Technische Universität Braunschweig
Institut für Software Systems Engineering

**Hilfswissenschaftliche Tätigkeit**

# Entwicklung einer Grammatik für MontiCore zum Parsen von Sensordaten

**Jan Oliver Ringert**
Matrikel-Nr.: 2772985

**Aufgabenstellung:** Eshref Januzaj
**Betreuer:** Eshref Januzaj

Braunschweig, den 20. Februar 2007

# Abstract

Diese Arbeit beschreibt die Entwicklung einer Grammatik in MontiCore (GKR06), mit der das Parsen Sensordaten eines bestimmten Formats möglich ist. Im Zuge der Bearbeitung der Aufgabe entstanden Probleme mit großen Datenmengen, die gesondert in (Rin07) behandelt wurden. Die zu parsenden Daten unterscheiden sich stark von normalen domänenspezifischen Sprache (DSL), für deren Entwicklung MontiCore ausgelegt ist. Probleme und Lösungswege werden in dieser Arbeit vorgestellt.

## Inhaltsverzeichnis

**1 Einführung** 1

**2 Eingabeformat** 2
2.1 “@meta” . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 2
2.2 “@data” . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 3

**3 Grammatik** 4
3.1 Optionen zur Codegenerierung . . . . . . . . . . . . . . . . . . . . . . . . . 4
3.2 Aufbau der Grammatik . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 5
3.3 Datenblock . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 6

**4 Verwendung der Grammatik durch MontiCore** 9

**5 Bewertung und Erweiterung** 10

**6 Vollständige Definition der Grammatik** 11

# 1 Einführung

Im Zuge einer hilfswissenschaftlichen Tätigkeit wurde die Aufgabe gestellt, eine Grammatik zu definieren, mit der Sensordaten in bestimmten Formaten gelesen werden können. Diese Daten sollen mit Hilfe des Konzeptes PrettyPrinter transformiert werden, um sie aus verschiedenen Quellen einheitlich verwenden zu können. Es sollten später weitere Grammatiken definiert werden, mit denen die verschiedenen Ausgaben von Sensoren gelesen werden könnten.
Während der Ausarbeitung einer passenden Grammatik traten Probleme auf, die zeigen, dass MontiCore nur eingeschränkt für diese Aufgabe geeignet ist. Die Probleme, die bei großen Datenmengen auftreten, werden in (Rin07) beschrieben.
Das dieser Arbeit zu Grunde liegende Dateiformat besteht aus bestimmten Abschnitten, welche hauptsächlich das Format einer CSV (comma separated values) Datei einhalten. In Abschnitt 2 wird das Format der Dateien genauer erklärt. Daraufhin wird eine Grammatik in Abschnitt 3 definiert. Es wird zu erst auf die Besonderheiten der Grammatik im Vergleich zu üblichen DSLs verwiesen und erklärt, welche Korrekturen notwendig sind. Abschnitt 4 zeigt, wie die Grammatik in einem MontiCore Projekt verwendet wird, um durch einen PrettyPrinter eine Ausgabe zu erzeugen. In Abschnitt 5 wird das Ergebnis der Arbeit zusammengefasst und Erweiterungen dargestellt.

