Konzeption und Implementation eines Expertensystems zur Überwachung der Sensorfertigung bei der Firma epro GmbH

zur Erlangung des akademischen Grades
Bachelor of Science (FH)

vorgelegt im Studiengang
Angewandte Informatik
des Fachbereichs Elektrotechnik und Informatik der Fachhochschule
Münster

von
Christoph Zuleger

Prüfer: Prof. Dr. rer. nat. Nikolaus Wulff
Betreuer: Dipl.-Ing. Peter Tasche

Münster, 12. September 2009
1 Danksagung

Mein besonderer Dank gilt Herrn Prof. Dr. rer. nat. Nikolaus Wulff, ohne dessen Unterstützung dieses Projekt nicht zu Stande gekommen wäre.


Auch möchte ich mich im Besonderen bei meinen Eltern Maria und Hans-Jürgen für ihre Unterstützung bedanken.

Abschließend danke ich meiner Lebensgefährtin Marina Gerliz für das Korrekturlesen meiner Arbeit.
Inhaltsverzeichnis

1 Danksagung

2 Einleitung
  2.1 Aufbau der Arbeit ........................................ 1
  2.2 Das Unternehmen ........................................... 2
    2.2.1 Geschichte ........................................... 2
    2.2.2 Geschäftsbereiche ....................................... 2
  2.3 Motivation .................................................. 3
  2.4 Zielsetzung ................................................ 3

3 Einführung
  3.1 Wirbelstromsensoren ......................................... 5
    3.1.1 Grundlagen magnetischer Felder ...................... 5
    3.1.2 Funktionsweise ......................................... 7
  3.2 Expertensysteme ............................................. 8
    3.2.1 Unterteilung der Wissensrepräsentation ............ 8
    3.2.2 Der RETE-Algorithmus .................................. 9
    3.2.3 Regelsyntax bei Drools ................................. 12
  3.3 Neuronale Netze ............................................ 13
    3.3.1 Bestandteile ........................................ 13
    3.3.2 Aufbau ................................................ 14
    3.3.3 Trainingsphase ........................................ 14
  3.4 Reguläre Ausdrücke ........................................ 15
    3.4.1 Arten von Regex-Maschinen ......................... 15
    3.4.2 Zeichenklassen ......................................... 16
    3.4.3 Gruppierungen und Alternationen ................... 16
    3.4.4 Quantoren ............................................ 17
    3.4.5 Backtracking .......................................... 18
    3.4.6 Possessive Quantoren und atomare Gruppen ....... 18

4 Konzeption .................................................. 20
  4.1 Anforderungsanalyse ...................................... 20
    4.1.1 Produktionsphasen .................................. 20
    4.1.2 Ist-Zustand ........................................... 21
    4.1.3 Funktionalität ......................................... 23
2 Einleitung

Im Rahmen dieser Arbeit entstand ein verteiltes Expertensystem. Zielsetzung dieses Systems ist eine kontinuierliche Analyse des Produktionszyklus und die grafische Aufbereitung der erzeugten Ergebnisse innerhalb der Sensorfertigung der Firma epro GmbH.

Als zentraler Bestandteil für die Festlegung der erwarteten Produkteigenschaften dienen die aus dem JBoss Projekt stammende Inferenzmaschine Drools. Die Darstellung der ermittelten Ergebnisse erfolgt über eine Swing-Oberfläche.

2.1 Aufbau der Arbeit

Das Kapitel Einleitung bietet dem Leser einen allgemeinen Überblick über das Projektumfeld, die zu lösende Problemstellung und die an das System gestellten Forderungen.

Der darauf folgende Abschnitt Einführung konzentriert sich auf die Vermittlung von Technologie-Wissen, das für das weitere Verständnis der Arbeit von zentraler Bedeutung ist.

Aufbauend auf diesem Wissen widmet sich das Kapitel Konzeption primär den Bereichen Anforderungsanalyse und Entwurf. Eine Darstellung und die anschließende Gegenüberstellung möglicher Lösungsansätze vervollständigt die Beschreibung der Konzeptionsphase.

Der umfangreiche Abschnitt Implementation dokumentiert neben den Komponenten des Expertensystems und deren Wechselwirkungen auch die Umsetzung der Konzeption in der bestehenden Infrastruktur.

Abschließend werden im Rahmen des Kapitels Ausblick und Fazit mögliche Erweiterungen, sowie Vor- und Nachteile der getroffenen Entscheidungen diskutiert.
2.2 Das Unternehmen


2.2.1 Geschichte


1985 Gründung einer Softwareabteilung


1991 Die Firma Thomson verkauft wegen Überkapazitäten die HSA Elektronik & Systemtechnik GmbH an 5 Investoren. Aus HSA wird epro ELEKTRONIK & SYSTEMTECHNIK GmbH.


2002 Im August werden die 90% - Anteile der Windhof AG an der epro von 4 Mitarbeitern des Unternehmens mit Unterstützung einer externen Investorengruppe übernommen.


2009 Übernahme der epro GmbH durch die Emerson Electric Co.

2.2.2 Geschäftsbereiche

Systemtechnik Der Geschäftsbereich Systemtechnik liefert Mechanik, Elektronik und Sensorik für Verpackungs- und Druckmaschinen, für die Militär- und Medizintechnik
2.3 Motivation

Trotz zunehmender Bemühungen der Politik und Wirtschaft Schadstoffemissionen und den Verbrauch an Rohstoffen zu senken, steigt in Deutschland insbesondere der Bedarf an elektrischer Energie [5]. Diese Tatsache hat einen unmittelbaren Einfluss auf die Krafwerkstätigkeit und deren Ansprüche an die Leistungsfähigkeit und Zuverlässigkeit der zur Stromerzeugung eingesetzten Turbinen.

Um diese sich scheinbar widersprechenden Forderungen zu erfüllen, ist eine ständige Überwachung der Anlagen notwendig. Die Sensoren zur Bestimmung der Wellenposition und deren Schwingung sind dabei unverzichtbare Indikatoren zur Früherkennung drohender Störfälle und werden aus diesem Grund aufwändig und somit kostenintensiv produziert.

Die, in Bezug auf die Präzision der Sensoren, sehr niedrige Toleranzschwelle und die zudem hohe Komplexität des Herstellungsprozesses führte jedoch in der Vergangenheit zu einem großen Anteil an nicht verwertbaren Produkten, die einen inakzeptablen Kostenfaktor darstellen.

2.4 Zielsetzung

Die generelle Zielsetzung des zu entwerfenden Softwaresystems ist die Reduzierung der Produktionskosten und eine Nachverfolgbarkeit der Produkteigenschaften. Die zuletzt

\[1\] MMS steht in diesem Kontext für Machine Monitoring Systems


Im Gegensatz zu dieser Herangehensweise bezieht sich der zweite Ansatz auf das Ermitteln und Erkennen von Ursachen für die Existenz fehlerhafter Sensoren. Der daraus entstehende Mehraufwand resultiert jedoch zusätzlich zu einer frühen Erkennung fehlerhafter Produkte in einem verbesserten Produktionsprozess und einer reduzierten Ausschussrate.


Zusammenfassend ergeben sich daraus die folgenden Anforderungen an die Lösung:

- Zentrale Speicherung von ermittelten Produkteigenschaften
- Analyse der Produkteigenschaften in Hinblick auf Verwendbarkeit
- Ermittlung von potentiellen Messfehlern
- Zeitnahe Bereitstellung des gewonnenen Wissens im Produktionsbereich
- Intuitive Aufbereitung der Analyseergebnisse
- Optional: Ermittlung von Parametern zur automatisierten Kalibrierung

Unabhängig vom gewählten Ansatz ist die Wartbarkeit und Erweiterbarkeit des Systems von entscheidender Bedeutung. Dies gilt insbesondere aufgrund der Tatsache, dass die Qualität der Analyse und die Anpassbarkeit an sich ändernde Rahmenbedingungen an eine kontinuierliche Weiterentwicklung gebunden sind.
3 Einführung


3.1 Wirbelstromsensoren


3.1.1 Grundlagen magnetischer Felder

Jeder elektrische Strom versetzt den ihn umgebenden Raum in einen besonderen Zustand, der sich dadurch äußert, dass Kräfte auf andere Ströme ausgeübt werden. Diesen besonderen Raumzustand bezeichnet man als magnetisches Feld [4, S. 131]. Um ein solches Feld zu beschreiben, existieren mehrere physikalische Größen, die im folgenden kurz erläutert werden.

Die magnetische Erregung Der Begriff magnetische Erregung ist ein Synonym für Feldstärke. Da aber die Stärke des Feldes auch von anderen Faktoren abhängt, schien diese häufiger benutzte Formulierung irreführend (vgl. [4, S. 134]). Die mit \( H \) gekennzeichnete gerichtete Größe ermöglicht eine Angabe des Feldverlaufs unter Vernachlässigung von Materialeigenschaften. Für eine langgestreckte, kreisförmige Spule gilt:

\[
H = \frac{IN}{l} \tag{3.1}
\]
3.1 Wirbelstromsensoren

I steht dabei für den in der Spule fließenden Strom, \( N \) für die Windungszahl und \( l \) für die Feldlinienlänge und beträgt bei einem Abstand \( r \) von einem langen geraden Leiter:

\[
l = 2\pi r
\]

(3.2)

Die magnetische Flussdichte und Permeabilität

Die mit dem Symbol \( B \) gekennzeichnete magnetische Flussdichte erweitert die Aussage der magnetischen Erregung durch die Berücksichtigung der Material Eigenschaften und beschreibt die Stärke eines Feldes:

\[
B = \mu_0 \mu_r H = \frac{\mu I N}{l}, \mu = \mu_0 \mu_r
\]

(3.3)

Der mit \( \mu \) bezeichnete Faktor steht in diesem Kontext für die Permeabilitätszahl des Materials, in dem sich das magnetische Feld ausbreitet (\( \mu_r \)) multipliziert mit einer Konstanten (\( \mu_0 \)). Dabei ist \( \mu_0 \) im Vakuum gleich 1 und liegt bei ferromagnetischen Stoffen\(^1\) zwischen \( 10^4 \) und \( 10^6 \).

Der magnetische Fluss

Die zuvor beschriebene Flussdichte kann als Feldliniendichte angesehen werden. Multipliziert man die vom Magnetfeld durchsetzte Fläche \( A \) mit dieser Dichte, so ergibt sich daraus der magnetischen Fluss. Für eine beliebige Fläche gilt nach [4, S. 139]:

\[
\Phi = \int_A \hat{B} \cdot d\vec{A}
\]

(3.4)

Der Vektor \( \hat{B} \) gibt die Richtung des Feldes an, das auf das durch den Vektor \( d\vec{A} \) repräsentierte Flächenelement trifft.

Die Induktionswirkung

Bewegt man einen Leiter durch ein magnetisches Feld, so ist eine Potentialdifferenz an den Leitungsenden festzustellen. Das Induktionsgesetz besagt:

"Die in einer Leiterschleife induzierte Spannung ist stets gleich der pro Zeiteinheit auftretenden Änderung des von der Schleife umfassen den magnetischen Flusses."\(^6\)

Alternativ zur Bewegung des Leiters, wird auch dann eine Spannung induziert, wenn sich die Richtung oder die Erregung des Feldes ändert.

\(^1\)Der Grund für die Verstärkung ist die Ausrichtung von Elementarmagneten in so genannten Weiß'schen Bezirken.
3.1 Wirbelstromsensoren

3.1.2 Funktionsweise


Abbildung 3.2 zeigt für 177 getestete Sensoren die daraus resultierenden Amplitudenänderungen in Abhängigkeit vom Abstand zum Zielkörper.

