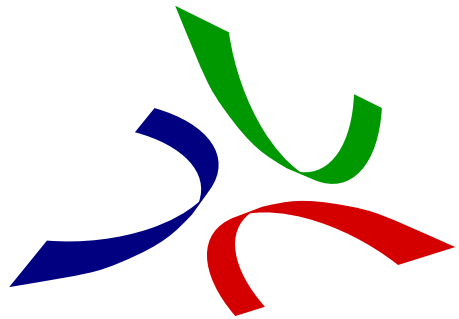
Kolloquium zur Bachelorarbeit

Entwicklung einer Datenbanklesekomponente für Datenberechnungsnetzwerke im Kontext des Flow-Based Programming Frameworks

Michael Eckel

Fachhochschule Gießen-Friedberg
Fachbereich Mathematik, Naturwissenschaften und Informatik

24. August 2009



Inhalt

Inhalt

Inhalt

Inhalt

Inhalt

Inhalt I

Das Projekt PARASUITE

- **P**roduct **A**nalysis and **R**eporting **A**pplication **SUITE**
- Forschungsprojekt
  - Cognidata GmbH
  - Fachhochschule Gießen-Friedberg
  - Philipps-Universität Marburg
- Entscheidungsunterstützendes System (*Decision Support System*)
- . . . im Bereich Produktlebenszyklus
  - Beginning-of-Life
  - Middle-of-Life
  - End-of-Life

Kunden von PARASUITE

Bombardier Transportation

- Hersteller von Lokomotiven
- Unterstützung bei *Beginning-of-Life*
- Bessere Produkte herstellen
- Schlechte Teile ausfindig machen
- Fehlerquellen finden

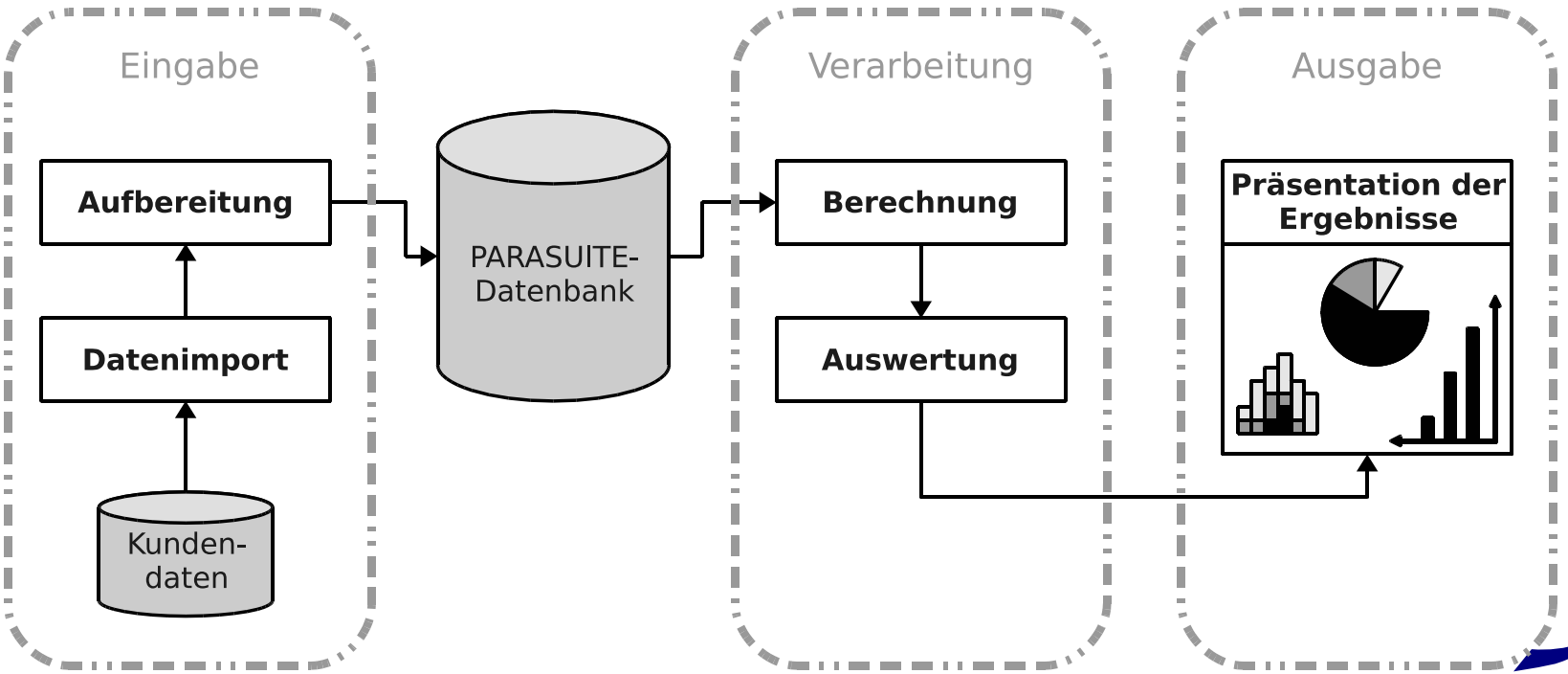
Kunden von PARASUITE

ThyssenKrupp Aufzüge

- Hersteller von Aufzügen
- Unterstützung bei *Middle-of-Life*
- Vorausschauende Produktwartung (*Predictive Maintainance*)
- Planen von Wartungsarbeiten
- Wann fällt wo welches Teil aus?

