext für diverse Plattformen und Sprachen generieren zu können.

Das Softwarekonzept ergab folgende Projektstruktur:

- Das Projekt „Agent“ dient der Zusammenfassung aller benötigten Komponenten um den gewünschten Agenten zusammenzusetzen. Dies bietet die Möglichkeit abhängig von dem Zielszenario nur die benötigten Komponenten auszuwählen und einen Agenten zu konfigurieren, der ideal auf die gewünschte Situation passt.

- Das Projekt „Common“ dient der Definition und Bereitstellung von allgemeingültigen Objektdefinitionen für alle Projekte die, sie benötigten. Dadurch werden mehrdeutige Objektdefinitionen und das Problem der Inkonsistenz vermieden.

- Das Projekt „Data Measurement“ entspricht der Definition innerhalb des Softwarekonzeptes und ist für das Entgegennehmen und Verarbeiten von externen Sensordaten zuständig.

- Das Projekt „Production Planning“ dient der abstrakten Kommunikation mit dem PPS-System. Zusätzlich dient es als lokales PPS-System für die Verwaltung der zugeordneten Jobs. Für die konkrete Kommunikation dient das Projekt „Adapter Production Planning“, welches abhängig von dem tatsächlich eingesetzten PPS-System implementiert ist.

- Das Projekt „Adapter Production Planning“ dient der konkreten Kommunikation mit dem PPS-System und ist abhängig von dem eingesetzten PPS-System implementiert.

- Das Projekt „Feedback“ entspricht der Definition innerhalb des Softwarekonzeptes und ist für das Bereitstellen einer Schnittstelle zur Rückmeldung des Produktionsprozesses zuständig.

- Das Projekt „Rule Engine“ entspricht der Definition innerhalb des Softwarekonzeptes und ist für das Entgegennehmen von Events, der Verarbeitung von Regeln, und dem ausführen von Actions zuständig.

Zusätzlich wurde ein Beispiel für die Erweiterbarkeit des Softwarekonzeptes gegeben, dies soll aufzeigen, dass das Softwarekonzept über die gewünschte Erweiterbarkeit verfügt. Anschließend wurden die, anhand der im Softwarekonzept beschriebenen, Benutzerschnittstellen umgesetzt und aufgezeigt.

Die Umsetzung wurde kritisch reflektiert. Dies resultierte darin, dass die aktuelle Umsetzung der Actions erweiterungswürdig wäre. Die Actions verfügen aktuell über keine Konfigurierbarkeit, das Erweitern um diese Funktionalität im Bereich der Actions und Regeln würde in der Vermeidung von doppeltem Quelltext und einer geringeren Anzahl von ActionTypes resultieren.

6.2 Fazit

Das Ziel der Arbeit war es, ein MAS aufzubauen. Dieses ermöglicht es den Produktionsprozess zu überwachen und somit die Reaktionsfähigkeit der KMU zu verbessern. Dieses Ziel wurde durch das Konzipieren eines Softwarekonzeptes und dessen prototypischen Implementierung erfüllt.
Es ist eine Softwarelösung entstanden die sich besonders durch die lose Koppelung der Komponenten, die flexible Event-basierte Kommunikation, und der daraus resultierenden Erweiterbarkeit, auszeichnet. Bei der prototypischen Implementierung hat sich die Qualität und die Realisierbarkeit des Softwarekonzeptes erproben lassen. Dabei hat sich gezeigt, dass das Softwarekonzept keine hersteller- oder plattformspezifischen Eigenschaften enthält, wodurch eine Umsetzung auf anderen Plattformen und mit anderen Programmiersprachen möglich ist. Dies ist ein weiterer Faktor, welcher die Zukunftssicherheit des Softwarekonzeptes aufzeigt.

6.3 Ausblick

Durch die vorgestellte Softwarelösung besteht die Möglichkeit für KMUs den Einblick in den Produktionsprozess zu erhöhen.

Der erste Schritt wäre die Praxistauglichkeit des Gesamtsystems in einem realen Kontext zu prüfen. Dazu müssten diversen KMUs herangezogen werden, um die vorgestellt Softwarelösung validieren zu können. Dies wird im Kontext des inMachine-Projektes geschehen.