# 2 Eingabeformat

Ziel der Arbeit ist es, eine Grammatik zu definieren, mit der Dateien geparst werden können, die einem bestimmten Format entsprechen. Die zu lesenden Daten sind Messwerte von verschiedenen Sensoren. Als erster Schritt soll ein verallgemeinertes Format eingelesen werden. Ein Beispiel für dieses Format findet sich in Listing 1.

```
1  @sse;;;
2  ;;Vergabe SSE1;Vergabe SSE2
3  @Meta;;;
4  GLT–ID;;PWCB1: HOBO Dofa, Wert 1;PWCB1: HOBO Dofa, Wert 2
5  IGSname;;104B1xy0DOFAT01;104B1xy0DOFAT02
6  Klartext;;freier Text 1;freier Text 2
7  Art des Werts;;Lufttemp Dofa;Lufttemp Dofa
8  Einheit;;°C;°C
9  Skala;;−20 bis 40;−20 bis 40
10 Platzhalter 1;;;
11 Platzhalter 2;;;
12 Platzhalter 3;;;
13 @Data;;;
14 30.06.2005;13:14:48;23,641;24,195
15 30.06.2005;13:29:48;23,809;24,388
16 30.06.2005;13:44:48;23,881;24,677
17 30.06.2005;13:59:48;23,978;24,315
18 30.06.2005;14:14:48;24,026;24,653
19 30.06.2005;14:29:48;24,339;25,307
20 30.06.2005;14:44:48;24,388;25,355
21 30.06.2005;14:59:48;24,002;24,363
22 30.06.2005;15:14:48;24,412;24,968
23 30.06.2005;15:29:48;24,677;24,774
24 30.06.2005;15:44:48;25,04;24,968
25 30.06.2005;15:59:48;24,968;25,258
26 30.06.2005;16:14:48;24,992;25,162
27 30.06.2005;16:29:48;25,113;25,355
28 30.06.2005;16:44:48;25,283;25,355
29 30.06.2005;16:59:48;24,919;24,968
30 30.06.2005;17:14:48;24,508;24,532
```

Listing 1: Beispiel einer zu lesenden Datei

Die Datei enthält Daten und Messwerte von mindestens einem Sensor (in diesem Beispiel zwei Sensoren). Sie ist aufgeteilt in die drei Blöcke "@sse", "@meta" und "@data".

## 2.1 "@meta"

Der zweite Block enthält Daten, mit denen die Sensoren identifiziert und beschrieben werden. Eine Zeile startet mit einer Überschrift gefolgt von einer leeren Zelle. Nach dieser Einleitung beginnen die Informationen zu dem jeweiligen Sensor. Dabei gibt die Spalte den Sensor an, sodass alle Datenzellen, die Sensor $i$ (mit $i = 1, ..., n$ und $n$ Anzahl der Sensoren) beschreiben in jeder Zeile die Position $i + 2$ haben. Daten, die hier angegeben werden, sind unter anderem der Name der Sensoreinheit (Überschrift "IGSname"), die "Art des Wertes" oder die "Einheit", in der die Daten gemessen werden. Es handelt sich in diesem Block größtenteils um freie Texte, die in der zu entwickelnden Grammatik als atomare Bestandteile betrachtet werden.

### 2.2 "@data"

Der letzte Block ist der größte in der Datei. Er enthält nach der Beschreibung der Sensoren deren Messwerte. Die Messwerte zu bestimmten Zeiten werden hier aufgelistet. Eine Zeile trägt in den ersten beiden Spalten Informationen über den Zeitpunkt der Messung. Darauf folgt pro Zelle ein Messwert. Die Anordnung stellt sicher, dass wie im zweiten Block die Werte zu Sensor $i$ in der Spalte $i + 2$ stehen. Jeder Messwert ist durch eine rationale Zahl gegeben.

# 3 Grammatik

Die gesamte Definition der Grammatik findet sich im Anhang als Listing 7. In diesem Kapitel werden die Besonderheiten gegenüber "normalen" Grammatikdefinitionen behandelt. "Normal" bezieht sich auf grundlegende Beispiele von Definitionen aus (GKR06). Weiterhin wird der Aufbau und die Entwicklung der Grammatik aus Listing 7 erläutert.

## 3.1 Optionen zur Codegenerierung

Die Optionen zur Codegenerierung werden in Abschnitt 5.1 von (GKR06) beschrieben. In dieser Grammatik verwendete Optionen sind in Listing 2 angegeben.

```
1          options {
2                  parser lookahead=3
3                  lexer lookahead=2
4                  nows
5                  noident
6                  nostring
7                  noslcomments
8                  nomlcomments
9          }
```

Listing 2: Optionen zur Codegenerierung aus Listing 7

Zusätzlich zu den üblichen Optionen sind die Optionen aus Zeile vier bis acht notwendig.