Die mit $U_{\text{Start}}$, $U_{\text{Mid}}$ und $U_{\text{End}}$ bezeichneten Größen entsprechen dabei den jeweils gemessenen Spannungsamplituden bei einer Distanz zum Zielkörper von 0,19, 1,23 und 2,23 Millimetern.
3.2 Expertensysteme


3.2.1 Unterteilung der Wissensrepräsentation

Expertensysteme existieren in drei wesentlichen Ausprägungen:

**Logische Wissensrepräsentation** Die Grundlage für Logikschatungen aus dem Bereich der Digitaltechnik stellt die Aussagenlogik dar. So lässt sich mit Hilfe von Junktoren eine beliebig lange Aussage bilden, deren Wahrheitswert durch eine Auswertung der atomaren Teilaussagen bestimmt werden kann. Junktoren dienen dabei als Verbindungsglieder zwischen den Teilaussagen. Beispiele sind die Konjunktion (UND-Verknüpfung) und Disjunktion (ODER-Verknüpfung) von Ausdrücken:

\[(a \land b) \lor (a \land \bar{b}) \iff a\]
Erweitert man den Begriff der Aussagenlogik um Quantoren, die eine Mengenbildung erlauben, so resultiert dies in der Prädikatenlogik. Aus der Mathematik stammende Beispiele für Quantoren sind die einschränkenden Symbole $\exists$ ("es existiert ein Element") und $\forall$ ("für alle Elemente gilt"): 

$$\forall\varepsilon\exists\delta \mid \delta \geq \varepsilon$$

Dies entspricht der Formulierung: für alle Epsilon existiert ein Delta größer oder gleich Epsilon.


### 3.2.2 Der RETE-Algorithmus

In ihrer Anwendung werden an Produktionssysteme hohe Anforderungen in Bezug auf ihre Verarbeitungsgeschwindigkeit gestellt, da auch große Datenvolumina, wie sie bei
3.2 Expertensysteme

der Überwachung technischer Anlagen die Regel sind, in verhältnismäßig kurzer Zeit analysiert werden müssen. Wesentlich dafür ist eine effiziente und fallbezogene Filterung der Produktionsdaten. Der zwischen 1978 und 1979 im Rahmen seiner Promotion von Dr. Charles Forgy entwickelte Algorithmus sieht eine Gliederung in vier entscheidende Knoten vor:

- Root: Wurzelknoten
- 1-Input: Element mit einem Eingang
- 2-Input: Element mit zwei Eingängen
- Terminal: Abschließender Knoten


**JBoss Rules / Drools**

Die im Rahmen des Projekts eingesetzte Inferenzmaschine Drools nutzt den RETE-Algorithmus in Form der folgenden Knotenunterteilung:

**ReteNode** Dieser Knoten entspricht der Wurzel des Graphen.

**ObjectTypeNode** Objekttypknoten dienen einer Effizienzsteigerung durch eine Unterteilung nach der Klasse eines Fakts. Die Erstellung erfolgt erst nach dem Eintreffen eines bislang unbekannten Typs.

**AlphaNode** Knoten dieses Typs dienen der Auswertung von literalen Vergleichen und können verkettet an einen Objekttypknoten angehängt werden.

**LeftInputAdapterNode** Dieser Knotentyp dient als Verbindungsglied.

**EvalNode** EvalNodes dienen der boolschen Auswertung einer Aussage.
3.2 Expertensysteme

**JoinNode** Diese Art von Knoten ermöglicht die Zusammenführung und den Vergleich unterschiedlicher oder gleicher Objekttypen und deren Attribute.

**NotNode** Ein NotNode entspricht einem negierten JoinNode.

**TerminalNode** Der Abschlussknoten.


Abbildung 3.3: Fallbeispiel des RETE-Graphen
Der blaue Elementtypknoten auf der linken Seite repräsentiert Instanzen der Klasse \( A \). Angehängte \( \textit{AlphaNodes} \) symbolisieren literale Vergleiche, wie sie Bestandteil der Bedingungen einer Regel sein können\(^2\). Ob jekte der Klasse \( B \), deren Einstiegspunkt auf der rechten Seite zu sehen ist, werden mit Elementen des Typs \( A \) durch einen Verbindungs knoten verknüpft. Auch dieser Knotentyp stellt eine Regelbedingung dar und führt bei Erfüllung zum Abschlussknoten (\( \text{TerminalNode} \)).

Für den Zugriff auf Attribute wird eine Namenskonvention vorrausgesetzt, die den im weiteren Verlauf der Arbeit erläuterten \textit{Java Beans} Vorgaben entspricht.

### 3.2.3 Regelsyntax bei Drools

Das der offiziellen Dokumentation von Drools 4.0.7 entnommene Diagramm \(^3\) zeigt den allgemeinen Aufbau einer Regel.

![Diagramm der Regelstruktur](image)

**Abbildung 3.4: Syntax einer Regel**

Eingeleitet wird eine Regel mit dem Schlüsselwort \textit{rule} gefolgt von ihrem Namen, den der Entwickler frei wählen darf\(^3\). Im Anschluss können der Regel optionale Attribute angegeben werden, die beispielsweise die Ausführungsreihenfolge betreffen. Einer der beiden Kernbestandteile einer Regel ist der Block an Bedingungen, dessen Anfang vom Schlüsselwort \textit{when} gekennzeichnet ist. Dieser als \textit{Left Hand Side} bezeichnete Abschnitt der Regel enthält sämtliche Einschränkungen, die in Form von \textit{Alpha-}, \textit{Eval}, \textit{Join-} und

\(^2\)z.B.: \( A.\text{stringAttr} == "xyz" \)

\(^3\)Es gibt bzgl. der Namensgebung einige Einschränkungen, auf die hier nicht weiter eingegangen wird.
3.3 Neuronale Netze


3.3.1 Bestandteile

Die Basis künstlicher neuronaler Netze bildet das Neuron als vereinfachtes Abbild einer biologischen Nervenzelle. Es zeichnet sich im Allgemeinen durch drei wesentliche Funktionen aus (vgl. [9, S. 26 ff]):

**Propagierung** Die in das Neuron eingehenden Verbindungen sind gewichtet und können entweder eine erregende oder auch eine hemmende Wirkung auf das Neuron haben. In Abhängigkeit von der Anzahl (n) und Wichtung (Weight) der eingehenden Verbindungen (Input) ergibt sich ein Nettoinput (Net). Für ein Neuron j gilt somit:

\[
Net_j = \sum_{i=1}^{n} (Weight_{ij} * Input_i)
\]

(3.5)

Der Nettoinput ist also die Summe aller gewichteten Ausgabezustände der sendenden Neuronen.

**Aktivierung** Eine Aktivierung des Neuron erfolgt erst, wenn die Erregung in Form des Nettoinputs eine bestimmte Grenze erreicht. Diese Grenze wird durch eine Aktivierungsfunktion (Act) festgelegt, die im einfachsten Fall aus den Werten des aktuellen Nettoinputs und der vorherigen Aktivierungsfunktion besteht:

\[
Act_j(t + 1) = F_j(Net_j(t + 1), Act_j(t))
\]

(3.6)

Wird diese Schwelle nicht erreicht, entfällt die Aktivierung der nachfolgend beschriebenen Ausgabefunktion.
Ausgabe Die Ausgabefunktion (Out) transformiert den inneren Aktivierungszustand, der an empfangende Neuronen weiter geleitet wird:

\[ \text{Out}_j(t) = F_j(\text{Output}_j(t)) \]  

(3.7)

3.3.2 Aufbau

Wie beim biologischen Vorbild, hängt das Leistungspotential eines Netzes primär von der Anzahl der Neuronen und deren Verbindungen ab. Abbildung 3.5 zeigt ein 3-Schichten-Modell, das beide Arten von Verbindungstypen beinhaltet. Zum einen existieren Verbindungen zwischen den Neuronen einer Schicht, die häufig eine erregende oder auch hemmende Wirkung haben, zum anderen Verbindungen zwischen Elementen angrenzender Schichten, die meist nur erregender Art sind.

Abbildung 3.5: Netzwerk mit drei Schichten von Neuronen


3.3.3 Trainingsphase

Unter der Trainingsphase versteht man einen iterativen Prozess, der die Anpassung der im Netzwerk vorhandenen Wichtungen als Ziel hat. Dazu werden zuvor ermittelte Muster in die Eingangsneuronen (d.h. an die Neuronen der Eingabeschicht) eingespeist, so dass sich das Ergebnis nach der Abarbeitung an den Neuronen der Ausgabeschicht mit dem erwarteten Resultat abgleichen lässt. Im Idealfall entspricht die Belegung der
Ausgangsneuronen dem gewünschten Vektor. Trifft dies nicht zu, so erfolgt eine Modifizierung der Wichtungen mit Hilfe einer geeigneten Methode wie beispielsweise dem Backpropagation-Algorithmus.

3.4 Reguläre Ausdrücke

Reguläre Ausdrücke stellen ein mächtiges, flexibles und effizientes Hilfsmittel zum Durchsuchen und zur Manipulation von Texten dar. Im Folgenden mit Regex abgekürzte Reguläre Ausdrücke basieren auf Endlichen Automaten, die einen Text nach speziellen Mustern durchsuchen und innerhalb der meisten Programmiersprachen verfügbar sind.

3.4.1 Arten von Regex-Maschinen

Regex-Maschinen lassen sich in zwei Arten unterteilen:


**NFA - Non-Deterministic Finite Automat** Im Gegensatz zu einem textgetriebenen DFA ist ein NFA mustergetrieben. Dies hat zur Folge, dass die Art der Formulierung eines Musters unmittelbaren Einfluss auf die Verarbeitungsgeschwindigkeit und in bestimmten Fällen sogar auf die Art der Übereinstimmung hat. Trotz dieses und anderer Nachteile sprechen für den Einsatz eines klassischen NFA die Fähigkeiten Text durch geklammerte Unterausdrücke einzufangen, die Möglichkeit Lookaround-Konstrukte, Nicht-gierige Quantoren und geordnete Alternationen zu formulieren sowie die Unterstützung von Possessiven Quantoren und Atomaren Gruppen.

Da lediglich die klassische NFA von allen gängigen Entwicklungsumgebungen umgesetzt wird, konzentrieren sich die folgenden Erläuterungen auf die Eigenschaften dieses Automatentyps.

4 Auf die Schilderung spezieller Ausprägungen wie dem POSIX-NFA wird an dieser Stelle verzichtet, da es nicht zum Verständnis der Arbeit beiträgt.
3.4.2 Zeichenklassen


Zu den wichtigsten Metazeichen zählen:

**Punkt ( . )** Der Punkt steht prinzipiell für ein beliebiges Zeichen. Zu beachten ist jedoch, das Newline-Zeichen in Abhängigkeit von der verwendeten Implementation und deren Konfiguration nicht durch den Punkt repräsentiert werden.

**Zirkumflex ( ˆ )** Der Zirkumflex kennzeichnet die Position des Anfangs einer Zeile.

**Dollarzeichen ( $ )** Das Dollarzeichen kennzeichnet die Position des Endes einer Zeile.

**Eckige Klammern ([ . . . ])** Die innerhalb der eckigen Klammern angegebenen Zeichen sind für dieses Muster zulässig.

**Eckige Klammern mit Zirkumex ([ ˆ . . . ])** Folgt auf die öffnende eckige Klammer ein Zirkumflex, so sind die enthaltenen Zeichen für dieses Muster **nicht** zulässig.