Wie arbeitet PARASUITE?

- Kunden sammeln Daten
  - Zustandsdaten von Komponenten zum Fehlerzeitpunkt
  - ...
- PARASUITE wertet diese Daten aus



Inhalt I

Inhalt II

Inhalt I

Inhalt II

Datenmodell – Was soll gespeichert werden?

- Produkte
- Meldungen zu den Produkten

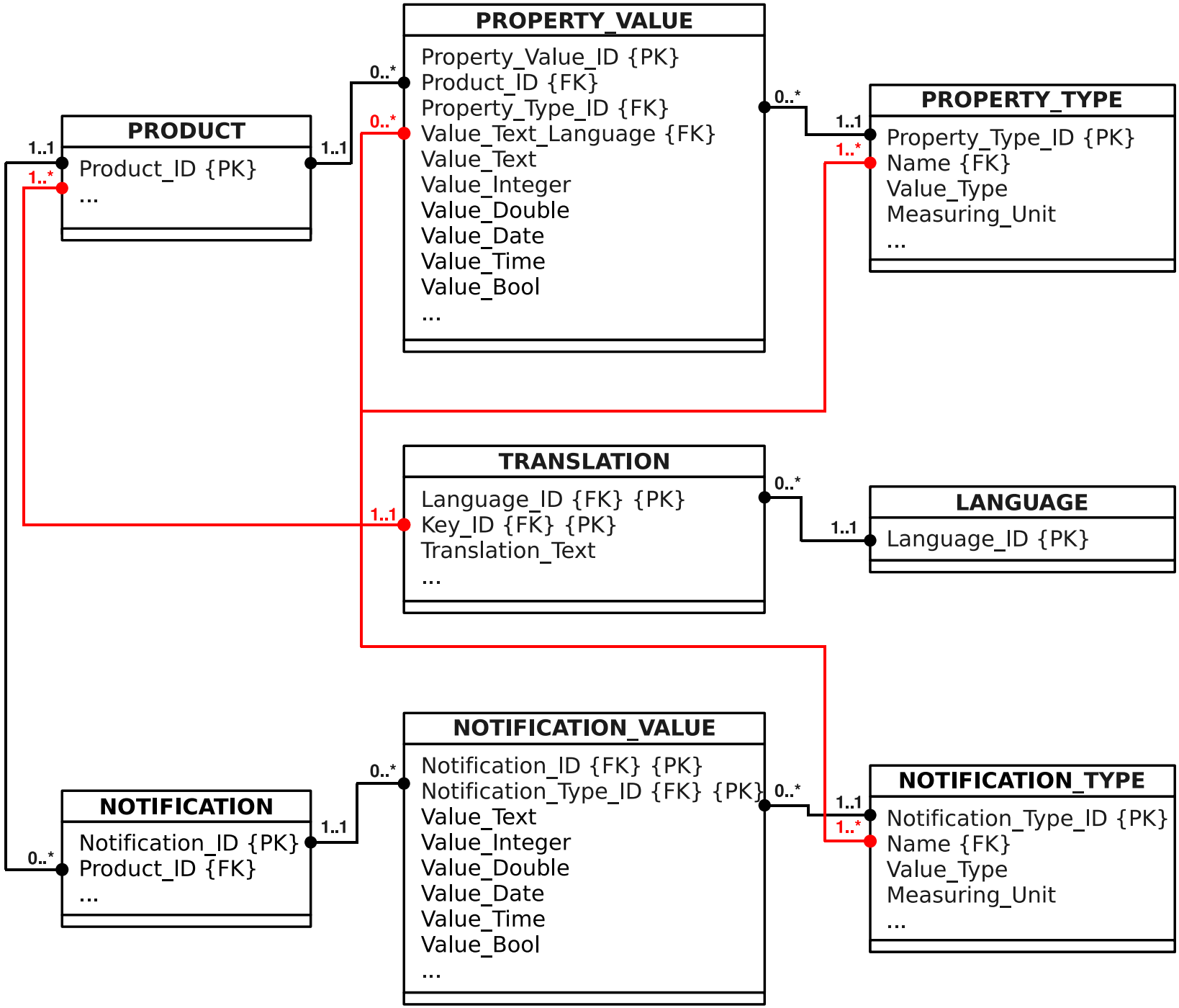
- Lokalisierung soll unterstützt werden

Inhalt I

Inhalt II

Das alte Datenbankschema

- Generisches Datenbankschema
- Zerlegung einer normalen Tabelle in ihre Bestandteile:
  - Tabelleninformationen → `PRODUCT`
  - Spalteninformationen → `PROPERTY_TYPE`
  - Werte → `PROPERTY_VALUE`
- Flexible Speicherung
- Keine Änderung am Datenbankschema nötig
- Lokalisierung mittels MD5-Hash-Werten


