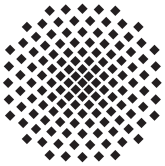

Universität Stuttgart

Handbuch
(ipv interne Version)

(Version 0.3)

05.03.2014

Ansprechpartner: Thomas Wurster, Hendrik Adler

Institut für Photovoltaik
Prof. J. H. Werner
Pfaffenwaldring 47
D-70569 Stuttgart

Inhaltsverzeichnis

Beschreibung

Sensetrace ist eine OpenSource-Software, zum Analysieren und Speichern von Sensordatenströmen. Durch Darstellung in einem RDF-Schema ermöglicht die Software benutzerfreundliche Datenabfragen. Ein Complex Event Processing System (CEP) klassifiziert den Sensordatenstrom nach vordefinierten Mustern. Das verwendete CEP-Modul kann neben neu eingehenden Messdaten auch bereits archivierte Daten klassizieren. Die Klassifizierung beschleunigt die Datenabfrage erheblich im Vergleich zu Systemen ohne Complex Event Processing. Die modulare OSGI-Architektur ermöglicht dem Entwickler eine einfache Erweiterung um neue Bestandteile und somit auch die einfache Integration beliebiger Typen von Datenloggern und Sensoren. Der Benutzer integriert CEP-Regeln, Sensoren und Gruppen von Sensoren über XML-Dateien im standardisierten SensorML-Format.

Kapitel 1

Speicherarchitektur

Das System zur Überwachung und Messung von Photovoltaikanlagen am *ipv* ist 2006 in Betrieb gegangen. Seitdem hat es die Messwerte von über 250 Sensoren aufgenommen. Sensetrace hat die Daten aus dem alten System übernommen. Dadurch speichert Sensetrace bereits jetzt eine sehr große Datenmenge von ca. $47x10^9$ Messwerten. Die Datenbank benötigt unkomprimiert einen Speicherplatz von ca. 3,5 Terrabyte.
Die Datenbank liegt auf einem ZFS-Dateisystem. Dieses erlaubt eine ca 6,5-fache Komprimierung auf Basis des Kompressionswerkzeugs gzip. Dadurch reduziert das Dateisystem die physische Speicherplatzbelegung auf nur 450 Gigabyte. Neben der Komprimierung des Dateisystems realisiert ZFS auch einen Raid-1-Verbund aus zwei Festplatten, um die Datensicherheit zu erhöhen. Kapitel 1.1 beschreibt das Vorgehen um eine defekte Festplatte in dem Verbund zu wechseln.
Der Sensetrace-Rechner läuft als virtuelle Maschine in einer Oracle VirtualBox-Umgebung. Kapitel 1.2 gibt einen Überblick über die Organisation und Aufteilung der Festplattenspeicherbereiche zwischen Host und virtueller Maschine.

1.1 Wechsel einer Festplatte im ZFS-Pool

Das ZFS-Dateisystem dient der Kompression der Datenbankdateien um einen Faktor 6 bis 7. Zusätzlich realisiert ZFS einen Raid-1-Verbund, indem es die Datenbankdateien auf zwei Festplatten spiegelt. Dadurch kommt es bei Ausfall einer Festplatte zu keinem Datenverlust. Ein Cron-Job fragt jede Stunde den Status des ZFS-Pool ab. Sollte eine Festplatte ausfallen, wird der Administrator per Email informiert. Durch folgende Schritte lässt sich die defekte Festplatte wechseln.

1. Mit "zpool status" den Status der Fesplatten überpüfen. Eine defekte Festplatte wird als "Offline" oder "Faulted" angezeigt.

2. Die defekte Festplatte entfernen und neue HDD einbauen.

3. Den Namen der neuen Festplatte mit "ll /dev/disk/by-id/" ermitteln.
   Die so ermittelte ID der Festplatte lautet zum Beispiel
   'scsi-SATA_VBOX_HARDDISK_VB1aada6b0-2230a83a-part1'

4. Um die neue Festplatte im ZFS-Pool einzubinden, folgenden Befehl eingeben:
   zpool replace scsi-SATA_VBOX_HARDDISK_VB1aada6b0-2230a83a-part1 c1t1d0

1.2 Integration des virtuellen Sensetrace-Servers auf dem Hostrechner

Der Sensetrace-Server läuft auf einem virtuellen Ubuntu Server. Dafür kommt auf dem Hostserver die Virtualisierungssoftware Oracle VirtualBox zum Einsatz. Bild 1.2 beschreibt dabei, wie der virtuelle Sensetrace-Server die physischen Festplatten des Hostservers nutzt.
Der Hostserver besitzt vier Festplatten, auf die eine Anwendung über die Gerätedateien "/dev/sda" bis "/dev/sdd" zugreift. Die ersten beiden Fesplatten "/dev/sda" und "/dev/sdb" realisieren einen Raid-1-Verbund. Raid-1 erhöht die Datensicherheit, indem es den Inhalt beider Festplatten synchron hält. Der Raid1-Verbund beinhaltet neben dem Betriebssystem, ein "/home/"-Verzeichnis, in dem zwei virtuelle Festplattenabbilder in Form von vdi-Dateien liegen. Oracle VirtualBox nutzt diese vdi-Dateien zur Simulation zweier Festplatten, die der virtuellen Sensetrace-Server als Swap-Bereich und zur Ablage der Dateien des Betriebssystem nutzt. Die Festplatten mit den Gerätedateien "/dev/sdc" und "/dev/sdd" hingegen reicht VirtualBox direkt an den virtuellen Sensetrace-Server durch. So muss der Host keine Festplatte aus einer vdi-Datei simulieren, was rechenaufwendiger ist, als ein Gerät direkt durchzureichen. Der virtuelle Server bildet aus den beiden Festplatten einen ZFS-Raid-1-Pool, in dem eine ZFS-Partition liegt, die die Dateien der PostrgreSQL-Datenbank verwaltet. Das ZFS-Dateisystem ermöglicht eine Komprimierung der Datenbankdateien um den Faktor 6,5. Das bringt erhebliche Einsparungen bei den Hardware-