3.4.3 Gruppierungen und Alternationen


```java
/*
 * Gruppierung durch runde Klammern
*/
"abcdef".matches("ab(cd)ef"); // true
"abcdcf".matches("ab(cd)ef"); // false

/*
 * Alternation durch das Pipe-Zeichen
*/
"abcdef".matches("ab(cd|de)ef"); // true
"abcdcf".matches("ab(cd|de)ef"); // true
```

Listing 3.2: Nutzung von Alternationen und Gruppen in regulären Ausdrücken (Java)
3.4 Reguläre Ausdrücke

Dabei ist zu beachten, dass diese Alternation von einem NFA unter Berücksichtigung der Reihenfolge verarbeitet wird und somit als geordnet angesehen werden muss. Eine Alternation ist in diesem Fall weder *gierig* noch *genügsam*. Durchsucht man den String

"Den Zuckerhut besteigen"

nach dem Muster

"(Zucker|Zu|Zuckerhut)"


3.4.4 Quantoren

Die Nutzung der in Abschnitt 3.4.2 vorgestellten Zeichenklassen ist häufig nur in Kombination mit so genannten Quantoren sinnvoll. Dabei dienen Quantoren der Angabe von Kardinalitäten um die Häufigkeit des zulässigen Auftretens von Elementen des Musters festzulegen. Listing 3.3 zeigt simple Beispiele für den Einsatz der verfügbaren Quantoren.

```java
/* Fragezeichen: kein oder ein Vorkommen */
"abc".matches("a?.") ; // true
"bcd".matches("a?.") ; // true

/* Stern: kein Vorkommen und beliebig häufig */
"abc".matches("a*.") ; // true
"bcd".matches("a*.") ; // true

/* Plus: mindestens 1 Vorkommen und beliebig häufig */
"abc".matches("a+.") ; // true
"bcd".matches("a+.") ; // false

/* Intervall: mindestens 1 Vorkommen, maximal 2 Vorkommen */
"abc".matches("a{1,2}.") ; // true
"aaabc".matches("a{1,2}.") ; // false
```

Listing 3.3: Nutzung von Quantoren in regulären Ausdrücken (Java)

Die hier angeführten Quantoren werden als *gierig* bezeichnet, da der Automat die maximale Anzahl von Zeichen zu konsumieren versucht. Betrachtet man beispielsweise eine Zeichenklasse, der ein Quantor in Form eines Intervals zwischen N und M

5Diese beiden Begriffe werden im Abschnitt 3.4.4 kurz erläutert
(N, M ∈ N₀, N ≤ M) zugeordnet ist, so wird der Automat Zeichen konsumieren bis entweder keine Übereinstimmung mit der angegebenen Zeichenklasse verfügbar ist oder das Maximum (in diesem Fall M) erreicht wurde. Schlägt ein darauf folgender Unter- ausdruck fehl, so werden dem gierigen Quantor bis zum vollständigen Fehlschlag alle Zeichen bis zu seiner Untergrenze (hier N) entzogen. Im Vergleich dazu verhalten sich nicht-gierige bzw. genügsame Quantoren absolut gegensätzlich und geben die Kontrolle nach der Verarbeitung von N Zeichen an den folgenden Teilausdruck weiter. Sollte dieser folgende Teilausdruck fehlschlagen, so wird der genügsame Teilausdruck mittels Backtracking bis zum Erreichen von maximal M Zeichen zum Konsum gezwungen.

3.4.5 Backtracking


3.4.6 Possessive Quantoren und atomare Gruppen


- **Kein Effekt**: Verläuft die Suche nach einem Treffer erfolgreich, bevor auf gespeicherte Zustände zugegriffen werden muss, die durch den Einsatz von possessiven Quantoren oder atomaren Klamern verworfen wurden, so besteht kein Unterschied zu einem Muster ohne diese Mechanismen.

- **Kein Treffer**: Ist für einen Treffer ein Backtracking zu verworfenen Zuständen notwendig, so ermittelt der Automat im Gegensatz zur üblichen Vorgehensweise keinen Treffer.
• **Ein anderer Treffer:** Abhängig von der Situation, ist alternativ zum Backtracking eventuell auch auf andere Weise ein Treffer möglich.

• **Schneller Fehlschlag:** Eine häufig angestrebte Zielsetzung ist ein früher Fehlschlag, sofern auch ohne atomare Klammern oder possessive Quantoren kein Treffer möglich ist. In diesem Fall wird die Analyse von irrelevanten Permutationen verhindert.

Der Unterschied zwischen atomaren Klammern und possessiven Quantoren ist die Art und Weise, wie der Zugriff auf gespeicherte Zustände verhindert wird. So hat der Einsatz von atomaren Klammern das Verwerfen von Zuständen im Anschluss an die Abarbeitung zur Folge. Possessive Quantoren unterbinden hingegen schon im Vorfeld die Speicherung.
4 Konzeption

Aufbauend auf den Ergebnissen der Anforderungsanalyse bezüglich der vorliegenden Rahmenbedingungen, der geforderten Funktionalität und den notwendigen Vorraussetzungen, beschreibt der Entwurf unter Anderem anhand einer Gegenüberstellung die grundlegenden Ansätze zur Lösung des Problems.

4.1 Anforderungsanalyse

4.1.1 Produktionsphasen


**Gießen des Wickelkörpers** Zu Beginn der Herstellung eines Sensors wird ein Ferrit-Kern mit Epoxidharz ummantelt.

**Drehen des Wicklungskopfes** Eine Nut wird zur Aufnahme der Spule eingebracht.

**Wickeln der Spule** Es wird eine Wicklung aus feinem Kupferdraht in die zuvor gedrehte Nut aufgebracht.

**Anschluss des Kabels** Die Enden der Wicklung werden mit den Adern des Anschlusskabels verlötet.

**Versiegelung des Wicklungskopfes** Der Sensorkopf wird mit der Anschlussstelle des Kabels unter Unterdruck mit Epoxidharz vergossen. Um eine Haftung des Harzes am Teflon-beschichteten Kabel zu gewährleisten, wird dieses zuvor im Plasma-Ofen aktiviert.

20
**4.1 Anforderungsanalyse**

**Schleifen des Sensorkopfes** Das Kopfende des Sensors wird durch Schleifen verkürzt, bis die parallel dazu gemessene Schwingkreisspannung oder Güte einen im Vorfeld definierten Wert erreicht. Zusätzlich werden zwei Spannungswerte im Abstand von 1,23 und 2,23 mm zur Schleifscheibe erfasst und zur Beurteilung der elektrischen Eigenschaften des Sensors genutzt. In Form von Stichproben werden diese Messungen am Wobbel-Messplatz verifiziert.

**Montage** In diesem Produktionsschritt erfolgt eine kundenspezifische Anpassung und die Kalibrierung des Konverters.


Die hohe Anzahl an Messungen, die in Tabelle 4.1 aufgeführt sind, ist auf die Tatsache zurückzuführen, dass es zu Beginn des Projekts nicht möglich war, eine präzisere Aussage über mögliche Störfaktoren zu treffen. Somit gilt es ein großes Spektrum an Informationen bereit zu stellen und im Anschluss den Überprüfungsaufwand durch ein Ausschlussverfahren von irrelevanten Messgrößen zu reduzieren.

**4.1.2 Ist-Zustand**

Nach Abschluss eines Produktionsschritts erfolgt die Erfassung der Messwerte an einem Arbeitsplatzrechner über eine eigens dafür entwickelte Software. Diese Software speichert die zuvor erfassten Daten in einer Microsoft Access Datenbank, die sich auf einem zentralen Fileserver befindet. Dabei werden einige Informationen wie beispielsweise die Umgebungstemperatur manuell, andere in Kombination mit Messgeräten automatisiert ermittelt.

Die Prüfdaten des Wobbel-Messplatzes werden in eine separate Access Datenbank geschrieben, da die Entwicklung der dafür zuständigen Software von anderen Teilen der Produktionsüberwachung losgelöst durchgeführt wurde und dieser Arbeitsschritt lediglich für Stichproben erfolgt.
<table>
<thead>
<tr>
<th>Nr.</th>
<th>Bezeichnung</th>
<th>Messungen</th>
<th>Abbildung</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Gießen des Wickelkörpers</td>
<td>keine</td>
<td><img src="image1" alt="Image" /></td>
</tr>
<tr>
<td>2</td>
<td>Drehen des Wicklungskopfes</td>
<td>keine</td>
<td><img src="image2" alt="Image" /></td>
</tr>
<tr>
<td>3</td>
<td>Wickeln der Spule</td>
<td>Umgebungstemperatur</td>
<td><img src="image3" alt="Image" /></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Kernlänge</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Induktivität des Sensorkopfes</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Güte des Sensorkopfes</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>Anschluss des Kabels</td>
<td>Umgebungstemperatur</td>
<td><img src="image4" alt="Image" /></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Induktivität des Sensorkopfes</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Güte des Sensorkopfes</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>Versiegelung des Wicklungskopfes</td>
<td>Umgebungstemperatur</td>
<td><img src="image5" alt="Image" /></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Induktivität des Sensors</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Güte des Sensors</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>Schleifen des Sensorkopfes</td>
<td>Schleifparameter (Schleif-, Mittel- und Endspannung)</td>
<td><img src="image6" alt="Image" /></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Sensorkopflänge</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Durchmesser des Sensorkopfes</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Umgebungstemperatur</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>Montage</td>
<td>Umgebungstemperatur</td>
<td><img src="image7" alt="Image" /></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Induktivität des Sensors</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Güte des Sensors</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Güte des Sensors (bedämpft)</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Sensorkopflänge</td>
<td></td>
</tr>
</tbody>
</table>

Tabelle 4.1: Chronologische Auflistung der Produktionsschritte und geplanter Messungen
4.1.3 Funktionalität


4.2 Entwurf

Alle folgenden Überlegungen gehen von der These aus, dass es sich nicht um einen entscheidenden Produktionsfehler handelt, der für das hohe Maß an fehlerhaften Sensoren verantwortlich ist. Durch vorangegangene Untersuchungen kann man viel mehr davon ausgehen, dass die Ursache für das Problem in einer nahezu beliebigen Kombination verschiedener Faktoren zu sehen ist. Erschwerend kommt hinzu, dass sich die Einflüsse einiger Arbeitsschritte auf die Produkteigenschaften kaum in physikalische Modelle übertragen lassen, da selbst geringe Abweichungen weitreichende Konsequenzen haben können.


Die anstehende Umstellung der Produktionsreihe erfordert eine flexible Struktur, die eine Anpassung an sich ändernde Rahmenbedingungen möglich macht. Die Art und Anzahl der Datenquellen, der durchzuführenden Messungen und Arbeitsschritte muss aus diesem Grund als variabel angesehen werden.

Um den Erfolg des Projekts nicht von der Wahl der im Folgenden erläuterten Analyseansätze abhängen zu lassen, besteht ein besonderer Anspruch an die Implementation in der Austauschbarkeit des jeweiligen Mechanismus. Ziel ist es, die zur Analyse ge-
nutzte Komponente in jeder Phase des Softwar-Lebenszyklus austauschen oder mit anderen kombinieren zu können. Entscheidend dafür ist die Modellierung einer Analysemaschine auf Basis von Gemeinsamkeiten beider Ansätze. So unterscheiden sich zwar Auswertungszeitpunkt\textsuperscript{1} und die Ergebnisdarstellung, andererseits kann eine einheitliche Modellierung von Messdaten und eine gemeinsame Schnittstelle für Analyseergebnisse nach dem Besucher-Muster\textsuperscript{2} zur Austauschbarkeit beitragen.

Als Metapher dient eine "Black Box", deren interne Arbeitsweise nicht von Bedeutung und somit austauschbar ist und über möglichst einheitliche Ein- und Ausgangskanäle verfügt. Der Preis für die hierdurch gewonnene Flexibilität ist jedoch die steigende Komplexität der Client-Komponente: Die Verarbeitung unterschiedlicher Ergebnisarten setzt eine differenzierte Implementation vorraus.

### 4.2.1 Anwendungsfälle

![Abbildung 4.1: Anwendungsfalldiagramm für die Sensorreihe PR6423](image)

\textsuperscript{1}Die Inferenzmaschine liefert Ergebnisse im Fall erfüllter Regelbedingungen, das künstliche neuronale Netz erst nach vollständiger Belegung sämtlicher Eingangsneuronen.