PROPERTY_VALUE
Property_Value_ID {PK}
Product_ID {FK}
Property_Type_ID {FK}
Value_Text_Language {FK}
Value_Text
Value_Integer
Value_Double
Value_Date
Value_Time
Value_Bool
...
PRODUCT
Product_ID {PK}
...
PROPERTY_TYPE
Property_Type_ID {PK}
Name {FK}
Value_Type
Measuring_Unit
...
TRANSLATION
Language_ID {FK} {PK}
Key_ID {FK} {PK}
Translation_Text
...
LANGUAGE
Language_ID {PK}
NOTIFICATION_VALUE
Notification_ID {FK} {PK}
Notification_Type_ID {FK} {PK}
Value_Text
Value_Integer
Value_Double
Value_Date
Value_Time
Value_Bool
...
NOTIFICATION
Notification_ID {PK}
Product_ID {FK}
...
NOTIFICATION_TYPE
Notification_Type_ID {PK}
Name {FK}
Value_Type
Measuring_Unit
...
0..*
1..1
1..*

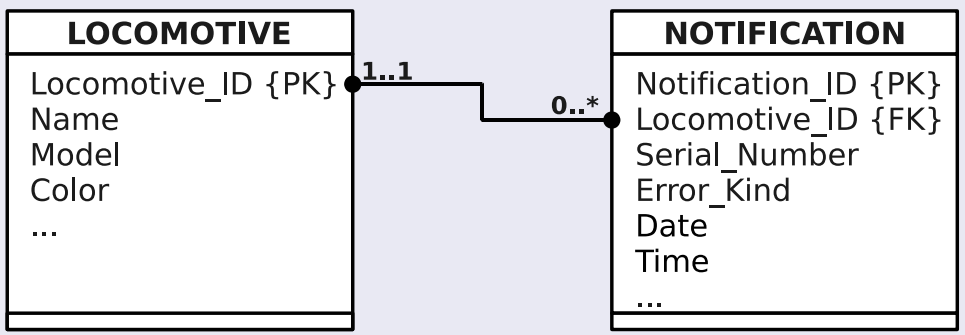
Inhalt I

Inhalt II

Das neue Datenbankschema

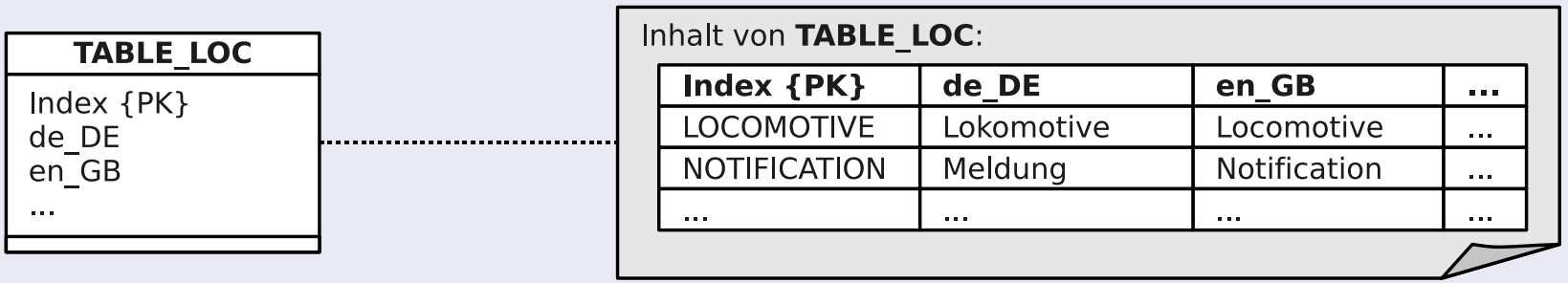
- Generatives Datenbankschema
- Tabellen auf gewohnte Weise anlegen
- Keine Einschränkungen zu beachten