**"nows"**
: Diese Option sorgt dafür, dass MontiCore beim Erstellen des Lexers keine Regeln für das Erkennen von Zwischenräumen verwendet. Wird diese Regel nicht benutzt, so gelten Leerzeichen und Zeilenumbrüche automatisch als Beginn eines neuen Tokens und werden entfernt. Das Verhalten hat den Nachteil, dass Zeilenumbrüche nicht mehr erkannt werden können. Diese sind jedoch wichtig, da das Ende eines Feldes in dem Daten-Abschnitt nur durch einen solchen Umbruch gekennzeichnet ist. Weiterhin dürfen Leerzeichen nicht zum Terminieren eines Tokens führen, da innerhalb des Metadaten-Blocks Leerzeichen in Zellen verwendet werden dürfen.

**"noident" & "nostring"**
: MontiCore generiert standardmäßig die beiden Identifier IDENT und STRING (siehe (GKR06) Abschnitt 5.2). Diese Identifier sind nicht kompatibel mit den hier benötigten (siehe Abschnitt 3.2). Es tritt ein Nichtdeterminismus bei der Erkennung durch den generierten Lexer auf.

**"noslcomments" & "nomlcomments"**
: Da MontiCore für die Entwicklung von domänenspezifischen Sprachen (DSLs) entwickelt wurde, bietet es standardmäßig eine Behandlung von Kommentaren in dem zu parsenden Quelltext an. Diese Behandlung sieht vor, dass erkannte Kommentare vom Lexer ignoriert und nicht weiter geparst werden. Kommentare werden durch ein führendes "//" oder durch Einschließen in "/*" und "*/" erkannt. Diese Zeichen sind in den freien Texten der Metadaten erlaubt und dürfen nicht ignoriert werden.

Aus Listing 2 wird deutlich, dass viele der Hilfestellungen, die dem Entwickler durch MontiCore geboten werden, für diese Arbeit deaktiviert werden müssen. Das einzulesende Datenformat ist hier ein grundlegend anderes, als es bei üblichen DSLs der Fall ist.

## 3.2 Aufbau der Grammatik

An erster Stelle müssen geeignete Konstruktionen für die atomaren Bausteine (Tokens) der zu lesenden Datei gefunden werden (für Erklärungen von Tokens und der Funktionsweise des Lexers und Parsers siehe (Mil05) Kapitel 2). Der normale Weg zur Entwicklung einer Grammatik führt von der höchsten Abstraktionsebene zur niedrigsten. Hier stellt die niedrigste Ebene jedoch einige Probleme dar. Änderungen an den vom Lexer benötigten Definitionen ziehen Anpassungen auf höheren Ebenen nach sich.
Der hier behandelte Abschnitt der Grammatik ist in Listing 3 zu sehen.

```
1  //Alle Zeichen (mind. eins) bis zum Semikolon
2  ident TITEL "(~(';'))+";
3
4  //Start mit Semikolon und endet mit Semikolon oder Zeilenumbruch
5  ident FELD "(';') (~(';'|'\\r'|'\\n'))*";
6
7  //Die Header Datei enthält ein Schema und Metadaten
8  InternalFormatFile = schema:Schema metadaten:Metadaten daten:Daten;
9
10 //Das Scheme beginnt mit dem Keyword "@sse" und es folgen Zeilen
11 Schema = "@sse" RestZeile zeilen:InfoZeile+;
12
13 //Metadaten sind durch das Keyword "@Meta" gekennzeichnet und es folgen
       Datenzeilen
14 Metadaten = "@Meta" RestZeile zeilen:InfoZeile+;
15
16 //Eine Daten Zeile besteht aus einem Titel und mehreren Feldern
17 InfoZeile = titel:TITEL? felder:FELD* "\\r\\n";
18
19 //RestZeile besteht aus beliebig vielen Feldern, die ignoriert werden
      sollen
20 RestZeile = FELD* "\\r\\n";
```