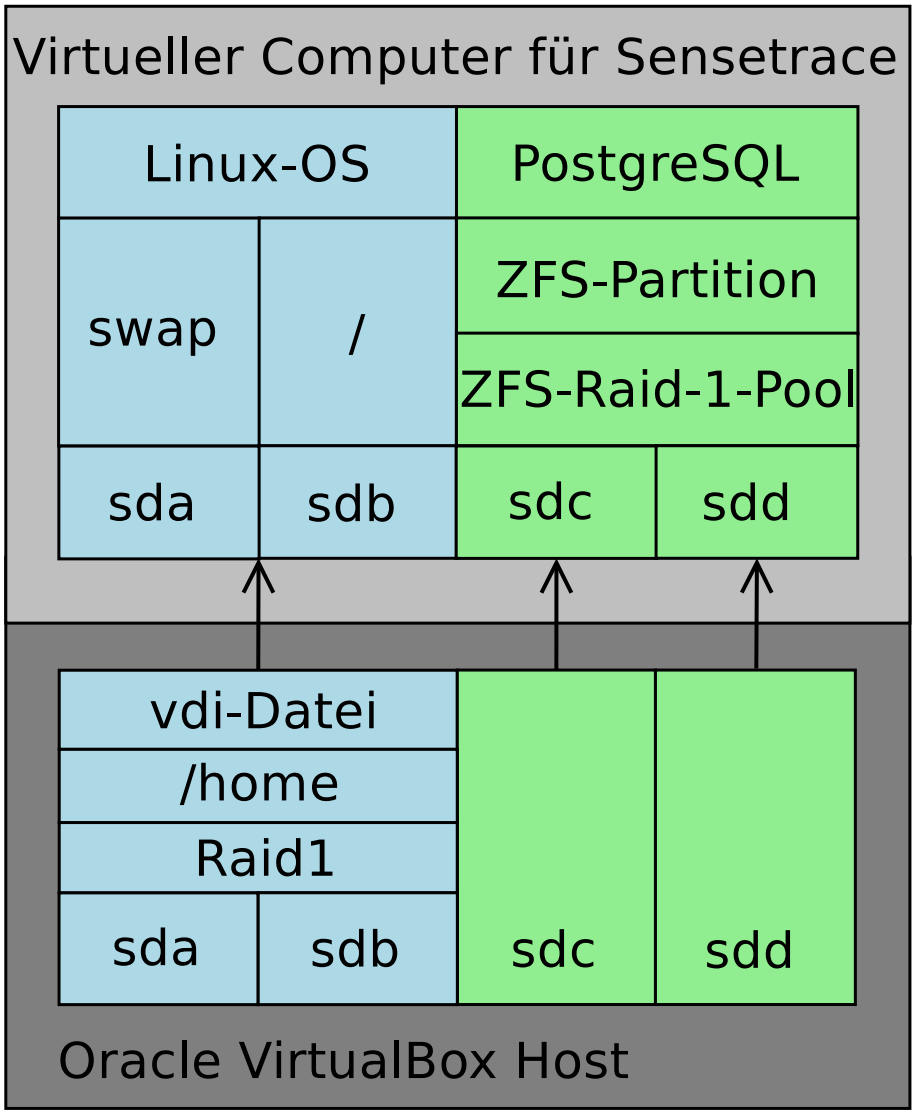

Bild 1.1: Festplattennutzung des virtuellen Servers. Der Sensetrace-Server läuft als virtuelle Maschine auf einem Oracle VirtualBox Hostserver. Der Hostserver besitzt vier Festplatten. Linux präsentiert diese Festplatten in Form sogenannter Gerätedateien, als "/dev/sda" bis "dev/sdd". Die Dateien "dev/sda" und "dev/sdb" bilden einen Raid-1-Verbund. In diesem Verbund liegt das Home-Verzeichnis "/home". Das Home-Verzeichnis beinhaltet zwei vdi-Dateien, die die virtuellen Festplatten "/dev/sda" und "/dev/sdb" im virtuellen Sensetrace-Server abbilden. Das Betriebssystem auf dem virtuellen Server benötigt "/dev/sda" für die Swap-Datei und "/dev/sdb' für das Wurzelverzeichnis "/". Zusätzlich stellt der Hostrechner zwei Festplatten in Form von Gerätedateien als "/dev/sdc" und "/dev/sdd' bereit. VirtualBox leitet diese Dateien direkt an den virtuellen Server durch. Der virtuelle Server nutzt diese beiden Dateien zum Aufbau eines ZFS-Raid-1-Pools, das eine ZFS-Partition beinhaltet. Diese Partition verwaltet die Dateien der PostgreSQL-Datenbank.

kosten. Das realisierte ZFS-Raid-1-Pool erhöht die Datensicherheit, in dem es die Daten auf beide Datenträger spiegelt. Zudem sind ZFS-Pools benutzerfreundlich zu verwalten. So ist der Austausch einer defekten Festplatten bei ZFS, wie Kapitel 1.1 beschreibt, relativ einfach zu bewerkstelligen.

Kapitel 2

Schnittstellen und Abfragen

Abbildung 2.1 zeigt die verschiedenen Schnittstellen, über die Anwendungen auf die gespeicherten Daten zugreifen können. Die PostgreSQL-Datenbank speichert neben den Messdaten, erkannte Klassifikationen und Fehler. Programme, wie zum Beispiel Matlab, können die SQL-Datenbank über die JDBC-Schnittstelle abfragen. Ein Anwender erhält mit Hilfe des grafischen Datenbankverwaltungstool pgAdmin3 Zugriff auf PostgreSQL. Abschnitt 2.1 erklärt dabei im Detail, mit welchen SQL-Befehlen ein Anwender die Daten am geschicktesten abfragt. Kapitel 2.2 begründet, warum ein SQL-Befehl immer mit "select distinct timestamp, value " erfolgen sollte. Abschnitt 2.3 beschreibt, wie ein Anwender mit dem Programm "pgAdmin III" Daten in eine CSV-Datei exportiert. Kapitel 2.4 erklärt, warum eine Abfrage von Messdaten, die ab dem Jahr 2014 aufgezeichnet wurden, schneller abläuft als die Abfrage von Messdaten aus den Jahren davor.
Zusätzlich stellt Sensetrace als Schnittstelle zur Datenabfrage den SPARQL-Server Jena Fuseki bereit. Dieser Server nimmt Anfragen über das HTTP-Protokoll entgegen. Nach Eingabe der URL "http://localhost:3030" in einem Browser gelangt der Benutzer auf eine Webseite zum Ausführen von SPARQL-Abfragen. Bei SPARQL handelt es sich um eine Sprache zum Abfragen von Daten aus einem Resource Description Framework (RDF). Die Jena Tripple Database (TDB), die als Modul Bestandteil von Fuseki ist, speichert CEP-Regeln, sowie Informationen zu Anlagen und Sensoren in RDF-Form. Sensetrace aktualisiert die Daten in dem Tripelspeicher bei jedem Start mit Hilfe der XML-Dateien, auf die Kapitel 4.1 eingeht. Ein weiterer Bestandteil von Fuseki ist das Modul D2RQ. Dieses mappt die Messdaten aus der SQL-Datenbank in das RDF-Schema von Jena Fuseki. Weitere Informationen zu D2RQ, RDF und SPARQL sind [1] zu entnehmen. Abschnitt 2.5 erklärt, wie ein

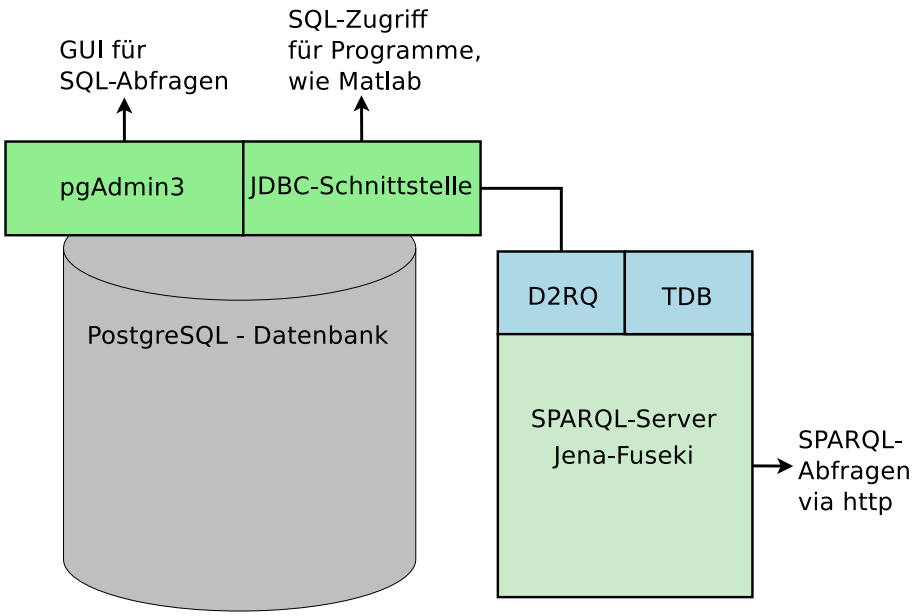

Bild 2.1: Schnittstellen zu den Daten. Die PostgreSQL-Datenbank speichert alle Messwerte, gefundene Klassifikationen und Fehler. Ein Anwender kann die Datenbank mit Hilfe des Programms pgAdmin3 abfragen. Zusätzlich stellt PostgreSQL eine JDBC-Schnittstelle bereit über die viele Programme, wie Matlab, auf die Daten zugreifen können. Neben PostgreSQL kommt der SPARQL-Server Jena Fuseki zum Einsatz, der via HTTP eine SPARQL-Schnittstelle bereitstellt. Das Modul D2RQ mappt die Daten von PostgreSQL in ein virtuelles RDF-Schema. Das TDB-Modul von Jena Fuseki ergänzt das Schema durch eine RDF-Tripelspeicher, der Informationen über Anlagen, Sensoren und CEP-Regeln enthält.

Anwender SPARQL-Abfragen genieren kann, um Messdaten abzufragen.