Kundenspezifische Tabellen



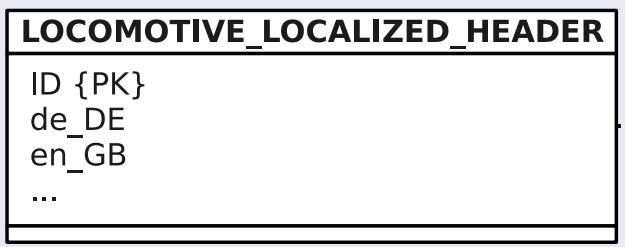
Lokalisierung im neuen Datenbankschema

Lokalisierte Tabellennamen



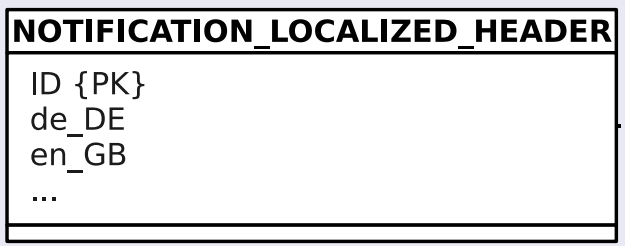
Lokalisierung im neuen Datenbankschema

Lokalisierte Tabellenspalten



Inhalt von **LOCOMOTIVE_LOCALIZED_HEADER**:

ID {PK}	**de_DE**	**en_GB**	**...**
Locomotive_ID	Lokomitiven-ID	Locomotive ID	...
Name	Name	Name	...
Model	Modell	Model	...
Color	Farbe	Colour	...

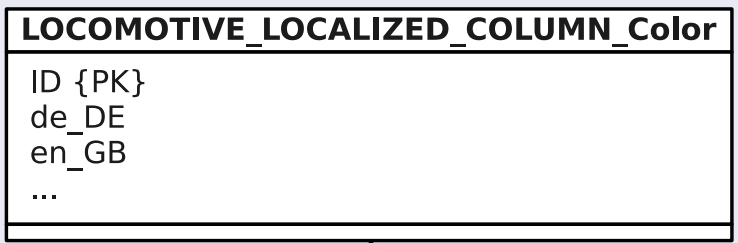


Inhalt von **NOTIFICATION_LOCALIZED_HEADER**:

ID {PK}	**de_DE**	**en_GB**	**...**
Notification_ID	Meldungs-ID	Notification ID	...
Locomotive_ID	Lokomotiven-ID	Locomotive ID	...
Serial_Number	Seriennummer	Serial Number	...
Error_Kind	Fehlerart	Kind of Error	...
Date	Datum	Date	...
Time	Uhrzeit	Time	...

Lokalisierung im neuen Datenbankschema

Lokalisierte Tabellenspalten



Inhalt von **LOCOMOTIVE_LOCALIZED_COLUMN_Color**:

ID {PK}	de_DE	en_GB	...
red	rot	red	...
blue	blau	blue	...
gray	grau	grey	...
green	grün	green	...
yellow	gelb	yellow	...

Inhalt I

Inhalt II

Vergleich der Datenbankschemen

	Datenbankschema	
	alt	**neu**
Generizität	++	−−
Performanz	−−	++
Handhabung	−	++
Lokalisierung	+	++

Inhalt I

Inhalt II

Datenbankschnittstellen in PARASUITE

ParasuiteEntityDAOBean

- Arbeitet auf dem neuen Datenbankschema
- Erlaubt generische Zugriffe
- Stellt grundlegende Funktionen zur Verfügung
- Datenabfrage nicht sehr performant

Datenbankschnittstellen in PARASUITE

ParasuiteEntityDAOBean

- Arbeitet auf dem neuen Datenbankschema
- Erlaubt generische Zugriffe
- Stellt grundlegende Funktionen zur Verfügung
- Datenabfrage nicht sehr performant

Value List Handler

- Datenabfrage performanter
- Realisiert *Paging*-Prinzip

Inhalt I

Inhalt I

Warum Datenvorberechnungen?

- Sehr große Datenmengen
- Lange Berechnungszeiten
- Aufteilung in zwei Schritte
  - Vorberechnungen
  - Auswertungen
- Vorberechnungen werden realisiert mit Flow-Based Programming

Inhalt I

Flow-Based Programming (FBP)

Was ist Flow Based Programming?

- Programmierparadigma
- Erfinder: John Paul Morrison
- Eigenschaften
  - Netzwerke von „black box“-Prozessen
  - Prozess = Komponente
  - Verbindungen zwischen diesen Prozessen
- Framework für Java und C#.NET verfügbar

Bestandteile von FBP

FBP-Komponente I

- Für Verarbeitung zuständig
- Erfüllt meist eine spezielle Aufgabe
- In JavaFBP als Thread realisiert
- Kann beliebig viele *Ports* besitzen
- Port-Typen:
  - Input Port
  - Output Port
  - Array Input Port
  - Array Output Port