Der nächste Schritt besteht darin die, KMUs noch stärker bei der Planung ihrer Produktionsprozesse zu unterstützten. Eine Mögliche Lösung für dieses Problem ist die Verlagerung der Planung, weg von einer zentralen Instanz, hin zu einer dezentralen Lösung. Das bedeutet, dass nicht eine globale zentrale Instanz die komplette Planung übernimmt. Die Planung soll ferner mithilfe der zur Verfügung stehenden Agenten durchgeführt werden. Wie bereits in dem Kapitel 3 vorgestellt wurde, wird den Agenten viel mehr Eigenständigkeit zugestanden und die Problematik der Verteilung von Jobs durch Auktionen realisiert. Ein möglicher Ablauf wäre folgender: Ein neuer Job steht zur Verteilung bereit. Jeder Agent überprüft wie optimal dieser Job in die aktuelle Planung passen würde und gibt für diesen Job ein Gebot ab.

Darüber hinaus würde die Kommunikation der Agenten untereinander gefördert werden. Sollte ein Arbeitsplatz unvorhergesehen gewartet werden müssen stellt der Agent, der diesen überwacht, seine aktuell zugeteilten Jobs zur Verteilung frei. Die Agenten übernehmen nun selbständig die Verteilung der Jobs, ohne die Notwendigkeit einer zentralen Instanz. Dies soll in einer Optimierung des Verteilungsprozesses resultieren und die Auslastung der Arbeitsplätze erhöhen.

Das in Kapitel 4 vorgestellte Softwarekonzept ermöglicht die Erweiterung der Agenten-Software um Komponenten zur Auktion und Kommunikation der Jobs, wodurch die Integration einer Lösung in das bestehende Softwarekonzept keiner technischen Einschränkung unterliegt.

Wie bereits in Kapitel 1 beschrieben wurde, werden häufig im Laufe eines Produktionsprozesses ein Teil der Fertigungsschritte an Zulieferer bzw. Veredler ausgelagert. In diesem überbetrieblichen Kontext ist die Datenverfügbarkeit verringert. Das Interesse an einer unternehmensübergreifenden Überwachung des Auftragsstatus und die Information über Störungen, die einen termingetreuen Ablauf gefährden könnten, ist sehr groß. Der Wunsch besteht darin auch Daten unternehmensübergreifend in die Planungs- und Steuerungsebene einfließen lassen zu können, um die Reaktionsfähigkeit des eigenen Unternehmens zu erhöhen.

Um den beschriebenen Anforderungen gerecht zu werden wird eine unternehmensübergreifende Schnittstelle geschaffen die zwei Funktionen erfüllen soll. Das Zulieferer-Unternehmen kann den aktuellen Auftragsstatus einsehen, um beispielsweise die Abholung der Werkstückes und damit die eigenen Planung zu optimieren. Darüber hinaus kann das Zuliefer-Unternehmen Rückmeldung über den aktuellen Auftragsstatus liefern, damit z.B. der genaue Ankunftszeitpunkt des Werkstückes bekannt ist und darauf aufbauend die eigene Planung verbessert werden kann.

Eidesstattliche Erklärung

Gemäß § 16 Abs. 5 der MPO erkläre ich an Eides statt, dass ich die vorliegenden gekennzeichneten Anteile der Arbeit selbständig angefertigt habe. Ich habe mich keiner fremden Hilfe bedient und keine anderen, als die angegebenen Quellen und Hilfsmittel benutzt. Alle Stellen, die wörtlich oder sinngemäß veröffentlichten oder nicht veröffentlichten Schriften und anderen Quellen entnommen sind, habe ich als solche kenntlich gemacht. Diese Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

Dortmund, 21. Juli 2017

(David Grimm)	(Andreas J. Wojtok)

Literaturverzeichnis

[AGN+10] Johan Akerberg, Mikael Gidlund, Jonas Neander, Tomas Lennvall, and Mats Bjorkman. Deterministic downlink transmission in WirelessHART networks enabling wireless control applications. pages 2120–2125. IEEE, November 2010.

[AHT90] James Allen, James A. Hendler, and Austin Tate, editors. *Readings in Planning*. The Morgan Kaufmann series in representation and reasoning. Morgan Kaufmann Publishers, San Mateo, Calif, 1990.