2.1 SQL-Abfragen

Um Daten aus der PostgreSQL-Datenbank abzufragen, sollte der Anwender verschiedene Views verwenden, die wie einfache Datenbanktabellen aussehen, hinter denen sich aber komplexe Datenbankabfragen und Funktionen verstecken, die zum Verknüpfungen von Daten aus verschiedenen Tabellen notwendig sind. Es folgt eine Übersicht über die benötigten Views.

Abfrage der Sekundendaten
Name der View: *measurement*
Spalten: *timestamp,value,value_cor,clid*

Abfrage der Minuten-Durchschnittswerte
Name der View: *avg1min*
Spalten: *timestamp,value,clid,clid_cover*

Abfrage der 15 Minuten-Durchschnittswerte
Name der View: *avg15min*
Spalten: *timestamp,value,clid,clid_cover*

Abfrage der Stunden-Durchschnittswerte
Name der View: *avg1h*
Spalten: *timestamp,value,clid,clid_cover*

Abfrage der Tages-Durchschnittswerte
Name der View: *avg1d*
Spalten: *timestamp,value,clid,clid_cover*

Abfrage der Monats-Durchschnittswerte
Name der View: *avg1month*
Spalten: *timestamp,value,value_cor,clid,clid_cover*

Abfrage der Jahres-Durchschnittswerte
Name der View: *avg1year*
Spalten: *timestamp,value,value_cor,clid,clid_cover*

Allgemein
Die Spalte *timestamp* gibt den Zeitpunkt der Aufnahme des Messwertes an. Sollte ein vom CEP-System generierter Eintrag in der Fehlertabelle vorhanden sein, enthält *value_cor* den korrigierten Wert. Enthält die Fehlertabelle keinen Vermerk, übernimmt *value_cor* den Wert von *value*. In der Spalte *value* stehen die unkorrigierten Werte, also die importieren Rawdaten. Fällt ein Messpunkt mit dem Zeitpunkt einer Klassifikation zusammen, so findet sich ein entsprechender Eintrag in der Spalte *clid*.

Beispiel
Folgende SQL-Abfrage fragt die Stunden-Durchschnittswerte des Einstrahlungssensors mit der *sensorid* = 24 ab. Den Anwender interessieren nur sonnige Tage über das Jahr 2013. Zur Ermittlung von durchgehend sonnigen Tagen existiert eine Klassifizierungsregel mit der *clid* = 30.

```
select distinct timestamp, value from "avg1h"
where sensorid=24 and timestamp>'2013-01-01 00:00:00'
and timestamp < '2014-01-01 00:00:00' and clid=30
order by timestamp;
```

2.2 Warum immer mit "select distinct" abfragen?

Der Benutzer sollte Daten immer mit "select distinct timestamp, value" abfragen. Ansonsten erscheinen Messdaten in einer Abfrage mehrfach, wenn der Benutzer keine Klassifikation durch "clid=x" angibt, und die Datenbank einem Messpunkt mehrere Klassifikationen zuordnet, wie zum Beispiel "sonniger Tag" und "stabile Einstrahlung".
Die Erweiterung "distinct" filtert identische Zeilen aus der **Ergebnismenge** heraus. Deshalb ist zu beachten, dass der Anwender nach dem "select distinct" immer "timestamp,value" angibt. Das gilt auch, wenn der Benutzer nicht an den Zeitstempeln interessiert ist. Ein "select distinct value" ohne die Spalte "timestamp" filtert nämlich gleiche Werte heraus, wodurch Messwerte verloren gehen können.
Das folgenden Beispiel löst das Problem durch eine Subabfrage. Der Befehl gibt gleiche Messwerte mehrfach aus ohne den Zeitstempel anzuzeigen.

```
select q1.value from
(select distinct value,timestamp from measurement) as q1;
```

2.3 Export von Daten in CSV-Datei

Am Effektivsten lassen sich Daten mit Hilfe des Programms "pgAdmin III" exportieren. Im folgendes Beispiel schreibt der copy-Befehl Sensordaten in die Datei "/home/pvsystem/ads/2011_Summer_G_c-Si3_1Sec.csv".

```
Copy (select distinct timestamp, value from "Rawdata" where sensorid=25
and timestamp>='2011-06-01 00:00:00' and timestamp<'2011-06-08 00:00:00'
```

```
 order by timestamp)
To '/home/pvsystem/ads/2011_Summer_G_c-Si3_1Sec.csv' With CSV;
```

Wichtig: Der Benutzer "postgres" benötigt Schreibzugriff auf das Verzeichnis, in dem der copy-Befehl die CSV-Datei erstellt.

2.4 Hinweis zur Performance vor und ab dem Jahr 2014

Das folgende Kapitel erklärt, warum Abfragen von Messdaten ab 2014 mehr als 30 mal so schnell ablaufen, wie solche von Messdaten aus den Jahren davor.
Die Messpunkte vor 2014 liegen in einer großen Tabelle. Messdaten hingegen, die Sensetrace ab Januar 2014 speichert, schreibt die Datenbank in partitionierte Tabellen. Hierbei erzeugt PostgreSQL in jedem neuen Jahr für jeden Sensor automatisch eine eigene Tabelle. Die Datenbank speichert die Daten also in vielen kleinen Tabellen ab, so dass auch die Indexdateien klein sind. Kleine Indexdateien passen in den Hauptspeicher des Rechners, wodurch Datenbankabfragen extrem beschleunigt werden.
Eine Partitionierung der alten Daten im Hintergrund ist möglich und würde vermutlich mehrere Wochen in Anspruch nehmen.

2.5 Homepage für RDF-Abfragen mit SPARQL

Zum Formulieren von SPARQL-Abfragen, muss die zu Grunde liegende RDF-Struktur bekannt sein. Die RDF-Struktur von Sensetrace erläutert die Diplomarbeit [1] auf den Seiten 33-34. Zudem sollte der Anwender mit den Grundlagen von RDF und SPARQL vertraut sein. Sensetrace umgeht dieses Problem durch eine benutzerfreundliche Homepage zum Darstellen und Abfragen von Sensordaten, die anhand der ausgewählten Sensoren und Klassifikationen eine SPARQL-Abfrage generiert. Abbildung 2.2 zeigt diese Homepage. Der Benutzer öffnet die Homepage unter der Adresse "http://localhost/index.html". Auf der Homepage wählt der Anwender gewünschte Sensoren aus der Baumstruktur auf der linken Seite. In dem darunter liegenden Feld befinden sich die Klassifikationen, die sich per Drag and Drop mit den Sensoren verknüpfen lassen. Der untere Graph zeigt die Sensorvorschau im gewählten Zeitbereich. Durch Markieren eines bestimmten Bereichs in dem Graf,