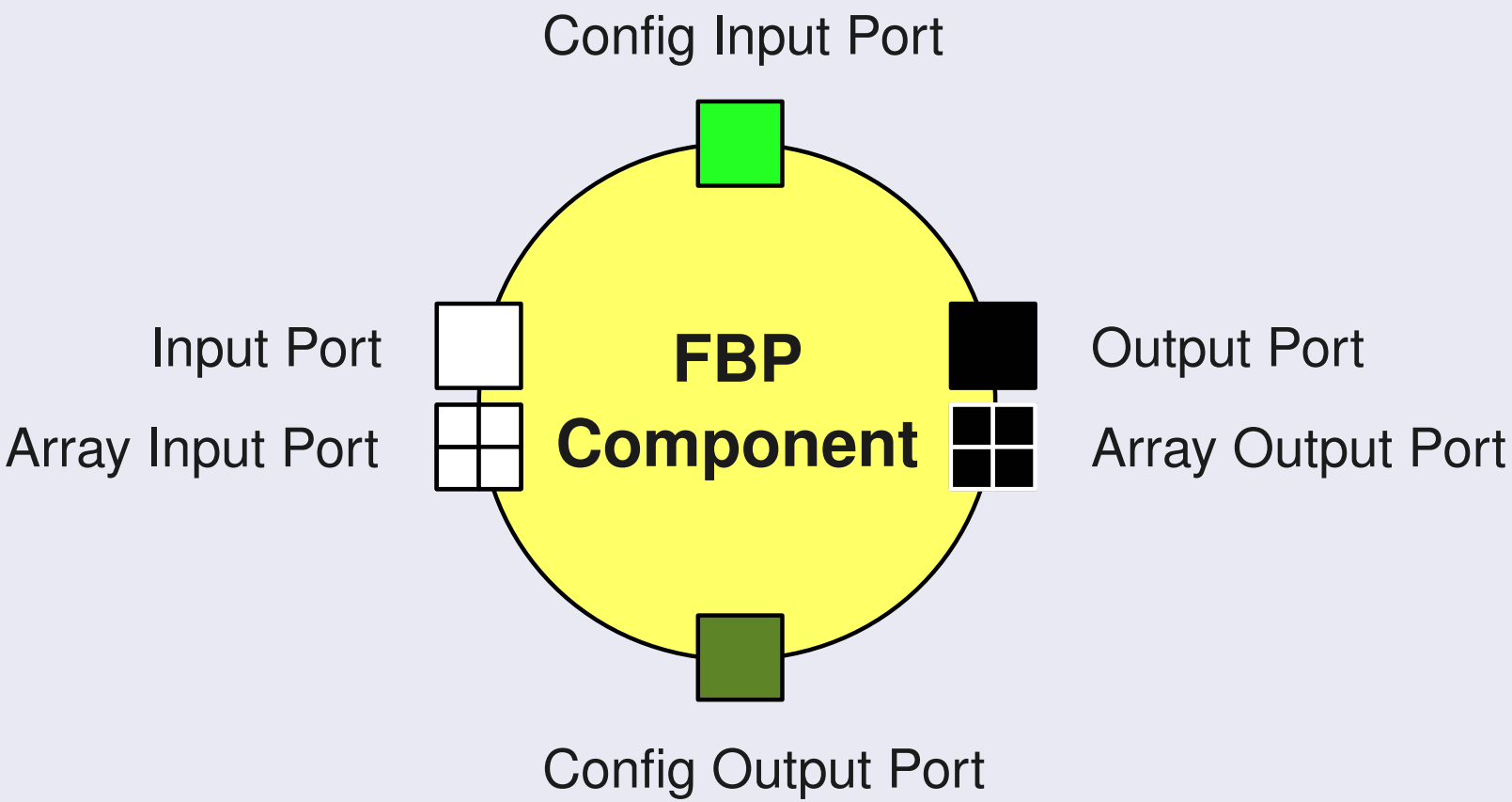
Bestandteile von FBP

FBP-Komponente I

- Für Verarbeitung zuständig
- Erfüllt meist eine spezielle Aufgabe
- In JavaFBP als Thread realisiert
- Kann beliebig viele *Ports* besitzen
- Port-Typen:
  - Input Port
  - Output Port
  - Array Input Port
  - Array Output Port
  - Config Input Port (von PARASUITE eingeführt)
  - Config Output Port (von PARASUITE eingeführt)

Bestandteile von FBP

Komponente (auch: Knoten) II



Bestandteile von FBP

Verbindung

- Gemeinsam genutzte Datenstruktur
- Ähnlich einer `BlockingQueue` in Java
- Verbindet einen Ausgangsport mit Eingangsport
- Uni-direktional (vgl. UNIX Pipe)
- Transportiert Informationspakete *Information Packet*