[AU09] John Langshaw Austin and James Opie Urmson. *How to Do Things with Words: [The William James Lectures Delivered at Harvard University in 1955]*. Harvard Univ. Press, Cambridge, Mass, 2. ed., [repr.] edition, 2009. OCLC: 935786421.

[Aus72] John Langshaw Austin. *Zur Theorie der Sprechakte: (How to do things with words)*. Number Nr. 9396-98 in Universal-Bibliothek. P. Reclam, Stuttgart, 1972.

[Avc03] Oral Avci, editor. *Web-Programmierung: Softwareentwicklung mit Internet-Technologien - Grundlagen, Auswahl, Einsatz - XHTML & HTML, CSS, XML, JavaScript, VBScript, PHP, ASP, Java*. Vieweg, Wiesbaden, 1. aufl edition, 2003. OCLC: 76506169.

[BH05] Frank Boos and Barbara Heitger. *Wertschöpfung im Unternehmen: Wie innovative interne Dienstleister die Wettbewerbsfähigkeit steigern*. Springer-Verlag, January 2005. Google-Books-ID: vfuYWivk8v4C.

[BL11] Helmut Balzert and Peter Liggesmeyer. *Lehrbuch der Softwaretechnik. [2]: Entwurf, Implementierung, Installation und Betrieb*. Lehrbücher der Informatik. Spektrum, Akad. Verl, Heidelberg, 3. aufl edition, 2011. OCLC: 750951360.

[Büt11] Ricardo Büttner. *Automatisierte Verhandlungen in Multi-Agenten-Systemen*. PhD thesis, Gabler Verlag | Springer Fachmedien Wiesbaden GmbH, Wiesbaden, Wiesbaden, 2011.

[Con86] Conway, Melvin E. Committees Paper. http://www.melconway.com/Home/Committees_Paper.html, 1986.

[DeT89] Arthur DeTour. An Introduction to Expert Systems. In *Journal of Insurance Medicine*, volume 21, 1989.

[Dic15] Philipp Dickmann. *Schlanker Materialfluss: mit Lean Production, Kanban und Innovationen*. Springer-Verlag, September 2015.

[DN97] Avinash K. Dixit and Barry J. Nalebuff. *Spieltheorie für Einsteiger: strategisches Know-how für Gewinner*. Schäffer-Poeschel, Stuttgart, 1997. OCLC: 75746232.

[DP10] Waltenegus Dargie and Christian Poellabauer. *Fundamentals of Wireless Sensor Networks: Theory and Practice*. Wiley series on wireless communications and mobile computing. Wiley, Chichester, West Sussex, U.K. ; Hoboken, NJ, 2010.

[DP17] Dr. Sabine Graumann and Prof. Dr. Irene Bertschek. Wirtschaft DIGITAL 2017. Technical report, Bundesministerium für Wirtschaft und Energie (BMWi), Berlin, June 2017.

[eBa] eBay. Ebay. http://www.ebay.de/.

[Fet11] Ian Fette. The WebSocket Protocol. https://tools.ietf.org/html/rfc6455, 2011.

[Fie00] Roy Thomas Fielding. *Architectural Styles and the Design of Network-Based Software Architectures*. PhD thesis, University of California, Irvine, 2000.

[FLM95] Tim Finin, Yannis Labrou, and James Mayfield. KQML as an agent communication language. Technical report, Computer Science and Electrical Engineering University of Maryland Baltimore County Baltimore MD USA, September 1995.

[Fou16] Wikimedia Foundation. Vermaschtes Netz, November 2016. Page Version ID: 159980837.

[Fow03] Martin Fowler. *Patterns of Enterprise Application Architecture*. The Addison-Wesley signature series. Addison-Wesley, Boston, 2003.

[Fow04] Martin Fowler. Inversion of Control Containers and the Dependency Injection pattern. https://martinfowler.com/articles/injection.html, 2004.

[Fow14] Martin Fowler. Microservices. https://martinfowler.com/articles/microservices.html, March 2014.

[FWW+93] Tim Finin, Jay Weber, Gio Wiederhold, Michael Genesereth, Richard Fritzson, Donald McKay, James McGuire, Richard Pelavin, Stuart Shapiro, and Chris Beck. Specification of the KQML. https://www.csee.umbc.edu/csee/research/kqml/kqmlspec/spec.html, 1993.