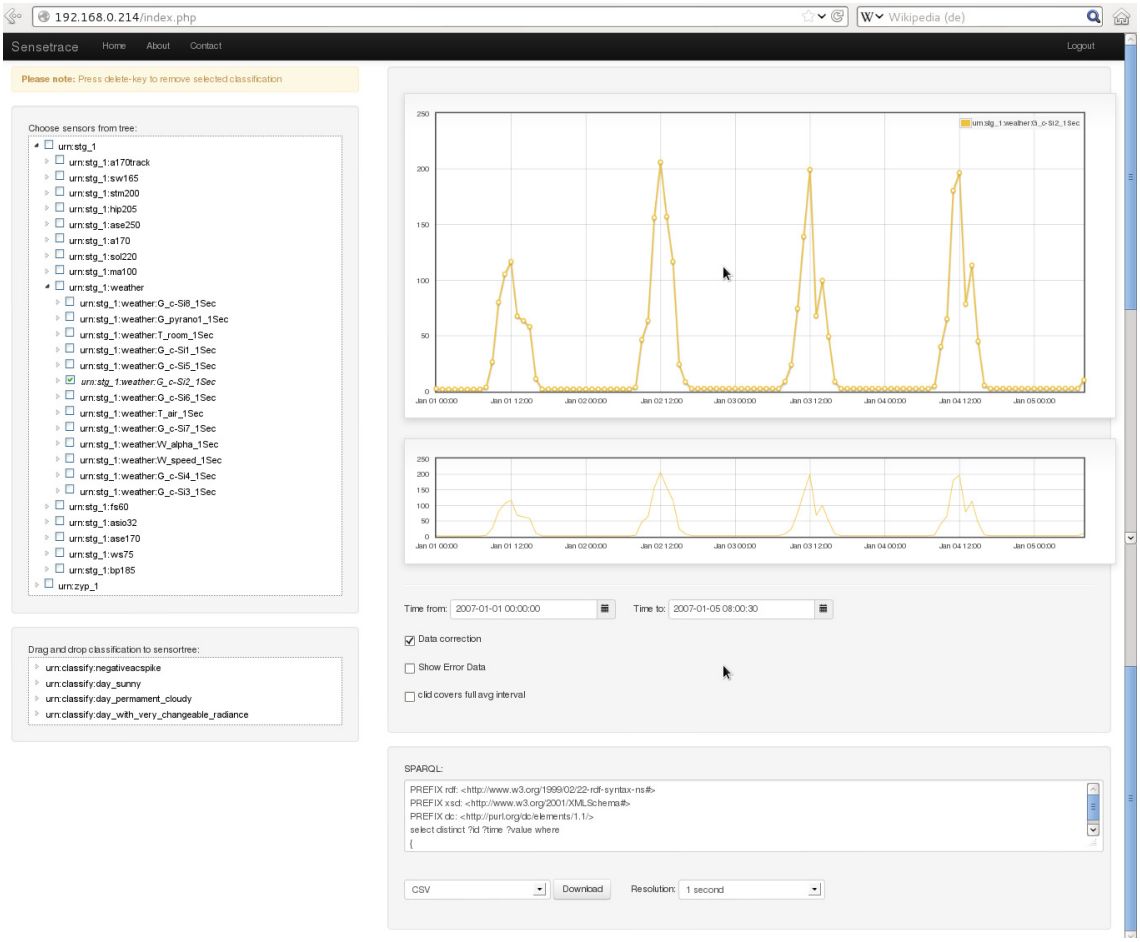

Bild 2.2: Homepage zur Datenvorschau und Abfrage. Der Anwender wählt darzustellende Sensordaten aus der Baumstruktur auf der linken Seite. In dem darunter liegenden Feld finden sich die Klassifikationen, die der Anwender per Drag and Drop mit den Sensoren verknüpfen kann. Der untere Graph zeigt die Sensorvorschau im gewählten Zeitbereich. Das Markieren eines bestimmten Bereichs in dem Graph, führt zu einer vergrößerten Darstellung der Auswahl im oberen Graphen. Zusätzlich kann der Anwender die Optionen, "Data correction", "Show Error Data" und "clid covers full avg interal" aktivieren oder deaktivieren. Die Auswahl generiert eine SPARQL-Abfrage, die das untere Feld zeigt. Der Button Download sendet die Anfrage an den SPARQL-Server und startet den Download gewünschter Messdaten.

wird die Auswahl in der oberen Grafik vergrößert dargestellt. Standardmäßig ist die Option "Data correction" aktiviert. Dadurch bezieht die Abfrage Daten aus der Fehlertabelle mit ein. Ist die Option deaktiviert zeigt die Vorschau die Daten ohne sie zu korrigieren, also so, wie sie vom Datenlogger importiert wurden. Die Option "Show Error Data" zeichnet nur die Datenpunkte aus der Fehlertabelle. Hat der Benutzer "clid covers full avg interall" ausgewählt und einen Sensor mit einer Klassifikation verknüpft, zeichnet der Graph nur die Durchschnittsmesswerte, wenn deren zu Grunde liegenden Sekundenwerte ausnahmslos in die Klassifikation fallen. Bei deaktivierter Option zeigt die Homepage Durchschnittsmesswerte als klassifiziert an, wenn auch nur ein zu Grunde liegender Sekundenwert zu der gewählten Klassifikation gehört. Anhand der Auswahl von Zeitbereich, Sensoren, Klassifikationen und Optionen, generiert das System einen SPARQL-Abfragestring, den die Homepage im unteren Feld anzeigt. Der Button Download sendet die Anfrage an den SPARQL-Server und startet den Download gewünschter Messdaten. Neben der zeitlichen Auflösung, hat der Benutzer die Möglichkeit als Format für die zu ladende Datei mit den Messwerten CSV oder JSON auszuwählen.

Kapitel 3

Programmablauf und Steuerung

Kapitel 3.1 beschreibt allgemein, wie Sensetrace, die vom Datenlogger importieren, Sensordaten analysiert und dabei sogar Durchschnittswerte und historische Daten in die Analyse einbezieht. Der Anwender steuert Sensetrace über eine OSGI-Konsole. Kapitel 3.2 gibt deshalb einen Überblick über alle Befehle, die zur Steuerung notwendig sind. Kapitel 3.3 erklärt wie der Anwender die Durchschnittswert-Berechnung von historischen Daten anstoßen kann, wenn diese nicht schon beim Import stattgefunden hat. Abschnitt 3.4 beschreibt den Messdatenimport via FTP und alternativ über ein Verzeichnis, in das der Delphin Busmanager zuvor Messdaten exportiert hat. Kapitel 3.5 erklärt, wie ein Benutzer einen virtuellen Sensordatenstrom erzeugt, um Daten neu zu klassifizieren oder nachträglich auf Fehler zu prüfen.

3.1 Ablauf einer Sensordatenanalyse

Bild 3.1 beschreibt schematisch den Ablauf einer Analyse von Messdaten. Die Abbildung setzt voraus, dass Messwerte als 1s-Rawdaten bereits in der Datenbank vorliegen. Als erstes verarbeitet das Complex Event Processing System diese Daten und speichert erkannte Fehler und Klassifikationen in einer Fehlertabelle in der Datenbank ab. Die Datenbank überdeckt mit einer View falsche Messwerte mit korrigierten oder NULL-Werten. Aus den so korrigierten Werten berechnet das System im zweiten Schritt Durchschnittswerte. Im dritten Schritt analysiert das CEP-Modul die 1m, 15, und 1h - Durchschnittswerte. Erkennt das CEP-System diesmal ein weiteres Fehlermuster, erfolgt im vierten Schritt für das Intervall des Fehlers eine Neuberechnung der Durchschnittswerte. Der 5. Schritt erfolgt analog zum 3. Schritt. Diesmal analysiert das CEP-Modul allerdings Tages- und Jahresdurchschnittswerte in einem