Listing 3: Aufbau der Grammatik aus Listing 7

Für den Lexer wichtig sind die Zeilen eins bis fünf, in denen zwei Identifier definiert werden (für Hinweise zu Identifiern siehe (GKR06) Abschnitt 5.2). Der erste Identifier heißt TITEL. Es handelt sich dabei um das erste Feld in einer Zeile, welches mit einem Semikolon endet, um das nächste Feld starten zu lassen. Die Definition in den Anführungszeichen wird direkt an das Tool ANTLR (WWWb) weitergegeben. Das Semikolon, mit dem das Feld endet ist nicht mehr Teil des ersten Feldes sondern Teil des folgenden. Der Identifier für alle weiteren Felder (Zellen) ist der Identifier FELD. Ein Feld beginnt nach dieser Definition mit einem Semikolon und endet mit einem Semikolon oder einem Zeilenumbruch. Insbesondere ist es möglich, dass ein Feld leer ist.
Der terminierende Zeilenumbruch zählt wie das Semikolon nicht zu dem FELD, da er auf höherer Ebene benötigt wird, um das Ende einer Zeile zu bestimmen (siehe Zeile 17 und Zeile 20). Das Semikolon muss jeweils zu dem folgenden FELD gehören, da sonst die Definition von FELD auch leere Zeichenketten zulassen würde. Diese können jedoch überall erkannt werden und führen dazu, dass der von MontiCore generierte Code bei Ausführung nicht terminiert.
In der entwickelten Grammatik wird zwischen mehreren verschiedenen Zeilenarten unterschieden:

**RestZeile**

Die RestZeile wird verwendet, um den Rest einer angefangenen Zeile zu lesen. Es werden keine Informationen über den Inhalt der Felder gespeichert.

**InfoZeile**
: Eine InfoZeile trägt einen Titel und hat eine beliebige Menge von Feldern bis zum nächsten Zeilenumbruch. Auf den Titel und die Inhalte aller Felder kann später beim Durchlaufen des AST zugegriffen werden.

**DatenZeile**
: Die DatenZeile unterscheidet sich von der InfoZeile nur soweit, dass es neben dem einleitenden TITEL noch ein weiteres ausgezeichnetes Feld gibt. Die Inhalte der beiden ersten Felder können im AST aus den Attributen datum und zeit gelesen werden. Alle folgenden Zellen werden in einer Liste gespeichert.

Der hierarchische Aufbau der Grammatik wird nun wie oben angesprochen Top-Down erläutert. Auf der obersten Ebene befindet sich die Einteilung in die drei Blöcke Schema, Metadaten und Daten (siehe Abschnitt 2). Dieser Aufbau wird in Zeile acht definiert.
Das Schema beginnt mit dem Schlüsselwort "@sse" und einer RestZeile, deren Inhalt ignoriert wird (normalerweise ist diese Zeile leer). Darauf folgt mindestens eine InfoZeile, deren Daten von Interesse sind und in den AST aufgenommen werden.
Der zweite Block sind die Metadaten, welche durch das Schlüsselwort "@meta" eingeleitet werden. Der weitere Aufbau dieses Abschnitts ist analog zu dem des Schemas.

## 3.3 Datenblock

In (Rin07) wurde bereits eine Grammatik für den Datenblock definiert. Diese beschreibt die Daten auf niedriger Ebene und setzt die aus ganzen Zahlen mit Trennzeichen wie ":", "-" und ";" zusammen. Ein Auszug aus der Definition ist in Listing 4 gegeben.

```
1      ident INT "('0'..'9')+";
2
3      //InternalFormatFile ist Wurzel der Datei
4      InternalFormatFile = Daten:Daten;
5
6      //Symbole für die Datenerfassung
7      Zeit = Stunde:INT ":" Minute:INT ":" Sekunde:INT;
8
9      Datum = Tag:INT "." Monat:INT "." Jahr:INT;
10
11     //für den Datenblock
12     Daten = "@Data" ";"+ "\\r\\n" Zeilen:DatenZeile*;
13
14     DatenZeile = datum:Datum ";" zeit:Zeit ";" Zellen:DatenZelle+;
15
16     Float = vorzeichen:"-"? vorkomma:INT ("," nachkomma:INT)?;
17
18     DatenZelle = inhalt:Float? (";" | "\\r\\n");
```