[Gab17] Gabler Wirtschaftslexikon. Definition » PPS-System « | Gabler Wirtschaftslexikon. http://wirtschaftslexikon.gabler.de/Archiv/72918/pps-system-v10.html, 2017.

[Gar16] Gartner. Top 10 Technology Trends Impacting Infrastructure & Operations - Smarter With Gartner. http://www.gartner.com/smarterwithgartner/top-10-technology-trends-impacting-infrastructure-operations/, 2016.

[Gol11] Joachim Goll. *Methoden und Architekturen der Softwaretechnik*. Studium. Vieweg + Teubner, Wiesbaden, 1. aufl edition, 2011. OCLC: 747803493.

[Haa08] Oliver Haase. *Kommunikation in verteilten Anwendungen: Einführung in Sockets, Java RMI, CORBA und Jini*. Oldenbourg, München, 2., überarb. und erw. aufl edition, 2008. OCLC: 239409695.

[KGJ13] Sebastian Kummer, Oskar Grün, and Werner Jammernegg, editors. *Grundzüge der Beschaffung, Produktion und Logistik.* wi - Wirtschaft. Pearson Studium, München, 2., aktualisierte aufl., [nachdr.] edition, 2013. OCLC: 857994694.

[Kut17] Ant Kutschera. Rules: Ant's simple rule engine for Java, Java 8, Scala and Node.js, May 2017.

[Lan09] Volker Lanninger. *Prozessmodell zur Auswahl betrieblicher Standardanwendungssoftware für KMU*. BoD – Books on Demand, 2009. Google-Books-ID: GZI18PtO1AgC.

[LBB17] Uwe Lämmel, Anatoli Beifert, and Marcel Brätz. Business Rules Die Wissensverarbeitung erreicht die Betriebswirtschaft Einsatzmöglichkeiten und Marktübersicht. (5), 2017.

[LFP99] Yannis Labrou, Tim Finin, and Peng, Yun. Agent communication languages: The current landscape. *IEEE Intelligent Systems*, 14(2):45–52, March 1999.

[LMP04] Michael Luck, Peter McBurney, and Chris Preist. *A Manifesto for Agent Technology: Towards Next Generation Computing.* Number 9 in Autonomous Agents and Multi-Agent Systems. 2004.

[LMSW05] Michael Luck, Peter McBurney, Onn Shehory, and Steve Willmott. *Agent Technology: Computing as Interaction - A Roadmap for Agent Based Computing.* AgentLink III, September 2005.

[Men05] Daniel A. Menasce. MOM vs. RPC: Communication Models for Distributed Applications. *IEEE Internet Computing*, 9(2):90–93, March 2005.

[Mey10] Mike Meyers. *CompTIA A+ All in One: Prüfungsvorbereitung und Hardware-Buch ; [aktuell zu den neuen A+-Prüfungen 220-701 Essentials und 220-702 Practical application].* mitp, Heidelberg, 4. aufl edition, 2010. OCLC: 845740311.

[Mey16] Mike Meyers. *CompTIA A+ Certification All-in-One Exam Guide (Exams 220-901 & 220-902).* All-in-one. McGraw-Hill Education, New York, ninth edition edition, 2016.

[Mik16] Mike Brock. Language Guide for MVEL 2.0. `http://mvel.documentnode.com/`, 2016.

[Moi] Alireza Moini. Smart sensors. `https://www.iee.et.tu-dresden.de/iee/analog/papers/mirror/visionchips/vision_chips/smart_sensors.html`.

[Mon17] MongoDB. MongoDB. `https://www.mongodb.com/de`, 2017.

[Nol09] Marco Nolte. *Entwicklung eines CAN-Bus-Adapters für spezielle Anforderungen zur Fahrzeuganbindung.* Europ. Hochsch.-Verl, Bremen, 1. aufl edition, 2009. OCLC: 845594653.

[ope17] open api. Open API. `https://www.openapis.org/`, 2017.

[PB14] Prof. Dr. Wolfram Höpken and Bärbel Häußler. Business Intelligence (BI) – Daten sammeln, aufbereiten und analysieren. Technical report, Oberschwaben-Ulm, July 2014.