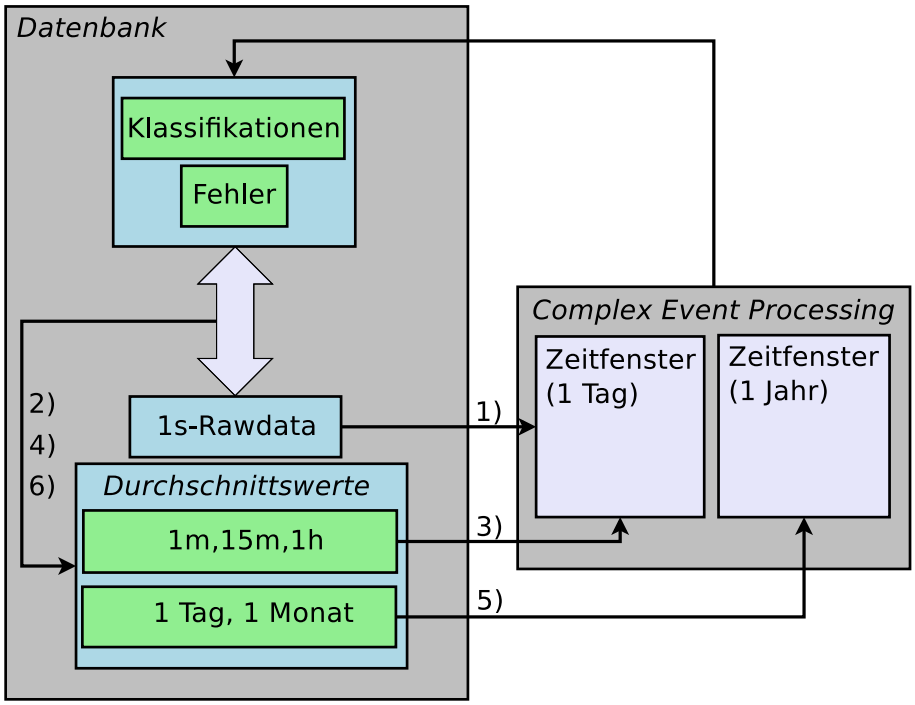

Bild 3.1: Ablaufschema bei der Analyse von Sensordaten. Die Zahlen im Bild beschreiben die verschiedenen Schritte, die das Programm durchläuft. Die 1s-Rawdaten stammen von den Datenloggern und werden unverändert in der Datenbank gespeichert. Im ersten Schritt liest das Complex Event Processing Modul diese Daten aus und analysiert sie in einem 1 Tag großen Zeitfenster auf Fehler- und Klassifikationsmuster. Die Datenbank archiviert so erkannte Klassifikationen und Fehler. Im zweiten Schritt berechnet die Datenbank aus den 1s-Rawdaten unter Berücksichtigung der Fehlerdaten Durchschnittswerte für verschiedene Intervalle. Im dritten Schritt analysiert das CEP-Modul die 1m, 15m und 1 h Durschnitsswerte. Erkennt das CEP-Modul Fehler, speichert die Datenbank diese und aktualisiert im vierten Schritt die Durchschnittswerte. Schritt 5) und 6) verlaufen äquivalent zu Schritt 3) und 4), mit dem Unterschied, dass das CEP-System diesmal Tages- und Monatswerte in einem ein Jahr großen Fenster analysiert.

ein Jahr großen Zeitfenster.

3.2 Bedienung über die Komandozeile - Befehlsübersicht

Zum Starten des Programms muss der Benutzer das Skript *"/home/pvsystem/sensetrace/export/start.sh"* ausführen. Das Skript startet Sensetrace nur, wenn keine weitere Instanz des Programms aktiv ist. Nach dem erfolgreichen Start gelangt der Benutzer zur OSGI-Konsole, über die er das Programm steuert. Dieses Kapitel gibt im Folgenden einen kurzen Überblick über die möglichen Befehle. Nach Abarbeitung eines Befehls beendet sich das Programm.

Durchschnittswerte berechnen (Abschnitt 3.3)
calc_avgs "sensoren" from "date1" to "date2" [sensorids]
Beispiel: *calc_avgs 20,21 from 2013-05-01 00:00:00 to 2013-05-02 00:00:00 28,29*

Import von den Datenloggern (Abschnitt 3.4)
Import der CSV-Dateien von den Datenloggern, die der Busmanager bereitstellt. Dabei benötigt Sensetrace Zugriff auf das Verzeichnis mit den CSV-Dateien:
import_from dl folder
Direkter Download der CSV-Dateien von den Datenloggern via ftp:
import_from dl ftp

Reklassifikation von Daten (Abschnitt 3.5)
virtual_ds from "date1" to "date2" [errorcheck oder classify] [sensorids]
Beispiel: *virtual_ds from 2009-01-01 00:00:00 to 2013-01-01 00:00:00 classify 19,24*

Hinweis: Parameter in eckigen Klammern [...] sind optional.

3.3 Durchschnittswerte bestimmen

Sensetrace bildet beim Import von einem Datenlogger automatisch die zugehörigen Durchschnittswerte. Dabei berücksichtigt die Software die Werte aus der Fehlertabelle. Die Datenbank bildet die Durchschnittswerte mit folgenden Auflösungen:

-1 Minute
-15 Minuten
-1 Stunde
-1 Tag
-1 Monat

Mit folgendem Befehl kann ein Benutzer die Durchschnittswerte von historische Messdaten berechnen, für die noch keine Durchschnittswerte vorliegen:
calc_avgs sensoren from [date1] to [date2] [sensorids]
Beispiel: *calc_avgs from 2013-05-01 00:00:00 to 2013-05-02 00:00:00 28,29*
Das Beispiel berechnet die Durchschnittswerte von den Sensoren 28 und 29 für den Tag des 01.05.2013. Das Weglassen des Parameters "sensorids" berechnet die Durchschnittswerte aller Sensoren in dem vorgegebenen Zeitintervall.

3.4 Import von den Datenloggern

Sensetrace bietet zwei Möglichkeiten die Datenlogger abzufragen. Bei der ersten Möglichkeit exportiert der Delphin Busmanager CSV-Dateien in ein Verzeichnis, aus dem Sensetrace die Daten anschließend importiert. Entsprechendes Verzeichnis bindet die Datei "/etc/sensetrace/config.cfg" in Sensetrace ein.

Der Befehl
import_from dl folder
liest die Daten aus dem Verzeichnis ein.

Als Alternative, z.B. wenn der BusManager ausfällt, ist ein direkter Download der CSV-Dateien von den Datenloggern via ftp mit dem Befehl
import_from dl ftp
möglich.

Nach dem Download stellt das Programm die Äquidistanz der Messwerte her und speichert sie in der Datenbank ab. Danach werden die importierten Daten entsprechend dem Ablauf aus Kapitel 3.1 auf Fehler analysiert und klassifiziert.
Das Programm "Gnome Scheduler" ruft täglich um 2 Uhr das Skript "/home/pvsystem/sensetrace/import.sh" auf. Dieses Shell-Skript startet einen Import der CSV-

Dateien aus dem Verzeichnis "/media/delphin/noseq", in das der Delphin Bus Manager täglich um 0 Uhr die Messdaten des letzten Tages exportiert. Das Verzeichnis "/media/delphin/noseq" ermöglicht den Zugriff auf eine Windowsfreigabe, die das Betriebssystem über die Datei "/etc/fstab" einbindet.

Wichtig: Der Import startet nur, wenn keine weitere Instanz von Sensetrace aktiv ist!

3.5 Neuklassifizierung

Nach dem Hinzufügen einer neuen Klassifizierungsregel vom Typ "classify", "help" oder "error" in die ceprule.xml-Datei nach Kapitel 4.5, soll diese Regel unter Umständen auf den historischen Datenbestand angewendet werden. Dazu ruft der Benutzer Sensetrace mit folgendem Befehl auf:

virtual_ds from "date1" to "date2" [errorcheck//classify] [sensorids]
Beispiel: *virtual_ds 2009-01-01 00:00:00 2013-01-01 00:00:00 classify 19,24*

Das Beispiel führt eine Reklassifizierung vom 1.1.2009 bis zum 1.1.2013 aus. Dabei erzeugt Sensetrace einen virtuellen Sensordatenstrom aus den Sensoren 19 und 24. Die Übergabe der Sensor-IDs ist optional. Ohne angegebene Sensor-IDs ermittelt Sensetrace automatisch die Sensoren, die die in der ceprules.xml-Datei definierten Klassifizierungsregeln benötigen.
Anstelle von "classify" lässt sich als Parameter "errorcheck" angeben. Hierbei sollte der Anwender benötigte Sensoren mit übergeben. Ansonsten prüft die Software **alle** registrierten Sensoren auf ihre Intervallgrenzen, auch wenn der Anwender keine sonstigen Regeln definiert hat. Dabei ist folgendes zu beachten:

-Bei einer Neuklassifizierung ist Vorsicht geboten, um Datenverlust zu vermeiden. Die Option "errorcheck" löscht vor einer Neuklassifizierung eventuell vorhandene Fehlerdaten für den angegebenen Zeitraum. Der Parameter "classify" löscht bereits gespeicherte Klassifikationen für das angegebene Intervall aus der Datenbank.

-Benötigen Fehler- oder Klassifizierungsregeln als Grundlage Durchschnitts-

werte von bestimmten Sensoren, muss der Benutzer diese Durchschnitte vorher mit calc_avgs aus Abschnitt 3.3 berechnen.