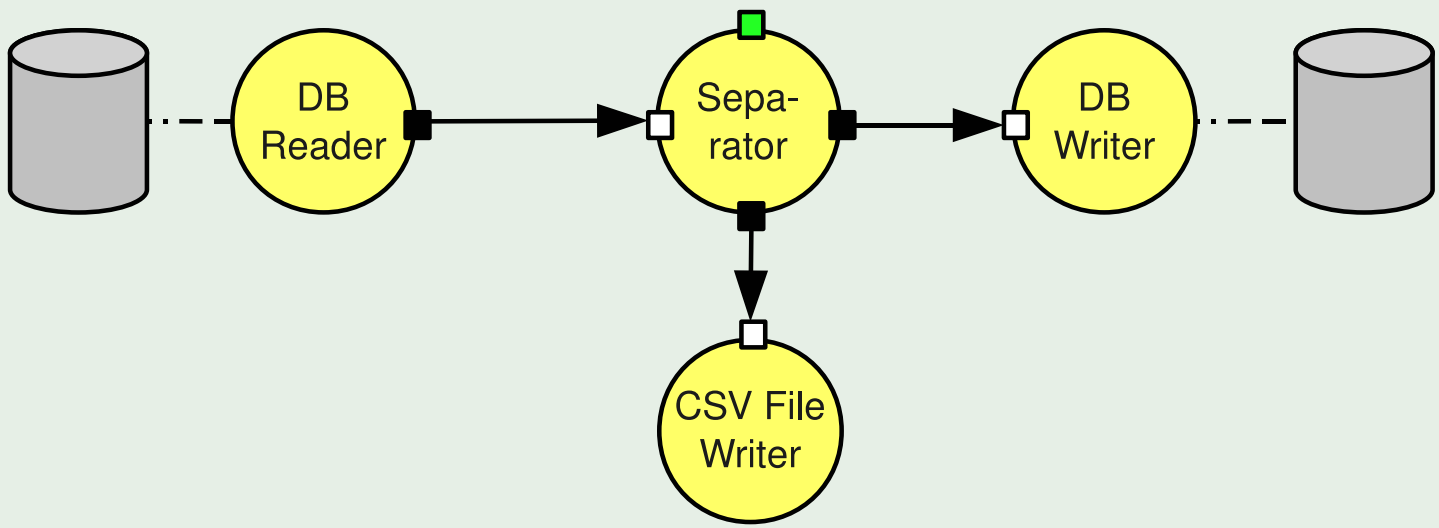
Bestandteile von FBP

Subnetz

- Besteht aus Komponenten und Verbindungen
- Hat eigene Ports
- Verhält sich wie eine normale FBP-Komponente
- Beliebig tiefe Schachtelung möglich
- Modularisierung, Wiederverwendbarkeit, Abstraktion