[Pul00] Torben Pullmann. Implementierung eines wissensbasierten Softwaresystems zum Entwurf und zur Bemessung von Stahlbeton-Tragwerken. 2000. Institut für Numerische Methoden und Informatik im Bauwesen.

[PX06] Yi Pan and Yang Xiao, editors. *Ad Hoc and Sensor Networks.* Number v. 2 in Wireless networks and mobile computing. Nova Science, New York, 2006. OCLC: ocm59003376.

[Res00] Eric Rescorla. HTTP Over TLS. `https://tools.ietf.org/html/rfc2818`, 2000.

[RMHCMH09] Mark Richards, Richard Monson-Haefel, David A. Chappell, and Richard Monson-Haefel. *Java Message Service*. O'Reilly, Sebastopol, CA, 2nd ed edition, 2009.

[Ros85] Jeffrey Rosenschein. *Rational Interaction: Cooperation among Intelligent Agents*. PhD thesis, Stanford University, 1985.

[Sch] Patrick Schnabel. IEEE 802.11s / Wireless Mesh Network. `http://www.elektronik-kompendium.de/sites/net/1408051.htm`.

[Sch05] Peter Scholz. *Softwareentwicklung eingebetteter Systeme: Grundlagen, Modellierung, Qualitätssicherung*. Springer, Berlin, 2005. OCLC: 64388320.

[Spr17] Spring. 2. Introduction to the Spring Framework. `http://docs.spring.io/spring-framework/docs/current/spring-framework-reference/html/overview.html`, 2017.

[SSLMD14] Casimir Saternos, Simon St. Laurent, Allyson MacDonald, and Rebecca Demarest. *Client-Server Web Apps with JavaScript and Java*. O'Reilly Media, Inc, Sebastopol, CA, first edition edition, 2014. OCLC: ocn878854657.

[swa17] swagger. Swagger – The World's Most Popular Framework for APIs. `http://swagger.io/`, 2017.

[TV01] Dirk H Traeger and Andreas Volk. *LAN Praxis lokaler Netze*. Vieweg+Teubner Verlag, Wiesbaden, 2001. OCLC: 863923938.

[Vei03] Daniel J. Veit. *Matchmaking in Electronic Markets: An Agent-Based Approach towards Matchmaking in Electronic Negotiations*. Springer, December 2003. Google-Books-ID: 7tZtCQAAQBAJ.

[Win92] Patrick Henry Winston. *Artificial Intelligence*. Addison-Wesley Pub. Co, Reading, Mass, 3rd ed edition, 1992.

[WJ95] Michael Wooldridge and Nicholas Jennings. *Intelligent Agents: Theory and Practice*. Number 10 in The Knowledge Engineering Review. 1995.

[Woo09] Michael J. Wooldridge. *An Introduction to Multiagent Systems*. John Wiley & Sons, Chichester, U.K, 2nd ed edition, 2009. OCLC: ocn246887666.

[WSM13] Vanessa Wang, Frank Salim, and Peter Moskovits. *The Definitive Guide to HTML5 WebSocket*. The expert's voice in web development. Apress, Berkeley, Calif., 2013. OCLC: ocn821699929.

Abbildungsverzeichnis

Tabellenverzeichnis

Abkürzungsverzeichnis

ERP Enterprise Ressource Planning.

HMI Human Machine Interface.

HTML Hyper Text Markup Language.

HTTP Hyper Text Transfer Protokoll.

JMS Java Message Service.

JS Javascript.

KMU kleine und mittelständische Unternehmen.

KQML Knowledge Query Manipulation Language.

MAS Multiagentensystem.

MOM Message oriented Middleware.

MVEL MVFLEX Expression Language.

PLM Product Lifecycle Management.

PPS-System Produktionsplanungs- und Steuerungssystem.

RPC Remote Procedure Call.

Glossar

Buildmanagement Bei Projekten müssen Ressourcen unterschiedlicher Art und Herkunftsquelle zu einem Ergebnis, dem so genannten Build, zusammengefasst werden. Dabei sollten Builds während ihrer Herstellung mit einer Build-Nummer eindeutig identifiziert werden können. Builds sollten regelmäßig ablaufen und reproduzierbar sein. Dieses Vorgehen nennt sich Buildmanagement.