Kapitel 4

Konfiguration

Dieses Kapitel beschreibt, wie der Benutzer die Software über verschiedenen Dateien konfigurieren kann. Dabei stellt Abbildung 4.1 die Verzeichnisstruktur dar, in der die Konfigurationsdateien organisiert sind. Kapitel 4.1 erklärt wie der Benutzer Partitionen von Sensoren erstellt und diese einer Solaranlage zuordnet. Kapitel 4.2 veranschaulicht, wie sich Sensoren in das System integrieren lassen, und gibt eine Übersicht der dafür erforderlichen Parameter.
Die Datei "config.cfg" im Verzeichnis "/etc/sensetrace/" umfasst allgemeine Parameter, wie Verzeichnisse und Zugangsdaten zur Datenbank. Kapitel 4.3 geht im Detail auf diese Parameter ein.
Über die Datei "ceprules.xml" legt der Anwender Regeln fest, die Fehler im Datenstrom erkennen und Daten klassifizieren. Zur Definition solcher Regeln kommt die Etalis Event Language zum Einsatz. Kapitel 4.4 gibt eine Einführung in diese Beschreibungssprache. Kapitel 4.5 zeigt anhand eines Beispiels wie der Benutzer CEP-Regeln über die Datei "ceprules.xml' einbindet und erklärt die verschiedenen Parameter.

4.1 Erstellen und Verknüpfen von Anlagen, Partitionen und Sensoren

Die beiden Solaranlagen in Stuttgart und auf Zypern bestehen aus mehreren kleineren Anlagen verschiedener Modulhersteller. Solche Teilanlagen werden innerhalb der Software als Partitionen bezeichnet. Zu einer Partition gehört eine Gruppe von Sensoren. Der Zusammenhang von Anlagen, Partitionen und Sensoren lässt sich somit

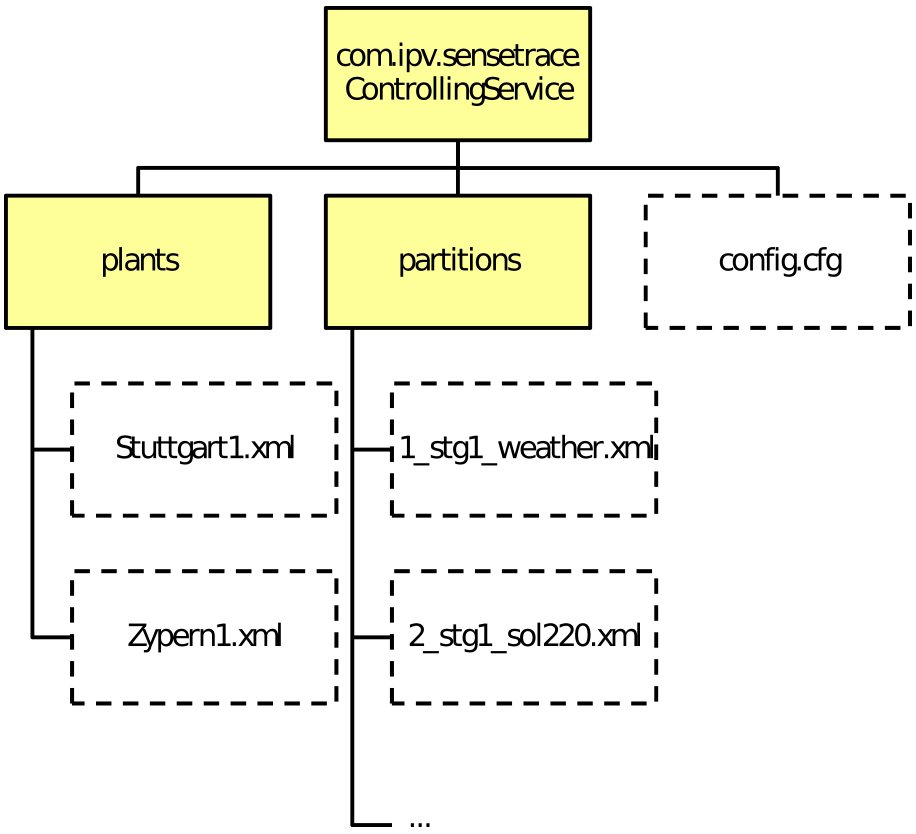

Bild 4.1: Organisation der Konfigurationsdateien. Das Verzeichnis "/etc/sensetrace" beinhaltet alle für Sensetrace erforderlichen Konfigurationsdateien. Das Verzeichnis "plants" beinhaltet zwei XML-Dateien im SensorML-Format zur Beschreibung der beiden Solaranlagen in Stuttgart und auf Zypern. Jede Solaranlage besteht wiederum aus mehreren Partitionen. Jede Partition braucht eine eigene XML-Datei im Ordner "partitions". Die config.cfg im Wurzelverzeichnis beinhaltet alle Parameter, die für die Software erforderlich sind.

kurz als

$$Anlage \overset{owns}{\Longrightarrow} Partitionen \overset{owns}{\Longrightarrow} Sensoren$$

darstellen. Die XML-Dateien im SensorML-Format zum Einbinden von Anlangen, Partitionen und Sensoren sind nach dieser Darstellung organisiert. Die XML-Dateien im Ordner "plants" beschreiben die Solaranlagen. Derzeit ist eine Anlage in Stuttgart und eine in Zypern vorhanden. Abbildung 4.1 stellt die beiden XML-Dateien im Verzeichnis "plants" dar. Jede Solaranlage (plants) setzt sich aus mehreren kleineren Anlagen (partitions) zusammen. Der Ordner "partitions" enthält für jede Partition eine XML-Datei. Diese XML-Dateien beschreiben neben der jeweiligen Partitionen die Sensoren, die zu ihr gehören.
Der Abschnitt im Folgenden einen Auszug aus einer XML-Datei, die eine Partition beschreibt. Sie realisiert die Verknüpfung $Partitionen \overset{owns}{\Longrightarrow} Sensoren$.

```
<SensorML
...
version="1.0">
 <System id="urn:stg_1:weather">
  <outputs>
   <OutputList>
        <output name="weatherMeasurements">
     <swe:DataGroup>
          <swe:component name="G_c-Si1_1Sec">
          <swe:Quantity definition="G_c-Si1_1Sec" uom="urn:ogc:def:
             unit:WperSm" sensorid="23" creationdate="2006-05-31"
             decimalplace="0" type="avg-1s" lowerLimit="-10"
             upperLimit="1225" mysqlid="23" />
               ...
      </swe:component>
        </swe:DataGroup>
       </output>
      </OutputList>
  </outputs>
 <!--================================== -->
 <!-- System Communication Interfaces -->
 <!--================================== -->
 <interfaces>
```

```
  <InterfaceList>
   <connection>
    <Link>
     <source ref="192.168.1.102/media/m01/mem064/memory.mem" />
     <destination ref="ftp" />
    </Link>
   </connection>
  </InterfaceList>
 </interfaces>
</System>
</SensorML>
```

Der dargestellte Auszug aus einer XML-Datei aus dem Ordner *partitions* beschreibt die Wetter-Partition der Solaranlage in Stuttgart. Diese umfasst mehrere Sensoren, die meteorologische Daten erfassen. Die Partition trägt als System-ID $urn : stg\_1 : weather$. Um das Beispiel kurz zu halten, beschreibt der Abschnitt $< swe : component >$ für diese Partition nur einen Strahlungssensor.
Folgender Auszug stammt aus einer XML-Datei im Ordner "plants" und zeigt, wie die Verknüpfung $Anlage \overset{owns}{\Longrightarrow} Partitionen$ erfolgt.

```
<SensorML
...
version="1.0">
 ...
 <System id="urn:stg_1">
  <swe:component name="urn:stg_1:weather">
  </swe:component>
  ...
 </System>
</SensorML>
```

Dieser Auszug aus der SensorML-Datei im Ordner plants beschreibt die Solaranlage in Stuttgart. Sie trägt den Bezeichner mit der System-ID $urn : stg\_1$. Der Abschnitt $< swe : component >$ weist der Solaranlage neben weiteren, aber hier nicht dargestellten Komponenten, die im vorigen Beispiel definierte Wetterpartition zu.