\textsuperscript{2}Das auch als *Visitor-Pattern* bekannte Entwurfsmuster wird im Verlauf der Arbeit näher erklärt.
4.2.2 Voraussetzungen

Um den zuvor beschriebenen Anforderungen an das Expertensystem gerecht zu werden, ergeben sich einige grundlegende Voraussetzungen an die bestehende oder zu schaffende Umgebung:


Kommunikation zwischen Erfassungspunkten Ausgehend von der These verteilter Ursachen von fehlerhaften Sensoren, ist die Berücksichtigung aller Messdaten für die Analyse notwendig. Erfolgt die Datenerfassung an verschiedenen Punkten innerhalb der Produktion, so ist eine Bündelung der Informationen unumgänglich.


Erweiterung der Datenerfassung Zusätzlich zu den bislang erfassten Produktionsdaten, ist eine kontinuierliche Anpassung der Art und des Umfangs notwendig, um neuen Erkenntnissen und ungeklärten Problemen gerecht zu werden.

Bereitschaft zur aktiven Weiterentwicklung In Abhängigkeit vom gewählten Analyseansatz und dessen adaptiven Eigenschaften besteht unter Umständen die Notwendigkeit einer aktiven Weiterentwicklung.

4.2.3 Komponentenverteilung

4.2 Entwurf

Abbildung 4.2: Komponentenverteilung des Expertensystems


4.3 Neuronale Netze


4.4 Inferenzmaschine


Der Entwurf einer für diesen Einsatzbereich geeigneten Struktur sieht eine Unterteilung in zwei Verarbeitungsphasen vor:

4.4.1 Analyse

Die so genannte Analysephase umfasst Regeln zur Identifikation von Fehlmessungen und zur Bestimmung erwarteter Produkteigenschaften. Diese Regeln sind einem "RuleFlow" zugeordnet, der eine chronologische Ausführungsreihenfolge nach Gruppenzugehörigkeit ermöglicht, so dass keine Festlegung der Produkteigenschaften erfolgt, sofern es sich um einen Messfehler handelt.
4.4.2 Reflektion


Abbildung 4.3: Schematische Darstellung der Reflektionsphase


4.5 Gegenüberstellung der Lösungsansätze

Die im weiteren Verlauf näher beschriebenen Ansätze bestehen zum einen in der Anwendung künstlicher neuronaler Netze und zum anderen im Einsatz von Inferenzmaschinen.

Wie im Kapitel *Einführung* erläutert, zeichnet sich ein neuronales Netz insbesondere
4.5 Gegenüberstellung der Lösungsansätze


In Hinblick auf die zuvor genannten Anforderungen, stellt eine Hybridlösung eine vielversprechende Alternative zur exklusiven Wahl eines Analyseansatzes dar. So nutzte beispielsweise die Firma Wincor-Nixdorf eine Kombination aus neuronalem Netz und Inferenzmaschine, um die von ihnen vertriebenen Geldautomaten auf Fehler zu überwachen und Servicetechnikern Hinweise auf mögliche Störungursachen zu liefern. Durch die Verarbeitung akustischer Signale diente das adaptive Netz dabei als zusätzliche Datenquelle für die Regelmaschine[12].
5 Implementation


5.1 Projektmanagement


Das quelloene und ebenfalls von der Apache Software Foundation stammende Projekt Maven widmet sich den zuvor angeführten Hindernissen. Es kapselt einerseits die Komplexität möglicherweise zyklischer Abhängigkeitsstrukturen und der Erstellungsrei-
5.2 Refactoring der Messdatenspeicherung

Wie im Abschnitt zum Entwurf beschrieben, erfolgt die Ersterfassung von Messungsdaten durch eine von Mitarbeitern der Abteilung MMS entwickelte Software, deren Entwicklungsumgebung die direkte Kommunikation mit Diagnosegeräten unterstützt. Die Modellierung der Access-Datenbank, die als Zwischenspeicher und Schnittstelle zum Expertensystem fungiert, fand dabei anfangs ohne Berücksichtigung von Normalisierungsregeln statt. Dieser Umstand machte eine Neukonzeption der Tabellenstruktur notwendig, wie sie in Abbildung 5.1 zu sehen ist. In Bezug auf die Datenbank, die der Speicherung der am Wöbbelnessplatz ermittelten Informationen dient, war eine derartige Anpas-

---

1 Das Akronym steht für Project Object Model
2 Hierzu zählen Versionsnummer, Name, Zielformat, etc.
3 Ein Repository ist eine Art Lager für digitale Objekte
sung nicht möglich, da sich die Software zur Datenerfassung bereits im Produktiveinsatz befand und eine Migration der Daten zu aufwändig wäre.

5.3 Darstellung von Fakten


Diese Tatsache trug in der Modellierungsphase zu der Entscheidung bei, den Messwert (und somit auch dessen Datentyp) als Bestandteil konkreter Implementierungen zu formulieren. Wie im Kapitel Entwurf bereits erwähnt, resultiert daraus für eine Fakten konsumierende Komponente des Softwaresystems ein zusätzlicher Aufwand, um zwischen den jeweiligen Ausprägungen zu unterscheiden. Geht man von der Notwendigkeit mehrerer Konsumenten aus, so ergibt sich zudem die Gefahr einer inkonsistenten Implementation, sofern die Differenzierung nach Messungsart durch eine Alternation erfolgt. In diesem

Beispiele hierfür sind Zeichenketten oder Byte-Arrays

4
5.3 Darstellung von Fakten

Abbildung 5.2: Klassendiagramm zur Kommunikation zwischen Fakten und Konsumenten

Fall wäre zu gewährleisten, dass jeder Konsument über eine vollständige Liste der vorkommenden Faktenarten verfügt, um auf diesem Weg mittels Typumwandlung auf den Messwert zugreifen zu können. Auch wenn das Besucher-Muster nicht den Nachteil des erhöhten Implementationsaufwands ausgleicht, so verhindert es in diesem Kontext die Inkonstanz des Unterscheidungsmechanismus: Ausgehend von der Überlegung, dass ein Faktenobjekt die Information zur Messungsart enthält, die es repräsentiert, nutzt es diesen Umstand durch eine zusätzliche Methode, die der Empfänger der Messdaten am Messungsobjekt aufrufen kann.

Als Parameter erwartet die Funktion eine Instanz der allgemeinen Konsumentenschmittstelle FactConsumer, die ihrerseits die spezifischen Empfängermethoden für sämtliche Messungsarten erbt. Diese Vorgehensweise stellt sicher, dass nur die Konsumenten in der Lage sind, die Methode `specifyFactType` aufzurufen, die entweder die Schnittstelle FactConsumer selbst implementieren oder über Objekte verfügen, an die jene Verarbeitung delegiert werden kann. Die Anforderung an das Verhalten der Konsumenten wird also durch die erzwungene Implementation der allgemeinen Konsumentenschmittstelle vorgeschrieben und kann auf diese Weise zentral beeinflusst werden.

Wird im Rahmen der Produktionsdatenerfassung die Konzeption einer neuen Messungsart notwendig, so spiegelt sich die Erweiterung in der Programmierung von zwei korrespondierenden Komponenten wieder: Zum einen muss eine Schnittstelle entworfen...
werden, die von Fact erbt und sie in Abhängigkeit vom entsprechenden Datentyp der Messung um Getter- und Setter-Methode erweitert. Andererseits wird eine entsprechende Konsumentenschmittstelle benötigt, die für den Empfang des neuen Messungstyps geeignet ist und innerhalb der Methode `specifyFactType` aufgerufen werden kann. Abschließend erfolgt eine Ergänzung der allgemeinen Konsumentenschmittstelle durch die hinzugekommene Vererbungsbeziehung. Befindet sich die Software bereits im produktiven Einsatz, so hat dies zur Folge, dass sich die aktualisierte Version erst dann kompilieren lässt, wenn alle Konsumenten in angemessener Form auf die zusätzliche Messgröße reagieren.

Die Erweiterung des Modells um die Schmittstelle `Specifiable` hat ihren Ursprung in der Notwendigkeit auch dann zwischen Fakten unterscheiden zu können, wenn Daten der gleichen Messungsart und des selben Produktionsschritts eine abweichende semantische Bedeutung haben. Dies gilt beispielsweise für Spannungsmessungen, die beim Schleifen des Sensorkopfes erhoben werden und sich abgesehen von ihrem Wert auch in ihrer Aussage unterscheide.

5.4 Darstellung der Analyseergebnisse


---

5Mindestvoraussetzung für einen erfolgreichen Kompiliervorgang ist eine nicht ausimplementierte Deklaration der Empfängermethoden
6vgl. Abbildung 3.2
7Die Messung einiger Größen lässt Rückschlüsse auf falsche Windungszahlen, defekte Lötzstellen und Ähnliches zu
5.4 Darstellung der Analyseergebnisse

Abbildung 5.3: Sequenzdiagramm der Kooperation zwischen Result und ResultListener

8Die Bestimmung der Abweichung kann sowohl absolut durch einen fixen Zahlenwert oder relativ zu einer gemessenen Größe erfolgen

9Ein Diagnosegerät zur Messung von Widerstand (R), Kapazität (C) und Induktivität (L)
Nutzung der Messwerte und zudem den Verlust der Beurteilungsgrundlage für die weitere Analyse. Um eine solche Fehlmessung zu protokollieren, dient das als Präzisionsfaktor bezeichnete Attribut. Da das Expertenwissen kaum als lückenlos und fehlerfrei angesehen werden kann, gilt es zusätzlich einen Indikator bereit zu stellen, der insbesondere im Rahmen der Inbetriebnahme einen Hinweis auf Ungenauigkeiten in der Wissensbasis darstellt. Der als Fehlerfaktor bezeichnete Zahlenwert wird im Aktionsteil einer Regel erhöht, sofern sich Unterschiede zwischen den Vorgaben des Experten und den tatsächlich gemessenen Informationen ergeben. Dabei ist der entscheidende Unterschied zwischen den Epsilon-Faktoren, die später für die Beurteilung der Funktionstüchtigkeit eines Sensors genutzt werden und dem Fehlerfaktor, der ebenfalls Abweichungen dokumentiert, der, dass Epsilonwerte vorzeichenbehaftet sind und sich somit in ihrer Summe kompensieren können. Der Fehlerfaktor stellt eine absolute und vorzeichenlose Größe dar, die für die Identifikation von Regeln genutzt werden kann, die am wenigsten den realen Gegebenheiten entsprechen. Enthalten die Ergebnisse eines Sensors, der anhand von abschließenden Überprüfungen die vorgegebenen Toleranzen einhält, einen oder mehrere hohe Fehlerfaktoren, so sind die für die höchsten Fehlerfaktoren verantwortlichen Regeln aller Wahrscheinlichkeit nach überarbeitsbedürftig.

5.4.1 Ergebnisse der Inferenzmaschine

Wie im Kapitel Konzeption beschrieben, erfolgt eine Untergliederung der Ergebnisstruktur in die Bereiche Analyse und Reaktion.


10Die hierzu genutzte Methodik wird im Folgenden beschrieben
5.4 Darstellung der Analyseergebnisse

Abbildung 5.4: Hierarchie der Ergebnisstruktur und Fallbeispiel für Fehlmessungen

5.4 Darstellung der Analyseergebnisse

Abbildung 5.5: Klassendiagramm der Ergebnisstruktur der Inferenzmaschine

tor) aufsummiert. Diese Vorgehensweise erweist sich auch in Bezug auf die Darstellung von textuellen Hinweisen als nützlich. So erfolgt beispielsweise eine farbliche Hervorhebung der Hinweise durch die graphische Benutzeroberfläche in Kombination mit dem Präzisionsfaktor, um die Aufmerksamkeit des Mitarbeiters auf mögliche Fehlmessungen zu erhöhen. Dabei bezieht sich die Hervorhebung lediglich auf die Hinweise, die dem Regelergebnis mit der potentiellen Fehlmessung zugeordnet sind. Für eine ganzheitliche und einheitliche Bewertung dienen andererseits die kumulierten Angaben des Produktresultates, so dass die Berechnung nicht explizit durch Komponenten wie die graphische Benutzeroberfläche erfolgen muss.