Listing 4: MontiCore-Grammatik für einen Datenblock aus (Rin07)

Eine Verbindung der beiden Definitionen ist syntaktisch möglich und ergibt das Klassendiagramm in Abbildung 1.
Mit dem erzeugten Code ist es jedoch nicht möglich, eine vollständige Datei wie z.B. in Listing 1 einzulesen. ANTLR meldet beim Erstellen des Lexers einen Nichtdeterminismus. Das Problem besteht darin, dass der erzeugte Lexer die Eingabe wie in Abschnitt 3.2 beschrieben in die Identifier TITEL und FELD zerlegt. Die Zeilen im Datenblock besitzen einen Aufbau, der genau auf dieses Muster passt. Ein Datum oder eine Uhrzeit sind damit atomare Bestandteile und können über Definitionen in der Grammatik nicht weiter zerlegt werden. Das Parsen erzeugt Fehler, da ein INT erwartet und ein TITEL geliefert wird.

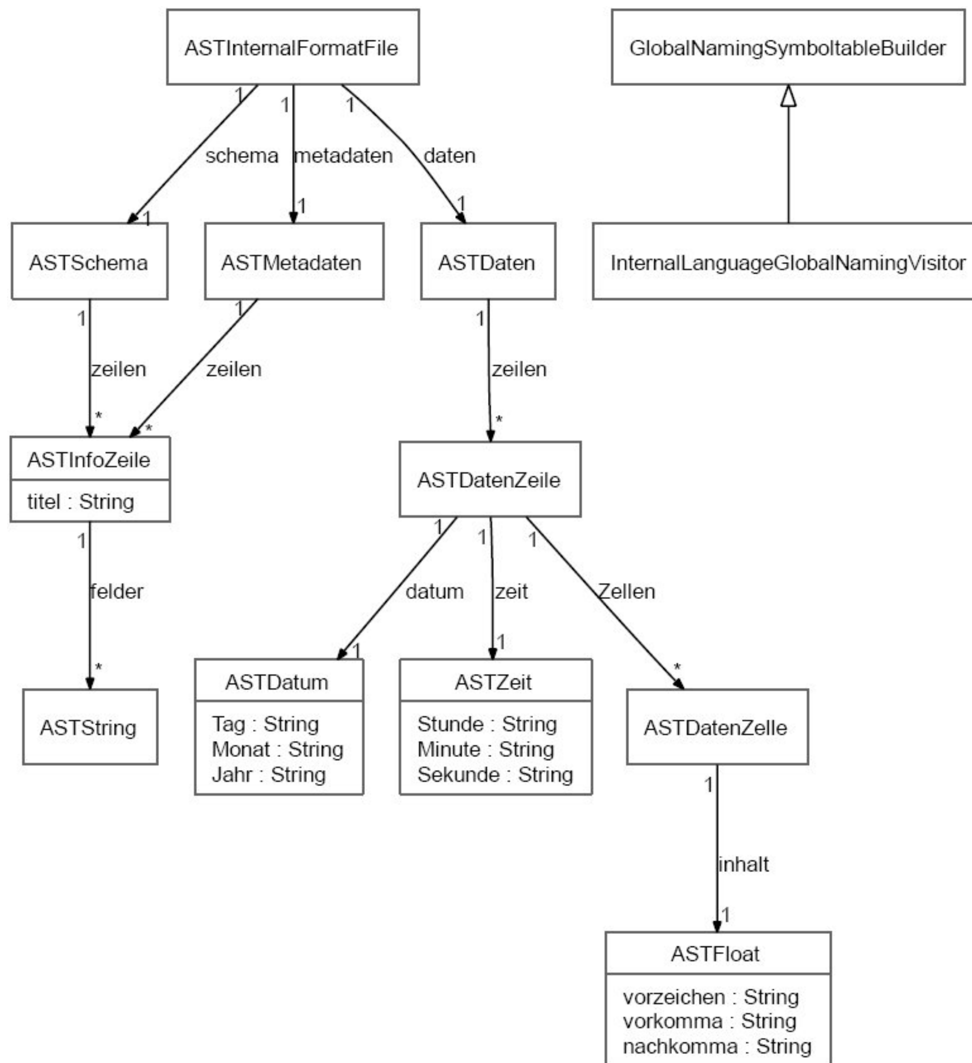


Abbildung 1: Gewünschtes Klassendiagramm aus Abschnitt 2 und (Rin07)

Um dieses Problem zu umgehen und die Ähnlichkeit des Zeilenformats auszunutzen, erfolgt das Einlesen der Daten durch die Definition in Listing 5.

```
1  Daten = "@Data" RestZeile zeilen:DatenZeile+;
2
3  //Symbole für die Datenerfassung
4  DatenZeile = datum:TITEL zeit:FELD zellen:FELD+ ("\\r\\n"|EOF);
```

Listing 5: Beschreibung des Datenblocks aus Listing 7

Ein Vorteil gegenüber der Definition aus Listing 4 ist der geringere Speicherplatzverbrauch des erzeugten ASTs. Der Parser verhält sich durch die Identifier TITEL und FELD nun eher wie ein Tokenizer mit dem Trennzeichen “;”. Eine Auflösung der Felder wie dem Datum, der Uhrzeit und der Messwerte muss beim Durchlaufen des ASTs “per Hand” geschehen. Fehler wie Buchstaben anstelle einer Dezimalzahl können nun nicht mehr vom Parser erkannt werden.
Unter Berücksichtigung der obigen Punkte ergibt sich die Grammatik aus Listing 7, welche durch das Klassendiagramm in Abbildung 2 dargestellt ist.