4.2 Sensorbeschreibungen

Wie in Abbildung 4.1 dargestellt, wird jede Anlage durch jeweils eine XML-Datei im Ordner "partitions" verwaltet. Diese Dateien erhalten zu einer Anlage Beschreibungen von Sensoren im standardisierten SensorML-Format. Folgender Auszug aus einer XML-Datei gehört zur Partition, die Sensoren zur Wetteraufzeichnung umfasst.

```
...
<!--================== -->
<!-- Station Outputs -->
<!--================== -->
<outputs>
 <OutputList>
  <output name="weatherMeasurements">
   <swe:DataGroup>
    <swe:component name="G_c-Si2_1Sec">
     <swe:Quantity definition="G_c-Si2_1Sec"
     uom="urn:ogc:def:unit:WperSm" sensorid="23" creationdate=
     "2006-05-31" decimalplace="0" type="avg-1s" lowerLimit="-10"
     upperLimit="1225" differencetopreviousvalue="3" />
    </swe:component>
    ...
   </swe:DataGroup>
  </output>
 </OutputList>
</outputs>
...
```

Das Beispiel zeigt die Definition eines Einstrahlungssensors. Der Benutzer trägt jeden Sensor, wie in Kapitel 4.1 beschrieben, als Element vom Typ $< swe : component >$ im Abschnitt $< swe : DataGroup >$ ein. Zur Beschreibung des Sensors sind mehrere Attribute notwendig.
Das Attribut $< swe : componentname >$ weist dem Sensor einen in der Partition eindeutigen Namen zu. Die Homepage zur Darstellung und Abfrage aus Kapitel 2.5 verwendet dieses Attribut zur Bezeichnung eines Sensors. Deshalb macht es Sinn erklärende und keine allzu kryptischen Namen zu wählen. Der Einstrahlungssensor aus dem Beispiel trägt die Bezeichnung G_cSi2_1Sec.

Das Attribut *definition* bestimmt, die im Datenlogger hinterlegte Bezeichnung für den Sensor.
Das Element *uom* definiert die Einheit der physikalischen Größe, die der Sensor aufzeichnet. Der Sensor im Beispiel misst die physikalische Einheit $\frac{W}{m^2}$.
Die *sensorid* legt fest unter welcher ID die PostreSQL-Datenbank die Messdaten des Sensors verwaltet.
Die Attribute *lowerlimit* und *upperlimit* definieren das Intervall, in dem die gemessene Daten gültig sind. Daten außerhalb dieses Intervalls markiert Sensetrace in der Datenbanktabelle *Errortable* durch den Wert *null* als undefiniert.
Das Attribut *differencetopreviousvalue* bestimmt, um viel Prozent sich ein Messwert zum vorhergehenden ändern muss, damit er zur Analyse an Jtalis gesendet wird. Der Parameter erlaubt eine drastische Reduzierung der zu analysierende Datenmenge. Je nach erforderlicher Analysegenauigkeit sollte dieser Wert möglichst groß sein.
Der Parameter *creationdate* bezeichnet das Datum, an dem der Sensor installiert wurde.
Das Attribut *decimalplace* gibt die gewünschte Genauigkeit des Sensors an. Sowie die Attribute *type* und *creationdate* hat *decimalplace* allerdings keinen Einfluss auf das Systemverhalten. Es dient ausschließlich zur Information eines Anwenders, der in der XML-Datei liest. Möglicherweise benötigt Sensetrace genannte Attribute in zukünftigen Versionen.

Weitere Attribute:
active="false": Deaktiviert Sensoren, die nicht außer Betrieb sind. Sensetrace ignoriert Sensoren mit diesem Flag beim Import vom Datenlogger.
statictest="false": Die Fehleranalyse betrachtet den Sensor nicht auf Einhaltung seiner Intervallgrenzen. Dieses Attribut macht zum Beispiel für die AC-Leistungssensoren Sinn, da hier oft Spikes über dem Maximum auftreten, die Jtalis durch Regeln erkennt und herausfiltert.

Achtung: Auf keinen Fall darf der Anwender das Attribut *sensorid* im Nachhinein ändern! Alle anderen Attribute können problemlos (aber mit Vorsicht) geändert werden, auch wenn schon Daten vorliegen.

4.3 Systemparameter

Dieses Kapitel beschreibt die Parameter, die zur allgemeinen Konfiguration des Programms erforderlich sind. Diese Parameter setzt der Anwender in der Datei config.cfg im Verzeichis "/etc/sensetrace". Es folgt eine stichwortartige Übersicht der Parameter:
$ceprulesfile$: Pfad des Verzeichnisses, in dem die Datei ceprules.xml liegt. Diese Datei beinhaltet die CEP-Regeln.

$plantfile$: Pfad des Verzeichnisses, in dem die XML-Dateien zur Definition der Plants nach Kapitel 4.1 liegen.

$sensormlfiles$: Pfad des Verzeichnisses, in dem die XML-Dateien zur Definition der Partitionen der Plants, wie in Kapitel 4.1 erklärt, liegen.

$xsltfolder$: Das Verzeichnis mit den xslt-Dateien. Sensetrace benötigt diese Dateien zum Einlesen und Konvertieren der XML-Dateien (Für den Anwender uninteressant).

$pgsqlcs, pgsqluser, pgsqlpwd$: Daten zum Anmelden an der PostgreSQL-Datenbank.

$memfiles$: Verzeichnis, in dem Sensetrace die mem-Dateien bei einem direkten FTP-Download vom Datenlogger speichert.

$memconvert$: Pfad zur converter.exe. Dieses Programm wandelt die per FTP geladenen mem-Dateien in CSV-Dateien um.

$RDFServer$: URL zum Jena-RDF Server in der Form $http://127.0.0.1:3030/combined$

$dataloggerdlfolder$: Im Standardmodus lädt Sensetrace neue Messwerte in Form von CSV-Dateien aus einem Verzeichnis. Der Parameter gibt entsprechenden Pfad an.

$number\_of\_files$: Gibt an , wie viele Dateien in $dataloggerdlfolder$ zu erwarten sind. Beträgt das Ergebnis "Anzahl Dateien im Verzeichnis" mod $number\_of\_files$ ungleich Null, geht Sensetrace von einem Datenloggerproblem aus und informiert

den Benutzer per Mail.

mail_to : Setzt Mailadressen, an die Sensetrace Status- und Fehlermails schickt.

mail_from : Email-Absender des Servers zum Verschicken einer Mail.

4.4 Einführung in die Etalis Event Language

Die Etalis Event Language (ELE) dient der Beschreibung von Mustern, die im Datenstrom erkannt werden sollen. Im Prinzip stellt Etalis in Prolog geschriebene Funktionen bereit, die der Anwender zum Erstellen von Abfragen einbezieht. Aus diesem Grund beginnt dieses Kapitel mit einer kurzen Einführung in Prolog bevor es auf ELE eingeht.
Das Kapitel endet mit der beispielhaften Darstellung einer Regel, die einen Sensorausfall auf dem Datenstrom erkennt.

Die Programmiersprache Prolog
Prolog gehört zur Klasse der deklarativen Programmiersprachen. Das bedeutet, dass der Prolog-Programmierer ein Problem anhand von Regeln und Fakten beschreibt. Prolog findet dann über Graphen-Algorithmen automatisch einen Lösungsweg für das definierte Problem.
Die Formulierung

```
Sensor(sensor1, time1, time2, value)
```

zur Definition eines Sensors in Etalis wäre ein solches Faktum. Dieses definiert einen eingehenden Messwert von $sensor1$ mit undefiniertem Wert $value$ und im undefinierten Zeitintervall $[time1, time2]$.
Eine einfache Regel wäre

```
neuermesswert(time1,time2):=Sensor(sensor1, time1, time2, value);
Sensor(sensor2, time1, time2, value)
```

Dabei ist := als "genau-dann-wenn" zu interpretieren. Der Operator ; liest sich als "oder". Deshalb tritt die Regel $neuermesswert(time1, time2)$ in Kraft, sobald das Faktum $Sensor(sensor1, time1, time2, value)$ oder $Sensor(sensor2, time1, time2, value)$

auftritt. Zum Schluss folgt ein kurzer Überblick der wichtigsten Operatoren von Prolog.