5.4 Darstellung der Analyseergebnisse


5.4.2 Ergebnisse des neuronalen Netzes

5.5 Ergebnisauswertung


Bei der Entwicklung von JAnalysis fiel die Entscheidung auf eine Unterteilung in drei Prioritätsstufen und zwei Aspekte - die Beurteilung eines Sensors erfolgt demnach mit einem Fokus auf die Plausibilität der Messungsdaten und die diagnostizierte Qualität des Produkts, wobei dem jeweiligen Aspekt eine Farbe zugeordnet ist. Dieser Ansatz äußert sich für den Anwender in einer leichteren Bedienung, da er die Situation aus Sicht des Expertensystems schneller erfassen kann.


Daraus folgt, dass in Abhängigkeit von der Perspektive einer Abteilung ein unterschiedlicher Anspruch an die Auswertung existiert. Um dieser Tatsache gerecht zu werden, entfällt die zentrale Beurteilung der Qualität eines Produkts und der Messungsplausibilität und wird an die Client-Komponente delegiert. Als Kern der Bewertungsmethode

5.6 Broker


5.6.1 Komponenten


\textsuperscript{12}Die Abkürzung steht für General Responsibility Assignment Software Patterns - eine Sammlung von Entwurfsmustern zur Festlegung der Zuständigkeiten
Gatherer / Provider


FilterChain

Oberflächlich mag eine Filterung von zuvor aufwändig erzeugten oder empfangenen Daten wenig sinnvoll erscheinen. Ausschlag gebend für die Konzeption eines solchen Mechanismus sind aber beispielsweise:

- Das parallele Auslesen einer Datenbank von verschiedenen Klienten
- Die parallele Nutzung verschiedener Datenquellen, deren Inhalte sich überschneiden
- Die künstliche Beschränkung der Ergebnismenge nach Produktionsschritt oder anderen Kriterien

Die beiden zuerst genannten Gründe dienen der Vermeidung von redundanten Informationen: Im Zuge einer Weiterentwicklung ist eine Ergänzung um native Schnittstellen zur direkten Kommunikation mit Diagnosegeräten denkbar und zur Effizienzsteigerung auch erwünscht. Da dies eine doppelte Erfassung von Messung zur Folge haben kann, bedarf es einer konfigurierbaren Möglichkeit um Informationen gezielt einzuschränken. Aus der Sicht des Anwenders im Produktionsbereich ist eine Filterung der eintreffenden Analyseergebnisse unerlässlich, da er sich ohne diese Option mit einer großen Menge


\[43\]
an irrelevanten Informationen konfrontiert sehen würde. Betrachtet man beispielsweise einen Mitarbeiter, der für den Produktionsschritt "Schleifen" zuständig ist, so wäre eine Überflutung mit Analyseergebnissen aus anderen Fertigungsschritten nicht nur unnötig, sondern auch kontraproduktiv.


Ein Fallbeispiel:
Gegeben sei eine Instanz der FilterChain `fc`, die zwei Filterobjekte enthält. Der erste Filter `f1` ist ein Redundanzfilter, der intern eine Historie der letzten n Fakten pflegt. Der zweite Filter `f2` entfernt alle Messungsdaten, die sich nicht auf den Arbeitsschritt "Wickeln der Spule" beziehen.

Die Abfolge der Methodenaufrufe beim Eintreffen einer neuen (zuvor nicht ermittelten) Messung `m1` aus der Produktionsphase "Wickeln der Spule":

1. Broker ruft `fc.doFilter(m1)`
2. `fc` ruft `f1.doFilter()` mit den Parametern `m1` und einer Referenz auf sich selbst
3. `f1` stellt fest, dass es sich um neue Informationen handelt und ruft `fc.doFilter(m1)`
4. `fc` ermittelt den nächsten Filter `f2` und ruft wie im ersten Durchlauf `f2.doFilter()`
5. Auch `f2` stellt die Gültigkeit von `m1` fest und ruft `fc.doFilter(m1)`
6. Da keine weiteren Filter existieren, delegiert die FilterChain den Fakt an den Broker zur weiteren Verarbeitung

Handelt es sich dagegen um eine Messung `m2`, die entweder innerhalb der Historie von `f1` enthalten oder aber einem anderen Produktionsschritt zugeordnet ist, so äußert sich diese "Ungültigkeit" des Fakts in Schritt 3 bzw. 5. Der Aufruf von `fc.doFilter(m2)` wird in diesem Fall durch `fc.factRejected()` substituiert und die Weiterleitung des Objekts an weitere Filter und den Broker entfällt.

### 5.6.2 Synchronisation

Eine besondere Eigenschaft des Brokers ist die Aggregation einer beliebigen Anzahl von Modulen zur Lösung einer Teilaufgabe. Je nach Anwendungsgebiet und Situation ist es so möglich, auf einige Komponenten komplett zu verzichten oder das Ziel mit Hilfe mehrerer Module zu erreichen. Wie eine Teilkomponente ihre Aufgabe erfüllt, muss für den Broker irrelevant sein. Abbildung 5.8 zeigt ein Beispiel für die Konfiguration eines *FactBrokers*, der sämtliche Messungsdaten ausliest, eine Filterung durchführt und die verbleibenden Daten abschließend veröffentlicht: Da die Fakten verteilt in zwei unterschiedlichen Datenbanken abgelegt sind, sind zwei *Gatherer* notwendig, die ihre Arbeit voneinander losgelöst verrichten. Die dabei erzeugten Daten durchlaufen die *FilterChain*,

![Abbildung 5.8: Beispiel für die Kooperation der Broker-Komponenten](image-url)
5.7 AnalysisMachine

Die Schnittstelle AnalysisMachine beschreibt das Verhalten des zur Analyse genutzten Moduls und ist aus diesem Grund für die Architektur des gesamten Expertensystems von entscheidender Bedeutung. Wie im Kapitel Konzeption erläutert, ist die Austauschbarkeit des Auswertungsansatzes ein grundlegender Anspruch an die zu entwickelnde Software. Um diese Zielsetzung zu erreichen, dient dieses Interface als Abstraktionsschicht, die einen Wechsel des Analyseansatzes zu einem beliebigen Zeitpunkt ermöglicht.

\[\text{Es handelt sich dabei um klassische Probleme bei nebenläufiger oder paralleler Ausführung (vgl. [13])}\]
Abbildung 5.9: Kooperation der Broker zwischen Client und Server

### 5.8 Server- & ClientFacade


Für die Persistenz der Konfiguration schreiben die Schnittstellen `ServerFacade` und `ClientFacade` die Methoden `storeConfig` und `loadConfig` vor. Das Prinzip der Implementation wird im Folgenden anhand der Klasse `ConfigClientFacade` erläutert:

Nach der Initialisierung der Konfigurationsverzeichnisse, die innerhalb einer zentralen `*.properties`-Datei festgelegt sind, existiert nach dem Speichervorgang, der sich über alle zum Zeitpunkt aktiven Module erstreckt, eine XML-Datei pro Broker-Komponente. Da es sich ausschließlich um Objekte handelt, die der Java Beans-Konvention entsprechen, ist eine direkte Serialisierung unter Verwendung des `XMLEncoder` möglich. Der Dateiname einer Einstellungsdatei setzt sich dabei aus drei Bestandteilen zusammen:

- **Komponententyp** Die Art des Broker-Moduls. Beispiele sind `FactFilter`, `ResultPublisher` oder `FactGatherer`.

Die Notwendigkeit ergibt sich hauptsächlich aus der generischen Konfigurationsschnittstelle, die im Abschnitt über Konfigurationsdialoge näher erläutert wird.
Name der implementierenden Klasse  Der vollständige Name der Klasse inklusive der 
Package-Hierarchie.

Rückgabewert der hashCode-Funktion  Eine Zahl vom Typ long, die eine Unterschei-
dung von Objekten gleicher Art mit unterschiedlichen Zuständen ermöglicht.

Der Aufbau des Dateinamens ist für spätere Anpassungen durch Änderungen an der
Konstanten CONFIG_FILE_NAME_FORMAT konfigurierbar. Sie setzt sich aus den
konstanten Zeichenketten zusammen, die in Listing [5.1] aufgeführt sind.

```java
protected final String CONFIG_FILE_SEPARATOR = "_";
protected final String CONFIG_FILE_EXTENSION = ".xml";
protected final String CONFIG_COMPONENT_TYPE = "[TYPE]";
protected final String CONFIG_COMPONENT_TYPE_WILDCARD = "[a-zA-Z0-9]++";
protected final String CONFIG_COMPONENT_IMPL = "[IMPL]";
protected final String CONFIG_COMPONENT_IMPL_WILDCARD = "([a-zA-Z0-9\.]+)++";
protected final String CONFIG_COMPONENT_HASH = "[HASH]";
protected final String CONFIG_COMPONENT_HASH_WILDCARD = "[-0-9]++";
protected final String CONFIG_FILE_NAME_FORMAT = CONFIG_COMPONENT_TYPE +
      CONFIG_FILE_SEPARATOR + CONFIG_COMPONENT_IMPL + CONFIG_FILE_SEPARATOR +
      CONFIG_COMPONENT_HASH + CONFIG_FILE_EXTENSION;
```

Listing 5.1: Konstanten zur Festlegung des Konfigurationsdateinamens

Bei genauerer Betrachtung fällt die Kombination aus Platzhalter und regulärem Aus-
druck auf: Die Struktur des Dateinamens wird durch Platzhalter festgelegt, die bei-
spielsweise durch Markierungen wie "[TYPE]" oder "[HASH]" angegeben, wo sich das
durch Korrespondierende reguläre Ausdruck dient hingegen der Prüfung und Selektion
eines Bestandteils des Dateinamens beim Auslesen des Verzeichnisinhalts. Die Liste al-
der Dateien, die sich im Konfigurationsverzeichnis befinden, kann somit im Zuge des
Einlesens der Konfiguration durch einen FileFilter, künstlich eingeschränkt werden. Die
überschriebene Methode accept überprüft mit Hilfe des zuvor genannten regulären Aus-
drucks die Gültigkeit des Dateinamens und filtert auf diese Weise alle Einträge, die
die der Konvention entsprechen. Es existieren zwei unterschiedliche Ausprägungen des
Dateifilters - zum einen der AllConfigFileFilter, der durch eine geordnete Alternation
sämtliche Modultypen zulässt und andererseits ein BrokerConfigFileFilter, dessen Kon-
struktor eine Referenz auf die Modulklassse erwartet, um die resultierende Dateiquote.
on einen Komponententyp einzuschränken.

Die Schnittstellen ConsumerFactPublisher und ListenerResultPublisher dienen als Bin-
deglieder zwischen den Brokern und systeminternen Beobachtern. Sie werden dem jewei-

16 Das doppelte Pluszeichen am Ende des Ausdrucks ist ein Possessiver Quantor und kein Druckfehler.
Sinn und Funktion sind im Kapitel Einführung beschrieben
5.9 GUI des Clients

Dieser Abschnitt beginnt mit einem allgemeinen Überblick über die Form der Informationsaufbereitung und vermittelt im Anschluss das Hintergrundwissen bezüglich interner Strukturen und Kommunikationswege.

Die Benutzerschnittstelle des Clients wurde mit der Grafikbibliothek Swing erstellt und ist vollständig lokalisiert. Sie besteht aus einem Hauptfenster mit Menüleiste, das durch Reiter in die Bereiche Allgemein, Messungen, Ergebnisse und Reaktion unterteilt ist. Da der Reiter Optionen lediglich Voreinstellungen auflistet und keinen relevanten Beitrag zum Verständnis des Systems leistet, wird auf eine Schilderung an dieser Stelle verzichtet.