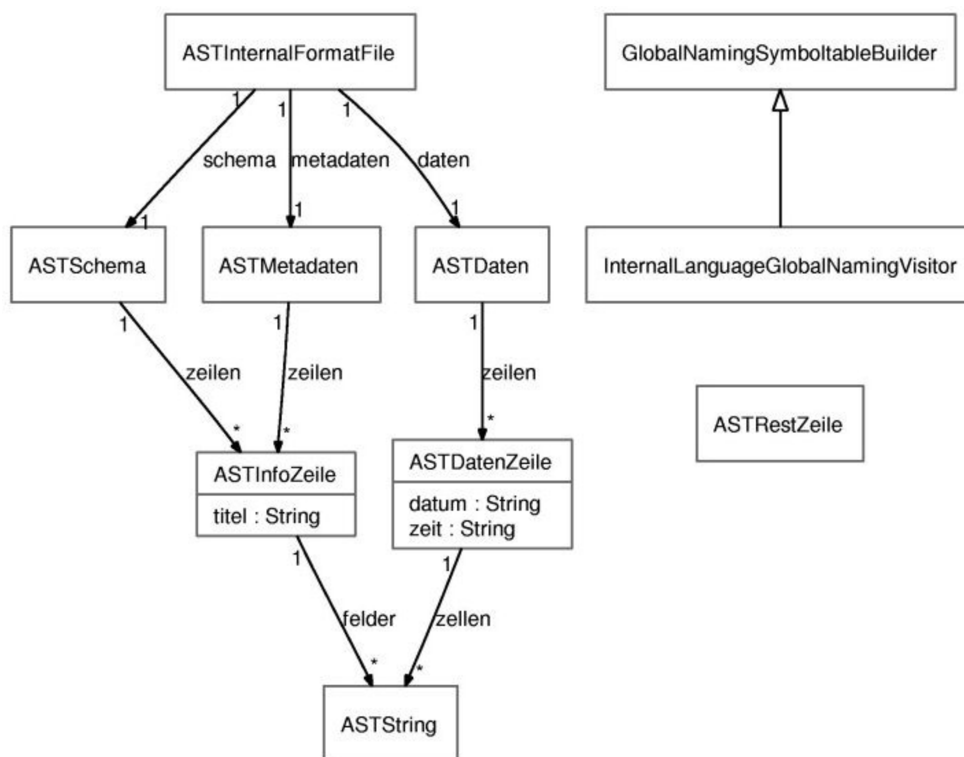


Abbildung 2: Klassendiagramm, das aus Listing 2 erzeugt wurde.

# 4 Verwendung der Grammatik durch MontiCore

Zur Verwendung der definierten Grammatik sind einige weitere Komponenten notwendig. Analog zu den Einführenden Beispielen zu MontiCore aus (GKR06) wurde ein DSLTool definiert, das der generierten MontiCore Konfiguration den Parser für ".csv" Dateien und einen PrettyPrinter hinzufügt (siehe */src/SimpleDSLTool.java*).
Für einen PrettyPrinter muss ein Workflow in MontiCore definiert werden. Dieser findet sich unter */src/datenleser/workflow/PrettyPrintWorkflow.java* und ist wie der Concrete-PrettyPrinter unter */src/datenleser/prettyprint/InternalFormatPrettyPrinter.java* analog zu den Beispielen aus (GKR06) aufgebaut.
Die eigentliche Arbeit übernimmt der ConcreteVisitor, welcher in der Datei */src/datenleser/prettyprint/InternalFormatPrettyPrinterConcreteVisitor.java* zu finden ist. Hier befindet sich für jeden Knoten des ASTs eine entsprechende Funktion, die die Daten aus den Objekten so ausgibt, wie sie in der Eingabedatei vorlagen (siehe Auszug in Listing 6). Der PrettyPrinter lässt sich mit der Datei */src/RunPrettyPrint.java* starten. Es werden alle Eingabedateien aus dem Verzeichnis */input* mit der Endung ".csv" eingelesen und durch den PrettyPrinter in dem Verzeichnis */output/prettyprint/output* unter selbem Namen wieder ausgegeben.

```
1 	public void visit(ASTSchema schema) {
2 		String output = "@sse;;;";
3 		p.print(output);
4 	}
5 
6 	public void visit(ASTMetadaten metadaten) {
7 		p.println();
8 		String output = "@Meta;;;";