Cash-to-Cash-Cycle-Time Zeitdauer, die das Kapital von der Beschaffung des Materials für ein Produkt, bis zur Bezahlung des Produktes durch den Kunden gebunden ist [KGJ13].

Human Machine Interface Benutzerschnittstelle.

Hyper Text Markup Language Beschreibungssprache, die vor allem für die Strukturierung digitaler Dokumente genutzt wird. HTML-Dokumente sind die Grundlage des World Wide Web und können von Webbrowsern dargestellt werden. [Avc03].

Hyper Text Transfer Protokoll Das Hyper Text Transfer Protokoll ist ein Protokoll zur Übertragung von Dateien über ein Rechnernetz. Es findet Hauptsächlich verwendung im World Wide Web. [Avc03].

Java Message Service JMS ist die Java API zum ansteuern einer MOM. Es ist Teil der Java Enterprise Edition un ermöglicht das Senden und Empfangen von Nachrichten. [RMHCMH09].

Javascript Javascript ist eine Scriptsprache, die ursprünglich in HTML-Dokumenten Anwendung fand. Sie wird dazu genutzt diese Dokumente dynamischer zu gestalten. Dazu wird Javascript vom Webbrowser interpretiert. Inzwischen wird Javascript auch außerhalb von HTML-Dokumenten genutzt, beispielsweise als Server. [SSLMD14] .

Logging Unter Logging wird das Führen eines Protokolls verstanden, in dem verschiedene Aktionen der Anwendung dokumentiert werden. Es werden alle Aktionen mitgeschrieben, die für spätere Untersuchungen wichtig sind.

Message oriented Middleware Die Message oriented Middleware bezeichnet eien Schicht, die auf der Übertragung von Nachrichten beruht. Diese Nachrichten ermöglichen sowohl eine synchrone als auch asynchrone Kommunikation. [RMHCMH09].

Multiagentensystem Bei einem Multiagentensystem handelte es sich um einen Zusammenschluss von Softwareagenten mit dem Ziel, Fähigkeiten & Pläne abzustimmen, um koordiniert zu handeln und Probleme zu lösen. [Ros85] [WJ95] [FLM95] [LMSW05].

Netflix Netflix ist ein Unternehmen, dass einen großen Streaming Dienst betreibt, der sich auch mit der Produktion von Filmen und Serien beschäftigt.

Polyglot Persistence Polyglot Persistence beschreibt den Ansatz, mehr als eine Datenbank für eine Anwendung zu benutzen. Die Datenbanken können verschiedenartig sein, sodass diese abhängig vom Verwendungszweck ausgewählt werden.

Produktionsplanungs- und Steuerungssystem „Unter PPS-Systemen werden computergestützte Produktionsplanungs- und -steuerungssysteme verstanden, die zur operativen Planung und Steuerung des Produktionsgeschehens in einem Industriebetrieb eingesetzt werden.“ [Gab17].

Queue Queues (Warteschlangen) sind unkt-zu-Punkt-Verbindung, die zwischen genau einem Sender und einem Empfänger aufgebaut werden. Wenn ein Empfänger eine Nachricht erhält, wird diese aus dem Nachrichtenkanal entfernt.[RMHCMH09].

Remote Procedure Call Remote Procedure Call ist eine Technik, die den Aufruf von Funktionen in anderen Adressräumen ermöglicht. Oft werden diese genutzt, um Funktionen auf einem anderen Computer aufzurufen.

Topic Topics (Themen) haben in der Regel viele Empfänger. Sender publizieren Nachrichten und Empfänger abbonieren diese. Wenn ein Empfänger eine Nachricht erhält, bleibt diese, anders als bei einer Queue weiter im Nachrichtenkanal, damit auch weitere Empfänger die Nachricht abgreifen können.[RMHCMH09].

World Wide Web Consortium Das W3C ist eine internationale Gemeinschaft, die sich mit der Entwicklung von Standards für das Internet beschäftigt .

YAML YAML ist eine einfache Auszeichnungssprache, die an die Extensible Markup Language (XML) angelehnt ist..