5.9.1 Allgemein

Der Reiter Allgemein umfasst innerhalb einer JScrollPane ein Textfeld vom Typ JTextArea, das Status- und Debuginformationen auflistet. Hierzu werden Standardausgabe- und Fehlerstrom während des Programmsstarts in einen internen PrintStream umgeleitet und

17Der Begriff Lokalisierung beschreibt die Anpassbarkeit der Beschreibungselemente, um eine Umstellung der Sprache ohne erneutes Komplieren zu ermöglichen. Die Lokalisierbarkeit bezieht sich neben den unmittelbaren Bestandteilen der GUI auch auf Objekte, deren toString-Methode zur Anzeige genutzt wird.
mittels SwingUtilities.invokeLater() synchronisiert auf der Textkomponente ausgegeben. Abbildung 5.10 zeigt diese Ausgabe nach dem Start des Programms.

Abbildung 5.10: Anzeige des Clients nach Start der Anwendung

5.9.2 Messungen

Um dem Benutzer einen geordneten Überblick über die Fakten zu ermöglichen, dient das in Abbildung 5.11 aufgeführte Registerblatt. Sämtliche Messungsdaten sind in einer Baumstruktur organisiert, die in ihrer ersten Hierarchieebene über die Einträge Alle Messungen, Nach Messungsart, Nach Sensor und Nach Produktionsschritt verfügt. Diese
5.9 GUI des Clients


5.9.3 Ergebnisse


Abbildung 5.12: Anzeige von Analyseergebnissen der Inferenzmaschine
5.9 GUI des Clients

Aus Sicht des Programmierers handelt es sich beim Inhalt des Ergebnis-Desktops um ein Steuerelement namens ResultDetailsPanel, das von JSplitPane erbt und die Schnittstelle DroolsResultListener implementiert. Werden in Zukunft auch andere Ansätze zur Diagnose genutzt, so muss das zuletzt genannte Interface durch ResultListener ersetzt werden, das die Empfängermethoden aller Analysemaschinen umfasst. Demzufolge läge es in der Verantwortung des Steuerelements, die graphische Aufbereitung an unterschiedliche Ergebnisstrukturen anzupassen.18 Ungeachtet dessen enthält das ResultDetailsPanel vier verschiedene Komponenten: Eine JTable, die in eine JScrollPane eingebettet ist, zwei TrafficLight Widgets, einen ResultDetailsTree, der von JTree erbt und ebenfalls die DroolsResultListener Schnittstelle implementiert und zuletzt eine JEditorPane, die der formatierten Anzeige von Hinweisen dient.


18 Eine Möglichkeit besteht im Austausch von Bestandteilen
Klick des Benutzers oder automatisch beim Eintreffen neuer Daten vorgenommen.

### 5.9.4 Reflektion

Der Reiter *Reflektion* besteht aus lediglich einem Steuerelement, das von *JTree* erbt und die Schnittstelle *DroolsResultListener* implementiert.

![Abbildung 5.13: Anzeige von Reflektionsergebnissen](image)

Innerhalb der Initialisierungs methode werden der Baumstruktur zwei Wurzelknoten hinzugefügt, die eine Unterteilung nach Regelname und Reflektionskategorie realisieren. Nach dem Empfang einer Instanz von *ReflectionRuleResult* erfolgt der Aufbau eines Ergebniseintrags für jeden zugeordneten Kategoriezweig und einmal nach dem Namen der Reflektionsregel. Sollte der Name der Regel oder eine assoziierte Kategorie noch keine
Entspricht als Knoten in der Baumstruktur haben, so wird die Erzeugung an die Funktionen `findByRuleName` bzw. `findByCategory` delegiert, die ansonsten für das Auffinden bereits vorhandener Elemente zuständig sind. Der verschachtelte Aufbau einer Ergebnisstruktur trug zu der Entscheidung bei, eine separate Methode für die Erzeugung eines jeden Bestandteils zu formulieren. So basiert ein Regelergebnis aus Sicht der Reflektion aus einer Menge an Produkt- und Regelergebnissen, die eine Liste von Fakten und Hinweisen enthalten. Der Aufruf von `createResultNode` reicht dementsprechend die Generierung von Teilknoten an spezialisierte Funktionen weiter und konzentriert sich auf das Zusammensetzen der zurückgegebenen `DefaultMutableTreeNode`.

### 5.9.5 Konfigurationsdialoge


Ein Ansatz der Implementierung einer Konfigurationsschnittstelle ist die Bündelung von Broker-Modul und Konfigurationseinheit. Der Programmierer ist in diesem Fall sowohl für die fachliche Formulierung der Broker-Komponente als auch für die Gestaltung eines korrespondierenden Dialogs verantwortlich. Zudem muss anhand von zusätzlichen Interfaces sichergestellt werden, dass sich die entsprechenden Einstellungen einheitlich speichern lassen.

Die offensichtlichen Nachteile dieses Verfahrens trugen zu der Entscheidung bei, eine generische Lösung anzustreben. Als Grundlage für eine automatische Erzeugung von Konfigurationen dient einerseits die `Reflection-API` von Java und andererseits die `Java Beans` Konvention:

Um dem Broker zur Laufzeit Module hinzufügen zu können, bedarf es eines offiziellen und parameterlosen Konstruktors sowie standardisierten Methodenpaaren, die einen lesenden und schreibenden Zugriff auf die Attribute des jeweiligen Objekts ermöglichen. Damit ausgestattete Broker-Module lassen sich mit einem Objekt vom Typ `BrokerComponentReflector` manipulieren.

Als Ausgangspunkt für einen solchen Mechanismus wird eine Liste vorhandener Brokerschnittstellen benötigt:

---

19 Hierzu zählt insbesondere der erhöhte Implementationsaufwand
5.9 GUI des Clients

Listing 5.2: Konstante Arrays mit den vollqualifizierten Namen der Brokerinterfaces

Im Folgenden wird eine weitere Liste erzeugt, die sämtliche Klassenpfade des Projekts beinhaltet. Da die Clientanwendung aus mehreren Java Archiven zusammengesetzt sein kann, ist eine Suche in allen Dateien der Anwendung unerlässlich. Hierzu werden erstmals Funktionen der Java Reflection-API genutzt, um die Datei zu ermitteln, in der sich der aufrufende BrokerComponentReflector befindet.


Listing 5.3: Initialisierung der Komponentenliste

```java
protected static final String[] FACT_FILES = {
    "de.epro.mms.janalysis.facts.interfaces.FactGatherer",
    "de.epro.mms.janalysis.facts.interfaces.FactFilter",
    "de.epro.mms.janalysis.facts.interfaces.FactPublisher",
};

protected static final String[] RESULT_FILES = {
    "de.epro.mms.janalysis.results.interfaces.ResultProvider",
    "de.epro.mms.janalysis.results.interfaces.ResultFilter",
    "de.epro.mms.janalysis.results.interfaces.ResultPublisher",
};

protected void initComponentList() {
    String[][] componentTypeFiles = {FACT_FILES, RESULT_FILES};
    classLoop : for (String className : entryList) {
        for(String[] currentComponentType : componentTypeFiles) {
            try {
                Class<?> componentClass = Class.forName(
                    currentComponent);
                Class<?> currentClass = Class.forName(className);
                int modifiers = currentClass.getModifiers();
                if (!componentClass.isAssignableFrom(currentClass)
                    || componentClass == currentClass
                    || currentClass.isInterface()
                    || Modifier.isAbstract(modifiers)) {
                    continue;
                }
                if(!NotChangeable.class.isAssignableFrom(currentClass)) {
                    continue classLoop;
                }
                for (Constructor<?> constructor : currentClass.
                    getConstructors()) {
```
Kern der Funktion sind verschachtelte Schleifen, die über alle ermittelten Klassennamen und jede Schnittstelle der Brokerspezifikation iterieren. Unter Verwendung des Classloaders werden im Körper der inneren Schleife erst Referenzen auf die jeweiligen Klassenobjekte erzeugt, so dass im Anschluss eine Überprüfung von Eigenschaften und der Vererbungsbeziehung erfolgen kann. Zu den Eigenschaften, die zur Filterung eines "Kandidaten" führen, zählen dabei:

Zuweisung nicht möglich Die Methode isAssignableFrom ermittelt, ob die aktuelle Klasse die gesuchte Schnittstelle implementiert. Ist dies nicht der Fall, wird der Iterationsschritt übersprungen.

Aktuelle Klasse entspricht gesuchter Schnittstelle Handelt es sich bei der aktuell untersuchten Klasse um die Klasse der gesuchten Schnittstelle, entfällt die weitere Verarbeitung.

Klasse ist ein Interface oder abstrakt Da abstrakte Klassen und Interfaces nicht instanziiert werden können, entfällt auch hier die weitere Verarbeitung.

Klasse implementiert die Schnittstelle NotChangeable Die Markerschnittstelle NotChangeable dient der Kennzeichnung von Klassen, die zwar ein oder mehrere Komponenten-Interfaces implementieren, aber nicht zur Konfiguration im Rahmen der GUI geeignet sind. Beispiele hierfür sind ConsumerFactPublisher, der intern Messungsdaten an Konsumenten weiterleitet und ListenerResultPublisher, der eine ähnliche Aufgabe in Bezug auf Analyseergebnisse erledigt.

Klassen, die alle Anforderungen an einen Broker-Bestandteil erfüllen, werden abschließend einer Liste gleichartiger Komponenten hinzugefügt. Die Ermittlung der entsprechenden Liste erfolgt über eine Map, der als Schlüssel der Schnittstellenname des aktuell

![Abbildung 5.14: Konfigurationsdialog für Filter](image1)

![Abbildung 5.15: Konfigurationsdialog für Provider](image2)

Der Unterschied zwischen den beiden Steuerelementen ergibt sich aus der Datenherkunft. So listet `initActiveTabPanel` unter Einsatz der `ClientFacade` all die Komponenten auf, die bereits ein Bestandteil des Brokers sind. Die Methode `initAvailableTabPanel` nutzt im Gegensatz dazu die Liste vorhandener Module des `BrokerComponentReflectors`, deren Erzeugung zuvor erläutert wurde. Da es sich bei den Elementen dieser Liste um Klassenreferenzen handelt, ist für die weitere Verwendung eine Instanziierung per Reflection erforderlich. Die erzeugten Objekte werden im Anschluss der Methode `getXXXActivatePanel` des `BrokerComponentReflectors` als Parameter übergeben, der für die automatische Generierung von Konfigurationsfeldern sorgt. Für die Erzeugung

---

XXX steht in diesem Zusammenhang für die jeweilige Art der Komponente. Beispiele wären `FactFilter` oder `ResultProvider`.

Als Grundlage für die Kennzeichnung einer Eigenschaft des Brokermoduls fungiert die Java Beans Namenskonvention: Attribute werden nur dann aufgeführt, wenn sie sowohl lesende als auch schreibende Operationen definiert sind und standardkonform benannt wurden - dabei gelten die Methodenprefixe is bzw. get für den Lesezugriff und set für den Schreibzugriff. Der Rest des Funktionsnamens wird als Beschreibung genutzt. So ergibt sich im konkreten Fall aus dem Methodenpaar getProductionStep und setProductionStep die Bezeichnung ProductionStep. Auch in diesem Bereich erfolgt die Überprüfung und Ermittlung der Bestandteile eines Methodennamens unter Verwendung von regulären Ausdrücken aus Listing 5.4.


Das eingangs genannte Interface BrokerConfigListener, das von allen Dialogen implementiert werden muss, wird nach dem Beobachter-Muster zur Benachrichtigung bei Änderungen der Broker-Konfiguration genutzt. So erfolgt das Hinzufügen und Entfernen von Modulen innerhalb eines speziellen Aktionsobjekts, das an die Instanz eines jeweiligen JButton übergeben wird. Wird die Aktion ausgeführt, so folgt der Ausführung der eigentlichen Aufgabe die Benachrichtigung aller Dialoge, die sich zuvor als Beobachter angemeldet haben. Die Information, das Module dem Broker hinzugefügt bzw. entfernt wurden, löst im Anschluss die Aktualisierung der Anzeige aus.