9 		p.print(output);
10 	}
11 
12 	public void visit(ASTDaten daten) {
13 		p.println();
14 		String output = "@Data;;;";
15 		p.print(output);
16 	}
17 
18 	public void visit(ASTInfoZeile zeile) {
19 		p.println();
20 		String titel = zeile.getTitel();
21 		if (titel != null)
22 			p.print(titel);
23 	}
24 
25 	public void visit(ASTString string) {
26 		p.print(string);
27 	}
28 
29 	public void visit(ASTDatenZeile zeile) {
30 		p.println();
31 		p.print(zeile.getDatum() + zeile.getZeit());
32 	}
```

Listing 6: Auszug aus dem ConcreteVisitor zur Datenausgabe

# 5 Bewertung und Erweiterung

Das Ergebnis dieser Arbeit besteht neben Erkenntnissen über die Verwendbarkeit von MontiCore für ungewöhnliche DSLs aus der Definition einer Grammatik, mit der Sensordaten der in Abschnitt 2 vorgestellten Form geparst werden können. Das Einlesen der Daten besteht hauptsächlich darin, die Datei in Abschnitte zu trennen, die durch die Trennzeichen "；" und Zeilenumbrüche gegeben sind. Durch das Zulassen sehr freier Formate in der Datei (insbesondere im Abschnitt "@meta"), ist es nicht einfach möglich, den Rest der Datei detailliert zu erfassen. Das heißt, dass eine Prüfung des Inhalts der Zellen auf syntaktische Korrektheit durch die erarbeitete Grammatik nicht möglich ist.
Die Frage, ob die Regeln des Lexers (Definition der Identifier) nur auf die ersten beiden Abschnitte der Datei bezogen werden können, wurde nicht untersucht.
Es bietet sich an, zur Validierung der Daten einen Visitor über den AST laufen zu lassen, der Prüfungen an den Daten durchführt. Zu diesen Prüfungen gehören die folgenden Punkte:

- Überprüfung des syntaktischen Formats der Eingaben (vor allem bei Zeit, Datum und Messwerten).
- Konsistenzprüfung der Daten bezogen auf die Anzahl der Sensoren und die Menge der jeweiligen Messwerte.

Mit dieser Arbeit wurde eine Grammatik definiert, die Sensordaten beschreiben kann. Der durch MontiCore generierte Lexer und Parser sind in der Lage einen AST aus Datendateien zu generieren, der durch einen PrettyPrinter wieder in das Ursprungsformat der Sensordaten transformiert werden kann.

# 6 Vollständige Definition der Grammatik

```
package datenleser;

grammar InternalLanguage {

        options {
                parser lookahead=3
                lexer lookahead=2
                nows
                noident
                nostring
                noslcomments
                nomlcomments
        }

        concept dsltool {
                root InternalFormatRoot<InternalFormatFile>;

                rootfactory InternalFormatRootFactory for InternalFormatRoot<InternalFormatFile> {
                        datenleser.InternalLanguage.InternalFormatFile internalFormatfactory <<start>>;
                }

                parsingworkflow InternalFormatWorkflow for InternalFormatRoot<InternalFormatFile>;
        }

        //Alle Zeichen (mind. eins) bis zum Semikolon
        ident TITEL "(~(';'))+";

        //Start mit Semikolon und endet mit Semikolon oder Zeilenumbruch
        ident FELD "(';') (~(';'|'\\r'|'\\n'))*";

        //Die Header Datei enthält ein Schema und Metadaten
        InternalFormatFile = schema:Schema metadaten:Metadaten daten:Daten;

        //Das Scheme beginnt mit dem Keyword "@sse" und es folgen Zeilen
        Schema = "@sse" RestZeile zeilen:InfoZeile+;

        //Metadaten sind durch das Keyword "@Meta" gekennzeichnet und es folgen Datenzeilen
        Metadaten = "@Meta" RestZeile zeilen:InfoZeile+;

        Daten = "@Data" RestZeile zeilen:DatenZeile+;

        //Eine Daten Zeile besteht aus einem Titel und mehreren Feldern
        InfoZeile = titel:TITEL? felder:FELD* "\\r\\n";

        //RestZeile besteht aus beliebig vielen Feldern, die ignoriert werden sollen
        RestZeile = FELD* "\\r\\n";

        //Symbole für die Datenerfassung
```

```
49          DatenZeile = datum:TITEL zeit:FELD zellen:FELD+ ("\\r\\n"|EOF);
50
51  }
```

Listing 7: Die komplette MontiCore-Grammatik als Ergebnis dieser Arbeit

## Literatur

[GKR06] Hans Grönniger, Holger Krahn, Bernhard Rumpe, Martin Schindler, Steven Völkel . *MontiCore 1.0, Framework zur Erstellung und Verarbeitung domänenspezifischer Sprachen* . TU-Braunschweig, Institut für Software Systems Engineering, Informatik-Bericht 2006-04 . 2006 .

[Mil05] Ashley J.S Mills . *ANTLR Tutorial.* University Of Birmingham . 2005 .
`http://supportweb.cs.bham.ac.uk/docs/tutorials/docsystem/build/tutorials/antlr/antlr.pdf`

[Rin07] Jan Oliver Ringert . *Monticore zum Parsen von großen Datenmengen* . TU-Braunschweig, Institut für Software Systems Engineering . 2007

[WWWb] Webseite von ANTLR . *ANother Tool for Language Recognition.*
`http://www.antlr.org/`