Anhang

Installationsanleitung

Im Folgenden werden die nötigen Schritte zur Installation und Einrichtung des Prototypen erläutert. Die Installation wird sowohl für Linux, als auch Windows Betriebssysteme erklärt.

Voraussetzungen

Um den Prototypen zu installieren wird mindestens ein JRE 8 benötigt. Dieses kann unter http://www.oracle.com/technetwork/java/javase/downloads heruntergeladen werden.

Übersetzen des Quellcodes

Der Quellcode und die Binaries befindet sich auf dem beigefügten Datenträger. Für den Fall, das es nötig seien sollte selber Binaries aus dem Quellcode zu erstellen werden die nötigen Schritte erläutert. Für das Übersetzen des Quellcodes wird mindestens ein JDK 8 benötigt. Die einzelnen Projekte können mit dem Buildwerkzeug Gradle[1] gebaut werden. Abbildung 6.1 und 6.2 zeigen den Befehl zum Erstellen der Binaries, dieser muss in dem entsprechenden Projektverzeichnis z.B. inMachine ausgeführt werden.

Die ausführbaren Binaries befinden sich anschließend im Ordner *build/libs*.

Installation der MongoDB

Der Prototyp benötigt die Anbindung an eine MongoDB. Die MongoDB kann unter http://www.mongodb.org/downloads für das entsprechende Betriebssystem heruntergeladen werden. Für die Installation nutzen Sie bitte die offizielle Anleitung http://docs.mongodb.org/manual/installation/. Um die MongoDB zu starten muss ein

[1] https://gradle.org/

```
# gradlew.bat clean build
```

Abbildung 6.1: Erstellen der Binaries unter Windows

```
# ./gradlew clean build
```

Abbildung 6.2: Erstellen der Binaries unter Linux

```
# mkdir /opt/mongo
# mkdir /opt/mongo/db
# mongod --fork --dbpath /opt/mongo/db --logpath /opt/mondo/log
```

Abbildung 6.3: Startanweisungen der MongoDB unter Linux

Datenbankpfad erstellt werden und ein Pfad für die Log-Datei(vgl. Abbildung 6.3 und 6.4).

Installation des ActiveMQ (optional)

Für die Installation des Apache ActiveMQ's halten Sie sich bitte an die offizielle Anleitung: http://activemq.apache.org/getting-started.html. Sollte kein ActiveMQ installiert werden können, können die Agenten nicht untereinander kommunizieren, das Betreiben der Agenten ist dennoch möglich.

Installation des ProductionPlanningMock

Zur Installation des ProductionPlanningMock wird die jar-Datei in das Installationsverzeichnis kopiert.

Die Anwendung erwartet, dass eine Mongo-DB unter 'localhost:27017' erreichbar ist. Sollte die Mongo-DB unter einer anderen Adresse erreichbar sein, muss eine 'application.yml' neben die jar-Datei gelegt werden. Mit dieser lässt sich der Pfad zur Mongo-DB konfigurieren vgl. Abbildung 6.6.

Der Befehl zum Starten der Anwendung ist in Abbildung 6.7 ersichtlich. Nach dem Start wird ein Beispieldatensatz generiert. Die Anwendung läuft auf dem Port 8081. Der Port kann in der 'application.yml' angepasst werden vgl. Abbildung 6.5.

```
# mkdir mongo
# chdir mongo
# mkdir db
# mongod.exe --dbpath mongo\db --logpath mongo\log
```

Abbildung 6.4: Startanweisungen der MongoDB unter Windows

```
server:
  port: <PORT>
```

Abbildung 6.5: Konfiguration des Ports in der application.yml.

```
spring:
    data:
        mongodb.host: <DNS-NAME>
        mongodb.port: <PORT>
```

Abbildung 6.6: Inhalt der application.yml, um eine andere URL für die MongoDB zu konfigurieren.

```
# java -jar ./productionPlanningMock.jar
```

Abbildung 6.7: Startanweisungen der Anwendung ProductionPlanningMock

Registrieren als Dienst

Unter Linux kann die Anwendung als Dienst unter *init.d* oder *systemd* registriert werden. Dazu nutzen Sie bitte die offizielle Anleitung von Spring-Boot https://docs.spring.io/spring-boot/docs/current/reference/html/deployment-install.html#deployment-service.