Logische Operationen
logisches oder: ;
logisches und: ,

Vergleiche
kleiner: $<$
größer: $>$
kleiner gleich: $<=$
größer gleich: $>=$
muster vergleich: $==$, Bsp.: $(3 == (1 + 2) => false$
numerischer Vergleich: $=:=$, Bsp.: $(3 =:= (1 + 2) => true)$
ungleich: $= \setminus =$

Arithmetik
Addition: $+$
Subtraktion: $-$
Skalarprodukt: $*$
Division: $/$
Modulo: mod

Temporale Operatoren von ELE
ELE erweitert Prolog um Operatoren, die die Reihenfolge eintreffender Ereignisse in einer Datenstromabfrage berücksichtigen. Abbildung 4.2 gibt eine Übersicht über die wichtigsten Operatoren. Weitere Operatoren und zusätzliche Beschreibungen sind dem Etalis Manual zu entnehmen. P_1, P_2 und P_3 repräsentieren drei unabhängige Ereignisse. P_1 SEQ P_3 löst ein Ereignis aus, wenn P_3 nach P_1 aktiv wird. Die Regel P_2 AND P_3 löst ein Ereignis aus, wenn P_2 und P_3 zeitgleich auftreten oder direkt aneinander grenzen. P_1 PAR P_2 hingegen löst nur ein Ereignis aus, sobald sich die beiden Ereignisse zeitlich überschneiden. P_2 OR P_3 ist aktiv, wenn eines der beiden Ereignisse auftritt. P_1 DURING P_2 ist wahr, solange P_1 in P_2 liegt. P_2 MEETS P_3 ist aktiv, wenn P_2 und P_3 aneinandergrenzen, aber sich nicht überschneiden.

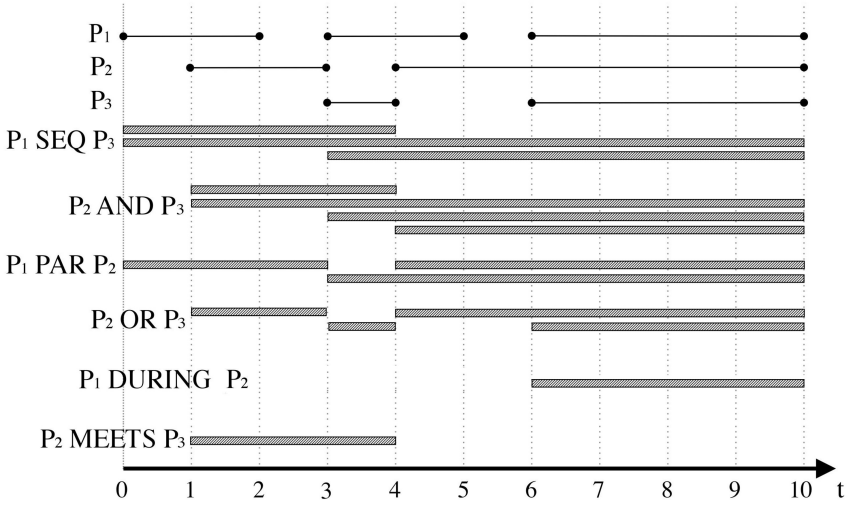

Bild 4.2: Temporale Vergleichsmöglichkeiten in ELE. P_1, P_2 und P_3 definieren drei Ereignisse, die in einem 10 Sekunden großen Zeitfenster auftreten und von unterschiedlicher Dauer sind. Die dickeren Balken zeigen weitere Ereignisse. Diese sind eine Folge verschiedener Kombinationen von P und temporalen Operatoren.

4.4.1 Beispiel zur Erkennung eines Einstrahlungs-Sensorausfalls.

Das folgende Beispiel zeigt wie der Anwender mit ELE eine Regel definiert, die den Ausfall eines Einstrahlungssensors erkennt.

```
window=300:stableradiation(timefrom1, timefrom2):= sensor('1
   sec_Sensor24',timefrom1,timeto1,value1) par
sensor('1sec_Sensor25',timefrom2,timeto2,value2) par sensor('1
   sec_Sensor26',timefrom3,timeto3,value3)
where ((timeto1-timefrom1)>60,(timeto2-timefrom2)>60, (timeto3-
   timefrom3)>60)
```

Die Regel $stableradiation(timefrom1, timefrom2)$ sucht nach einem Intervall im 1 Sekunden-Datenstrom, in dem alle drei Einstrahlungssensoren gleichzeitig stabile Einstrahlungswerte aufweisen. Die *where*-Bedingung legt fest, dass die Einstrahlung aller Sensoren für mindestens 60 Sekunden stabil sein muss. Die Fenstergröße

$window = 300$ begrenzt das Intervall von $stableradiation(value1, value2, value3)$ auf 300 Sekunden.

4.5 Parameter für CEP-Regeln

Das folgende Beispiel erklärt die Parameter, die bei der Erstellung von CEP-Regeln zur Anwendung kommen.

```
<swe:component name="acspike">
  <swe:Quantity definition="acspike" window="60" type="error"
        replacesensor="19" creationdate="2013-09-03" active="true"
       ceprule="acspike(Time1-1,Time2+10,interpolate)&lt;
       -sensor('1sec_Sensor19',Value1,Time1,Time2) where (Value1&gt
          ;2100)" />
</swe:component>
```

Das Beispiel zeigt die Regel "acspike". Diese Regel sucht nach Fehler-Spikes im Datenstrom von Sensor 19. Solche Spikes treten auf, wenn der Wert 2100 überschreitet und damit weit über dem oberen Maximalwert dieses Sensors liegt. Der Ausdruck

```
ceprule="acspike(Time1-1,Time2+10,interpolate)&lt;
-sensor('1sec_Sensor19',Value1,Time1,Time2) where (Value1&gt;2100)"
```

definiert diesen Zusammenhang. Sensetrace lädt die Regel bei Klassifizierungen, wenn active="true" ist. Durch ein active="false" lässt sich die Regel temporär deaktivieren. Der Parameter type="error" definiert den Ausdruck als Fehlerregel. Erkennt das CEP-System die Anwendbarkeit der Regel auf dem Datenstrom im Zeitintervall $[Time1, Time2]$, erstellt die Datenbank für jede Sekunde im Intervall $[Time1-1, Time2+10]$ einen Eintrag in der Fehlertabelle für Sensor 19 (replacesensor="19"). Der Parameter *replacementsensor* ist nur für Regeln vom $type = error$ gültig. Dabei berechnet Sensetrace die korrigierten Messwerte so, dass die fehlerhaften Rawdaten linear interpoliert werden. Der Anwender bestimmt das Verhalten durch den Wert $'interpolate'$. Die Angabe eines konstanten Wertes an Stelle von $'interpolate'$ ist auch möglich. Der Parameter $window = 60$ begrenzt das Intervall $Time2 - Time1$ von $sensor('1sec\_Sensor19', Value1, Time1, Time2)$ auf 60 Sekunden.
Neben den Fehler-Regeln, definiert durch type="error", existieren außerdem Klassifizierungsregeln (type="classify") und Hilfsregeln (type="helper"). Bei einer Klas-

sifizierungsregel darf der Benutzer im definierenden Faktum keinen Wert, sondern nur ein Zeitintervall in der Form $stableradiation(timefrom, timeto)$ angeben. **Eine Klassifikation besitzt nämlich keinen Wert.** Zusätzlich muss der Anwender die ID der Klassifikation setzen, zum Beispiel $clid = 10$.
Hilfsregeln bilden die Grundlage für andere Regeln. Hier spielt das Aussehen des definierenden Faktums keine Rolle.
Der Ausdruck $'1sec\_Sensor19'$ im Faktum $sensor('1sec\_Sensor19', Value1, Time1, Time2)$ zieht zur Analyse die Sekundendaten vom Sensor mit der Id=19 heran. Zur Analyse der Minuten-Durchschnittsdaten dieses Faktums, muss das Faktum $sensor('1min\_Sensor19', Value1, Time1, Time2)$ lauten. Entsprechend sind weitere Ausdrücke möglich:

-Für 15 Minuten-Durchschnittswerte: $'15min\_Sensor19'$
-Für 1 Stunden-Durchschnittswerte: $'1hour\_Sensor19'$
-Für 1 Tag: $'1day\_Sensor19'$
-Für 1 Monat: $'1month\_Sensor19'$

Abbildungsverzeichnis

Literaturverzeichnis

[1] H. Adler, *Langzeitüberwachung von Photovoltaikanlagen*. Uni Stuttgart, TU Hamburg-Harburg.