Listing 5.4: Reguläre Ausdrücke zur Analyse von Attributen eines Objekts

```
protected String getterString = "^(get|is)";
protected String setterString = "^(set)";
protected String attributeString = "([A-Z][\w]++)";
```

21 wird also auf den Button geklickt
5.10 Persistenz


5.11 Regelerstellung


5.11.1 Festlegung der Toleranzbasis


---

22 Derzeit existieren drei Filterkriterien: Nach Produkt, nach Produktionsschritt und nach Messungsart
23 Es handelt sich dabei derzeit um einen MySQL-Server
Abbildung 5.16: Flussdiagramm des Algorithmus zur Mittelwertbestimmung
Menge wird die Standardabweichung (hier \( \text{StdDev} \)) genutzt:

\[
\text{StdDev} = \sqrt{\frac{1}{n-1} \sum_{i=1}^{n} (x_i - \bar{x})^2}
\] (5.1)

Die Variable \( n \) entspricht der Anzahl an Elementen, die in Form von \( x_i \) in die Berechnung einfließen. Die mit \( \bar{x} \) bezeichnete Variable kennzeichnet das arithmetische Mittel. Da es sich bei diesem Wert um eine absolute Größe handelt, muss die Festlegung von Grenzwerten unter Berücksichtigung der Messungsart und des Produktionsschritts erfolgen.

Auf Basis einer Menge an Messungsdaten, die von funktionierenden Sensoren stammen, folgt differenziert nach Messungsart und Produktionsschritt eine Überprüfung auf die verfügbare Anzahl der Datensätze. Sollten weniger als \( N \) Datensätze existieren, bezieht sich die gespeicherte Prozedur auf die Berechnung des arithmetischen Mittels. Anderenfalls folgen bis zur Erreichung des Grenzwerts der Standardabweichung, der im Vorfeld festgelegt wird, weitere Schritte: Zwei temporäre Tabellen, die mit \( \text{master} \) und \( \text{slave} \) bezeichnet sind, dienen in diesem Zusammenhang als Zwischenspeicher. In jedem Iterationsschritt werden die Daten der Tabelle \( \text{slave} \) gelöscht und durch die Einträge der Master-Tabelle ersetzt. Im Anschluss daran erfolgt die erneute Bestimmung der Standardabweichung ohne die Berücksichtigung von minimalen bzw. maximalen Messwert, so dass ein Vergleich der ermittelten Standardabweichungen zur Entfernung des Datensatzes führt, der weniger zur Vereinheitlichung beiträgt.

Die hierdurch erzeugten Werte stellten ein wesentliches Fundament zur Festlegung von Toleranzen dar und werden in der Wissensbasis der Infrerenzmaschine genutzt.

5.11.2 Aufbau der Regeln

Der Aufbau einer Regel im Rahmen der Analysephase orientiert sich stark am Aufbau der entsprechenden Ergebnisstruktur. Um dem Anwender, der meist über keine Programmiererfahrung verfügt, eine Vorlage zur Verfügung zu stellen, die ähnlich einem Formular ausgefüllt werden kann, entspricht die Struktur einer Regel dem folgenden Muster:

\[\text{Absolute Größe}\] bedeutet in diesem Zusammenhang, das Ergebnisse der Funktion keinen prozentualen Bezug zu ihren Parametern haben - die Standardabweichung der Menge \( m_1=\{10, 20, 30\} \) gleich der von \( m_2=\{110, 120, 130\} \)
5.11 Regelerstellung

```java
rule "Name_der_Regel"
    ruleflow-group "Analyse"
    when
        $qualityFact : QualityFact($q : quality,
            $pid : productId,
            productionStep == ProductionStep.SPOOLING)
        $inductanceFact : InductanceFact($i : inductance,
            productId == $pid,
            productionStep == ProductionStep.SPOOLING)
    eval($q < 4711)
    eval($i > 42)
    then
        AnalysisRuleResult ruleResult = analysisResult.createRuleResult($pid, "Name_der_Regel");
        ruleResult.setRemoveOnMeasurementFaults(true);
        ruleResult.setFaultFactor(1);
        ruleResult.setPrecisionFactor(1);
        ruleResult.setEpsilonArray(new float[]{0.0f, 0.0f, 0.0f});
        ruleResult.addFact($qualityFact);
        ruleResult.addFact($inductanceFact);
        ruleResult.appendHint("Erster Hinweis");
        ruleResult.appendHint("Zweiter Hinweis");
        analysisResult.addRuleResult(ruleResult);
end
```

Listing 5.5: Beispiel für eine Regel der Analysephase

Somit ist zur Formulierung einer Regel die Beantwortung der aufgeführten Fragen erforderlich:

**Zielsetzung:** Ist das Ziel der Regel die Analyse des Produkts oder sollen mögliche Fehlmessungen identifiziert werden?

**Messungen:** Welche Messungen müssen berücksichtigt werden?

**Produktionsschritte:** Auf welche Produktionsschritte bezieht sich die Regel?

**Eigenschaften der Messungen:** Welche Attribute einer Messung sind im konkreten Fall für die Ausführung relevant?

**Fehlmessung:** Soll das Regelergebnis entfernt werden, sofern sich im weiteren Verlauf herausstellt, dass es sich um einen Messfehler handelt?

**Fehlerfaktor:** Wie ist die Abweichung von erwarteten Messwerten zu beurteilen?
5.11 Regelerstellung

**Präzisionsfaktor:** Wie plausibel sind die Daten der berücksichtigten Fakten?

**Epsilonfaktoren:** Wie wirken sich die neuen Erkenntnisse über das Produkt auf dessen charakteristische Eigenschaften aus?

**Hinweise:** Welche Handlungsanweisungen ergeben sich für den Mitarbeiter in der Produktionsabteilung aus der Situation?

Wie bereits angedeutet, unterteilt sich die Analyse durch einen RuleFlow in zwei Phasen. Die Gruppe der Regeln, die zur Erkennung von Fehlmessungen dienen, wird zuerst ausgeführt. Um eine Regel dieser Gruppe zuzuordnen, muss nach dem Schlüsselwort `ruleflow-group` der Name der Ausführungseinheit (hier *Fehlmessung*) angegeben werden.

Der übrige Aufbau der Regel unterscheidet sich vom obigen Beispiel durch vier zusätzliche Zeilen, die hier exemplarisch für eine Induktivitätsfehlmessung angeführt werden:

```java
1 MeasurementFaultFact faultFact = analysisResult.createMeasurementFaultFact($inductanceFact);
2 insert(faultFact);
3 retract($inductanceFact);
4 analysisResult.rejectResults(faultFact);
```

Listing 5.6: Beispiel des zusätzlichen Regelfragments bei Fehlmessungen


Das in Abbildung [5.17] aufgeführte Flussdiagramm stellt den Ablauf schematisch dar. Dabei repräsentiert Messung A das Ergebnis einer Überprüfung des Produkts, die vor

25Die erneute Messung ergibt einen stark abweichenden Wert
Abbildung 5.17: Flussdiagramm der Erkennung von Fehlern nach Folgemessungen

Da die von Drools unterstützte Syntax des Regelwerks nicht die Deklaration von Konstanten erlaubt, existiert für jede Messgröße eine Funktion, die ihren statistischen Mittelwert liefert. Diese Vorgehensweise stellt sicher, das Änderungen an den Vorgaben an zentraler Stelle erfolgen können und inkonsistente Angaben vermieden werden.

Weil sich die Zielsetzung im Bereich der Reflektion nicht mit einem sehr einheitlichen Aufbau der Regeln, wie sie in der Analysephase genutzt werden, vereinen lässt, ist es nicht möglich ein typisches Beispiel für eine solche Regel anzuführen. Dies bezieht sich insbesondere auf ein breites Spektrum an Bedingungen, die für das Auffinden von falsch oder ungenau formulierten Regeln erforderlich sind. Um dennoch ein Anwendungsgebiet zu skizzieren, ist eine Regel zur Identifizierung falsch prognostizierter Mittelspannungen in Listing 5.7 dargestellt.

```
rule "Umid->fehlerhaft, Sensor-Epsilon-Faktor->OK"
    when
        $midFact : VoltageFact($uMid : voltage, type == "MID", $pid : productId, 
        productionStep == ProductionStep.WOBBLE) 
        $productResult : AnalysisProductResult(productId == $pid) 
            ( 
                eval($uMid > getUmid() + getUMidDeviation()) or
                eval($uMid < getUmid() - getUMidDeviation()) 
            )
            eval($uMid + $productResult.getEpsilonArray()[1] <= getUmid() + 
            getUMidDeviation())
            eval($uMid + $productResult.getEpsilonArray()[1] >= getUmid() - 
            getUMidDeviation())
    then
        ReflectionRuleResult ruleResult = reflectionResult.createRuleResult("Umid-> 
        fehlerhaft, Sensor-Epsilon-Faktor->OK", $productResult); 
        ruleResult.addFact($midFact); 
        ruleResult.addCategory(ReflectionCategory.PRODUCT); 
        ruleResult.appendHint("Falscher Wert für Umid bei Barcode "+ $pid + "'!"); 
        ruleResult.appendHint("Wert von Umid liegt ausserhalb der Toleranz"); 
        ruleResult.appendHint("Ergebnis (erwartet): " + ($productResult. 
        getEpsilonArray()[1] + getUmid())); 
        ruleResult.appendHint("Ergebnis (gemessen): " + $uMid);
end
```

Listing 5.7: Reflektionsregel zur Ermittlung falsch prognostizierter Spannungswerte

Für das Verständnis der LHS ist es erforderlich zu wissen, dass Teilbedingungen ohne die explizite Verknüpfung mit dem Schlüsselwort or konjunktiv verbunden werden. Somit sind nur dann alle Vorraussetzungen der Regel erfüllt, wenn der Wert der Spannungsmessung die zuvor definierten Toleranzgrenzen über- bzw. unterschreitet und das

\footnote{Eine alternative Bezeichnung für den Bedingungsteil einer Regel}
Ergebnis der Analyse die Einhaltung der Grenzen vorsah.

5.12 Regelverwaltung

Die Inferenzmaschine Drools bietet zwei Möglichkeiten um Regeln zu importieren:


Abbildung 5.18: Frontend des BRMS (Liste der Regeln)

Aus diesen Gründen wurde im Rahmen des Drools-Projekts eine Erweiterung vorgesehen: Das BRMS\(^{28}\) ist eine J2EE-Anwendung, die als WAR-Datei\(^{29}\) angeboten wird und somit auf gängigen Servlet-Containern und Applikationsservern ausführbar ist. Zu den

\(^{27}\)XXX ist ein Platzhalter für Drl oder Xml

\(^{28}\)Business Rule Management System

\(^{29}\)Web ARchive
Bestandteilen zählt neben einem Web-Frontend und den Komponenten zur Regelkomplikierung auch eine Versionsverwaltung, mit deren Hilfe eine zentrale Speicherung der Wissensbasis möglich wird. Die Klasse `DroolsMachineImpl` enthält zur Initialisierung der Regelbasis die in Listing 5.8 aufgeführte Methoden.

```java
protected void initAnalysis() {
    analysisRuleAgent = RuleAgent.newRuleAgent(PropertyLoader.
        BRMS_ANALYSIS_CONFIG_FILE);
    analysisRuleBase = analysisRuleAgent.getRuleBase();
    analysisSession = analysisRuleBase.newStatefulSession();
    analysisResult = new AnalysisResultImpl();

    analysisResult.addResultListener(this);

    analysisSession.setGlobal("analysisResult", analysisResult);
}
```

Listing 5.8: Initialisierungsmethode der Klasse `DroolsMachineImpl`


Das BRMS sieht für die Veröffentlichung von Neuerungen zwei Schritte vor: Nach der Bearbeitung von Regeln über das Web-Frontend oder nach dem Hochladen einer modifizierten DRL-Datei folgt der Schritt `Build, validate and deploy`, in dem die syntaktische Fehlerfreiheit und das Vorhandensein aller benötigten Abhängigkeiten überprüft wird. Ist dies der Fall, so folgt unmittelbar die Kompilierung und eine Benachrichtigung bei Erfolg. Nun kann der Anwender entscheiden, ob die kompilierte Datei heruntergeladen werden soll, um sie manuell nach dem oben geschilderten Verfahren zu nutzen, oder die Veröffentlichung eines so genannten Snapshots veranlasst wird. Die Erstellung eines Snapshots führt im Rahmen der nächsten Anfrage eines RuleAgent zu dessen Aktualisierung.