Installation des Agenten

Nach der erfolgreichen Installation und Konfiguration des ProductionPlanningMock kann ein Agent installiert werden. Dazu muss die jar-Datei in den Installationsordner kopiert werden.

Neben der jar-Datei wird nun eine 'application.yml' angelegt, in der der PPS-Server, in diesem Fall der ProductionPlanningMock, konfiguriert wird vgl. Abbildung 6.10.

Der Agent erwartet, dass eine Mongo-DB unter 'localhost:27017' erreichbar ist. Sollte die Mongo-DB unter einer anderen Adresse erreichbar sein, lässt sich der Pfad zur Mongo-DB in der 'application.yml' konfigurieren vgl. Abbildung 6.6.

Zur Kommunikation der Agenten untereinander wird ein MessageBroker benötigt, dieser lässt sich über die 'application.yml' konfigurieren vgl. Abbildung 6.8.

Außerdem wird in der 'application.yml' die Rolle des Agenten definiert. Dazu werden die Topics angegeben, an denen sich ein Agent registrieren soll. Ein Mitarbeiter registriert sich für die Topics 'worker.one' und 'worker.all', ein Meister für 'expert.one' und 'expert.all' und ein Techniker für 'maintainer.one' und 'maintainer.all' (vgl. Abbildung 6.9).

Ein Agent überwacht immer eine Ressource, daher ist es nötig anzugeben, welche Ressource eine Agent überwacht. Dies wird ebenfalls über die 'application.yml' gemacht vgl. Abbildung 6.11.

```
spring:
        activemq:
                broker-url: tcp://<DNS-NAME>:<PORT>
                user: <NUTZERNAME>
                password: <PASSOWRT>
```

Abbildung 6.8: Konfiguration der ActiveMQ Adresse und Zugangsdaten, in der application.yml

```
destination:
        single: <"worker.one" | "expert.one" | "maintainer.one">
        all: <"worker.all" | "expert.all" | "maintainer.all">
```

Abbildung 6.9: Konfiguration der Rolle durch definieren der Topics, in der application.yml

```
inMachineAdapter.url: http://<DNS-NAME>:<PORT>/api/v1
```

Abbildung 6.10: Konfiguration des PPS-Systems, in der application.yml

```
inMachine:
        m2mId: <ResourceId>
```

Abbildung 6.11: Konfiguration der ResourceId, die ein Agent überwacht, in der application.yml

Registrieren als Dienst

Unter Linux kann die Anwendung als Dienst unter *init.d* oder *systemd* registriert werden. Dazu nutzen Sie bitte die offizielle Anleitung von Spring-Boot https://docs.spring.io/spring-boot/docs/current/reference/html/deployment-install.html#deployment-service.

Inhalt des Datenträgers

Im Folgenden wird der Inhalt des beigefügten Datenträgers aufgelistet.

- source_code.zip: Der Quellcode und die Binaries der einzelnen Komponenten
  - inMachine: Projekt in dem alle Komponenten zusammengeführt werden. Entspricht dem „Agent“ Projekt im Softwarekonzept.
  - inMachine-bom: Projekt in dem die Kompatibilität der Projekte untereinander definiert werden kann. Das Projekt generiert eine Bill of Material (BOM), die zum Auflösen von Versionen genutzt wird.
  - inMachine-common: Projekt in dem Klassen definiert wurden, die von verschiedenen Projekten genutzt wurden.
  - inMachine-dataMeasurement: Projekt der dataMeasurement Komponente.
  - inMachine-feedback: Projekt der feedback Komponente.
  - inMachine-productionPlanning: Projekt der productionPlanning Komponente.
  - inMachine-adapterProductionPlanningMock: Projekt der adapterProductionPlanningMock Komponente.
  - inMachine-productionPlanningMock: Projekt der productionPlanningMock Komponente.
  - inMachine-specification: Projekt in dem alle OpenApi Definitionen gesammelt werden.
  - inMachine-ruleEngine: Projekt der ruleEngine Komponente.
- Master_Thesis_Grimm_Wojtok.pdf: Die Master Thesis in digitaler Form.
- Performance Test
  - testPlan.xlsx : Testplan und Ergebnisse des Performancetests der Rule Engine.