### 5.13 Tests

Neben der manuellen Durchführung von Tests nach Erweiterungen und Änderungen am Quellcode, werden zentrale Bestandteile des Expertensystems automatisiert auf se-

---

30 Die Methode existiert in zwei Ausprägungen - eine für die Analyse und eine weitere für die Reflektion.

```java
protected EqualsHashCodeTest<QualityFactImpl> equalsHashCodeTester =
    new EqualsHashCodeTest<QualityFactImpl>();

public void testEquals() {
    long randomId = getRandomLong();
    long randomCreation = getRandomLong();
    float randomValue = getRandomFloat();

    QualityFactImpl first = new QualityFactImpl();
    first.setProductId(randomId);
    first.setQuality(randomValue);
    first.setCreationTime(randomCreation);

    QualityFactImpl second = (QualityFactImpl) first.clone();
    QualityFactImpl third = (QualityFactImpl) first.clone();

    try {
        assertTrue(equalsHashCodeTester.checkReflexivity(first));
        assertTrue(equalsHashCodeTester.checkSymmetry(first, second));
        assertTrue(equalsHashCodeTester.checkTransitivity(first, second, third));
        assertTrue(equalsHashCodeTester.checkNullReference(first));

        assertTrue(equalsHashCodeTester.checkRandomAttributes(first));
    } catch (Exception e) {
        fail(e.getMessage());
    }
}
```

Listing 5.9: Überprüfung der `equals`-Methode unter Verwendung der Hilfsklasse

Die mit `checkXXX` bezeichneten Funktionen der Hilfsklasse beziehen sich auf die allgemeinen Anforderungen, die an eine gute Implementation der `equals`-Methode gestellt werden: So sind einseitig Kriterien einer Äquivalenzrelation zu erfüllen (Reflexivität, Symmetrie und Transitivität) und andererseits muss sichergestellt werden, dass die Angabe eines Nullzeigers als Parameter nicht zu einer Ausnahme führt. Die Funktion `checkRandomAttributes` erweitert die Qualität des Tests um eine Überprüfung sämtlicher Attribute: Ausgehend von der Annahme, dass eine Änderung an einer beliebigen Eigenschaft zu einer Änderung des Rückgabewerts von `equals` führen muss, erfolgt dabei ein Vergleich zwischen einem Referenzobjekt und einer Liste von modifizierten Objekten, deren Länge gleich der Anzahl an öffentlichen Attributen der zu untersuchenden Instanz ist. Handelt es sich beispielsweise um eine Instanz der Klasse `AnalysisRuleResultIm-`
pl, die über einen zuvor festgelegten Zustand verfügt, so wird unter Verwendung der Reflection-API in Kombination mit regulären Ausdrücken für jede Eigenschaft (RuleName, RemoveOnMeasurementFaults, ProductId, etc.) ein modifiziertes Exemplar erzeugt. Diese Vorgehensweise stellt sicher, dass eine nachträgliche Erweiterung einer Klasse nicht zu versehentlichen Inkonsistenzen innerhalb der equals-Überprüfung führt.

In Bezug auf die fehlende Injektivität der Abbildung eines Objektzustands auf eine Zahl vom Typ long beschränkt sich die Überprüfung der Methode hashCode auf das Testen gleicher Rückgabewerte bei äquivalenten Objekten.

5.13.1 Fakten

Da diese Objektstruktur in sämtlichen Bereichen der Software Anwendung findet, ist mit der Sicherstellung der korrekten Funktionsweise ein wesentlicher Beitrag zur Stabilität des Systems geleistet. Abbildung 5.19 repräsentiert das Klassendiagramm der Testumgebung. Ausgehend von der Nutzung des Besucher-Entwurfsmusters ist in er-

### 5.13.2 Ergebnisse

6 Ausblick und Fazit

Die im Rahmen des Projekts entstandene Software stellt ein solides Fundament für weitere Ergänzungen dar, deren Notwendigkeit sich aus neuen Produktionsverfahren und der Optimierung von Kommunikationswegen ergibt.


\[1\text{Eine Gleichstellung zeichnet sich durch die voneinander unabhängige Koexistenz mehrerer Analysemaschinen aus}\]
regelbasierte Diagnose genutzt werden können.

Neben den Erweiterungen um Broker-Module oder Analysemaschinen ist der Wechsel zur Version 5 der Inferenzmaschine Drools als sehr sinnvoll anzusehen. Leider erfolgte die Veröffentlichung der überarbeiteten und erweiterten Fassung erst nach Abschluss der Implementationsphase, so dass eine nachträgliche Umstellung aus Zeitgründen entfallen musste. Der entscheidende Unterschied, der gegen eine Beibehaltung der Version 4.0.7 spricht, ist das Vorhandensein teilweise erheblicher Mängel des BRMS:


Eine Funktionalität der Inferenzmaschine, die bislang noch keine Verwendung fand, aber

\[2\text{in Kombination mit einem entsprechenden Plugin}\]
gerade für den Einsatz in Produktionsumgebungen geeignet scheint, ist die Ergänzung der Wissensbasis um Entscheidungstabellen.


Die optionale Zielsetzung der Generierung von Daten zur Kalibrierung des Konverters lässt sich derzeit aufgrund starker Streuungen bei der Messungserfassung nicht realisieren. Versuche ergaben für diesen Zweck zu große Abweichungen bei wiederholten Messreihen, so dass auf diese Funktionalität verzichtet werden muss.

Zusammenfassend entspricht der Funktionsumfang den Anforderungen, die im Rahmen der Konzeptionsphase ermittelt wurden. Darüber hinaus bietet das System den Raum für Erweiterungen und Anpassungen, die eine langfristige Weiterentwicklung sinnvoll erscheinen lässt.
Hiermit versichere ich, dass ich die vorliegende Bachelor-Abschlussarbeit ohne Hilfe Dritter und nur mit den angegebenen Quellen und Hilfsmitteln angefertigt habe. Alle Stellen, die aus den Quellen entnommen wurden, sind als solche kenntlich gemacht worden. Die Arbeit wurde bisher in gleicher oder ähnlicher Form keiner anderen Prüfungsbehörde vorgelegt und auch nicht veröffentlicht.

(Christoph Zuleger)

Münster, 12. September 2009
# Abbildungsverzeichnis

<table>
<thead>
<tr>
<th>Abbildungsverzeichnis</th>
<th>Seite</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.1 Magnetfeld einer von Wechselstrom durchflossenen Spule</td>
<td>7</td>
</tr>
<tr>
<td>3.2 Ergebnisse des Wobbel-Messplatzes</td>
<td>8</td>
</tr>
<tr>
<td>3.3 Fallbeispiel des RETE-Graphen</td>
<td>11</td>
</tr>
<tr>
<td>3.4 Syntax einer Regel</td>
<td>12</td>
</tr>
<tr>
<td>3.5 Netzwerk mit drei Schichten von Neuronen</td>
<td>14</td>
</tr>
<tr>
<td>4.1 Anwendungsfalldiagramm für die Sensorreihe PR6423</td>
<td>24</td>
</tr>
<tr>
<td>4.2 Komponentenverteilung des Expertensystems</td>
<td>26</td>
</tr>
<tr>
<td>4.3 Schematische Darstellung der Reflektionsphase</td>
<td>28</td>
</tr>
<tr>
<td>5.1 ER-Diagramm der Access-Datenbank</td>
<td>32</td>
</tr>
<tr>
<td>5.2 Klassendiagramm zur Kommunikation zwischen Fakten und Konsumenten</td>
<td>33</td>
</tr>
<tr>
<td>5.3 Sequenzdiagramm der Kooperation zwischen Result und ResultListener</td>
<td>35</td>
</tr>
<tr>
<td>5.4 Hierarchie der Ergebnisstruktur und Fallbeispiel für Fehlmessungen</td>
<td>37</td>
</tr>
<tr>
<td>5.5 Klassendiagramm der Ergebnisstruktur der Inferenzmaschine</td>
<td>38</td>
</tr>
<tr>
<td>5.6 Klassendiagramm der Ergebnis auswertenden Komponenten</td>
<td>41</td>
</tr>
<tr>
<td>5.7 Die Bestandteile eines Brokers</td>
<td>42</td>
</tr>
<tr>
<td>5.8 Beispiel für die Kooperation der Broker-Komponenten</td>
<td>45</td>
</tr>
<tr>
<td>5.9 Kooperation der Broker zwischen Client und Server</td>
<td>47</td>
</tr>
<tr>
<td>5.10 Anzeige des Clients nach Start der Anwendung</td>
<td>51</td>
</tr>
<tr>
<td>5.11 Anzeige der erfassten Messungen</td>
<td>51</td>
</tr>
<tr>
<td>5.12 Anzeige von Analyseergebnissen der Inferenzmaschine</td>
<td>52</td>
</tr>
<tr>
<td>5.13 Anzeige von Reaktionsergebnissen</td>
<td>54</td>
</tr>
<tr>
<td>5.14 Konfigurationsdialog für Filter</td>
<td>58</td>
</tr>
<tr>
<td>5.15 Konfigurationsdialog für Provider</td>
<td>58</td>
</tr>
<tr>
<td>5.16 Flussdiagramm des Algorithmus zur Mittelwertausschreibung</td>
<td>61</td>
</tr>
<tr>
<td>5.17 Flussdiagramm der Erkennung von Fehlern nach Folgemessungen</td>
<td>65</td>
</tr>
<tr>
<td>5.18 Frontend des BRMS (Liste der Regeln)</td>
<td>67</td>
</tr>
<tr>
<td>5.19 Klassendiagramm der Testkonfiguration für Fakten</td>
<td>70</td>
</tr>
</tbody>
</table>
Listings

3.1 Nutzung von Zeichenklassen in regulären Ausdrücken (Java) . . . . . . . 16
3.2 Nutzung von Alternationen und Gruppen in regulären Ausdrücken (Java) 16
3.3 Nutzung von Quantoren in regulären Ausdrücken (Java) . . . . . . . . . 17

5.1 Konstanten zur Festlegung des Konfigurationsdateinamens . . . . . . . . 49
5.2 Konstante Arrays mit den vollqualifizierten Namen der Brokerinterfaces 56
5.3 Methode zur Ermittlung von Brokerkomponenten . . . . . . . . . . . . . 56
5.4 Reguläre Ausdrücke zur Analyse von Attributen eines Objekts . . . . . . 59
5.5 Beispiel für eine Regel der Analysephase . . . . . . . . . . . . . . . . . . 63
5.6 Beispiel des zusätzlichen Regelfragments bei Fehl­messungen . . . . . . 64
5.7 Reflektionsregel zur Ermittlung falsch prognostizierter Spannungswerte . 66
5.8 Initialisierungsmethode der Klasse DroolsMachineImpl . . . . . . . . . 68
5.9 Überprüfung der equals-Methode unter Verwendung der Hilfsklasse . . . 69
Literaturverzeichnis