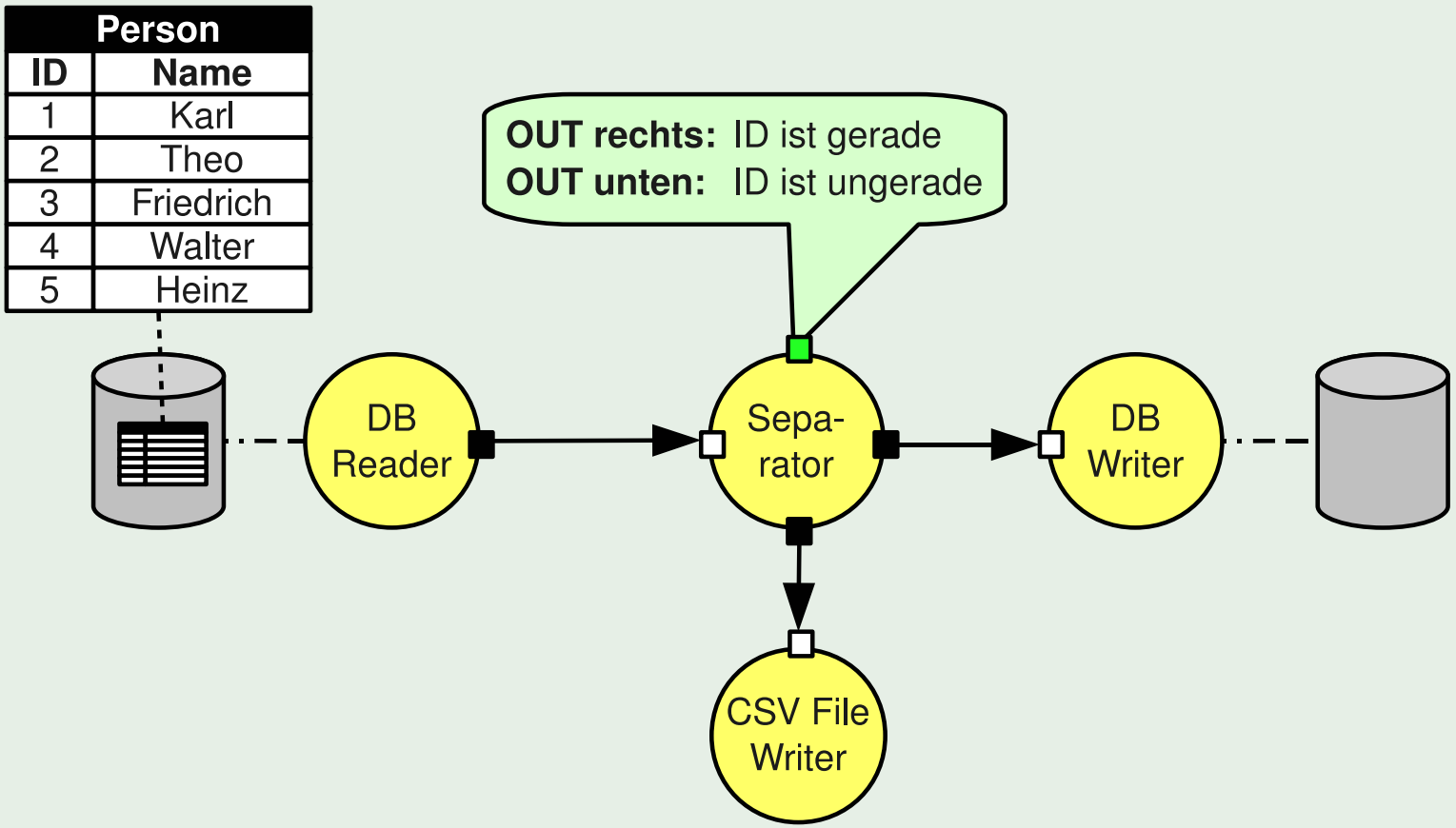
Bestandteile von FBP

Beispiel eines FBP-Netzwerks



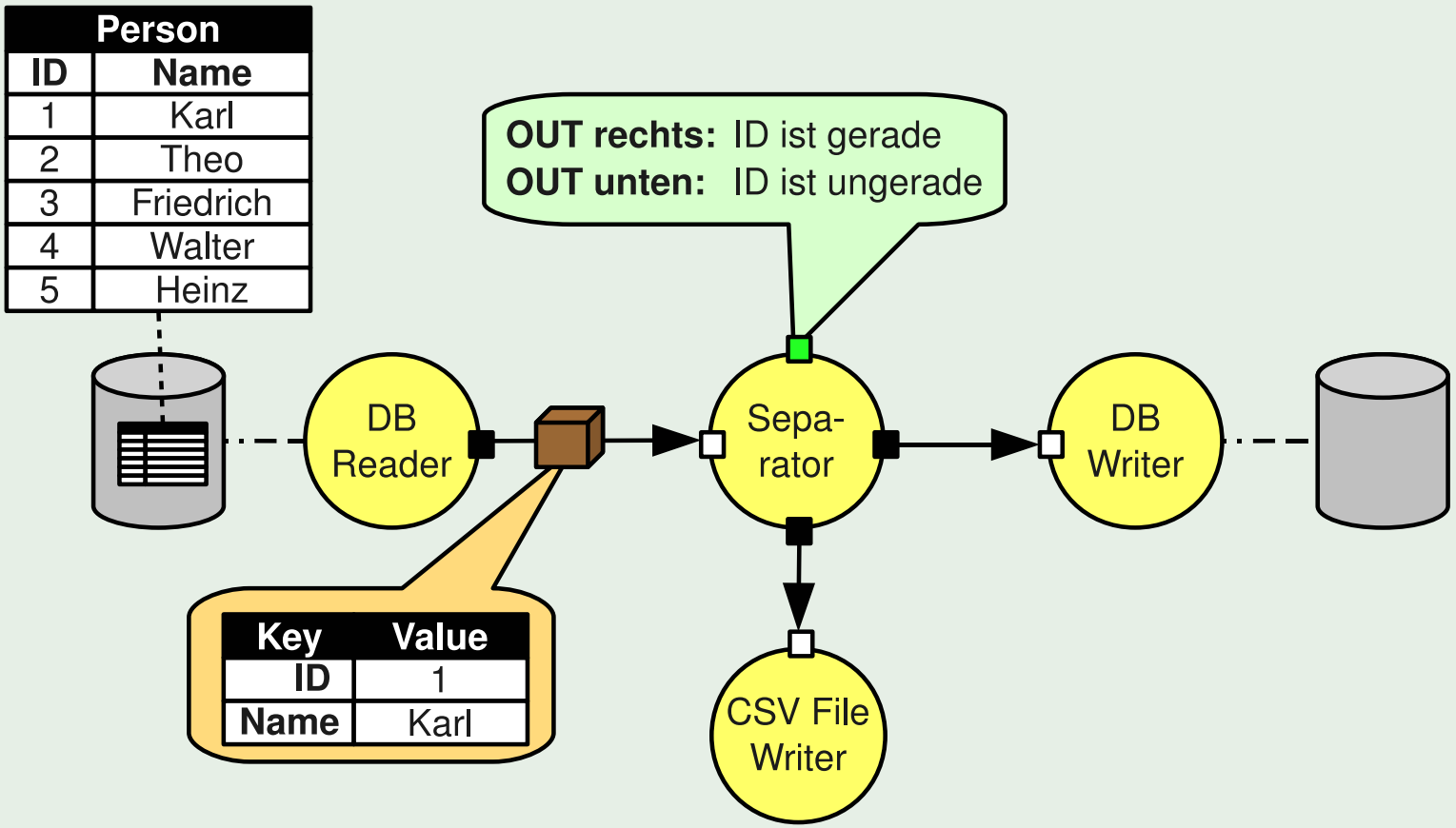
Bestandteile von FBP

Beispiel eines FBP-Netzwerks



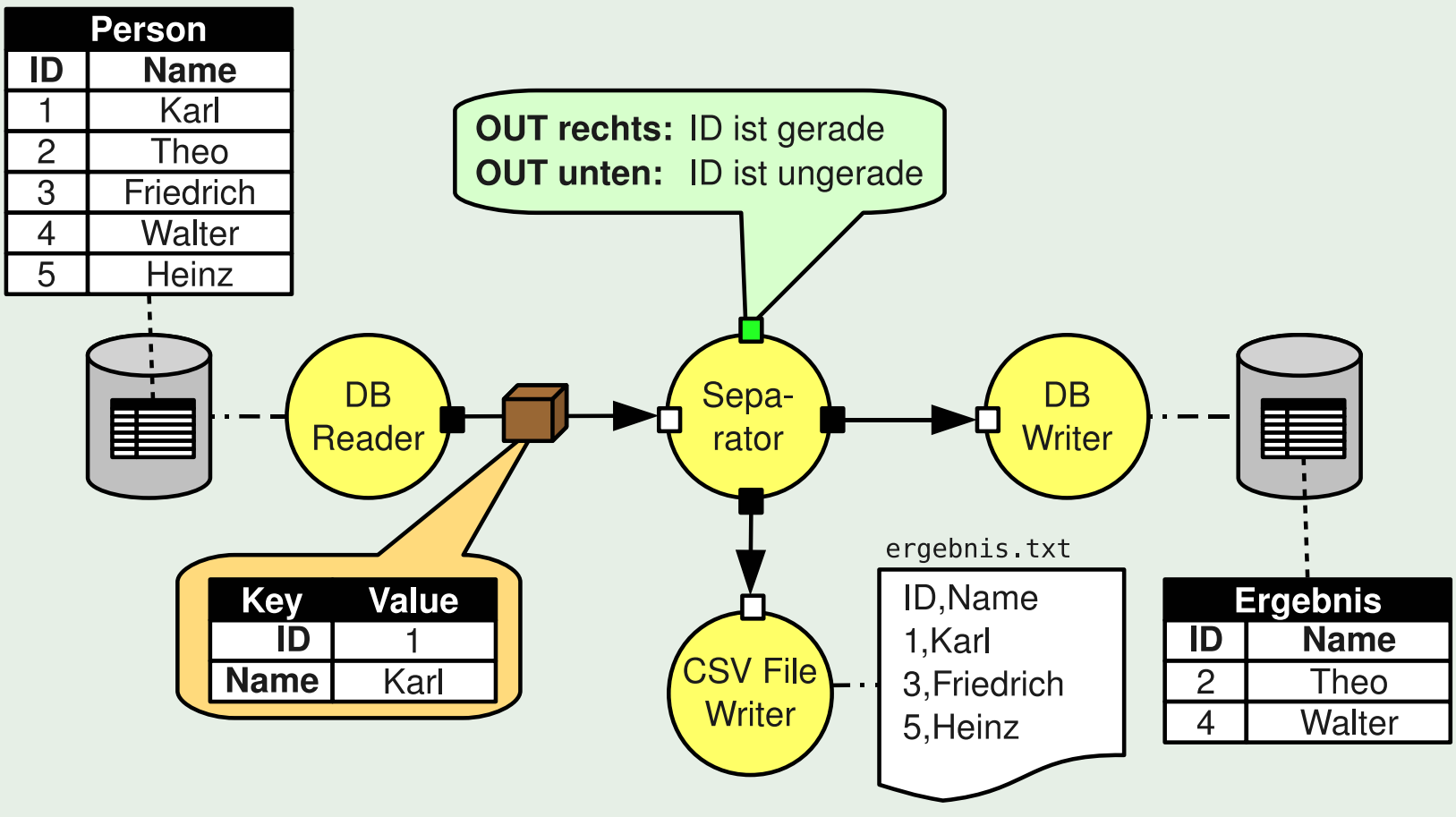
Bestandteile von FBP

Beispiel eines FBP-Netzwerks



Bestandteile von FBP

Beispiel eines FBP-Netzwerks



FBP-Netzwerke erstellen

FBP-Netzwerk in Java-Code

```
...
producer = component("producer",
    PacketProducer.class);

consumer = component("consumer",
    PacketConsumer.class);

connect("producer.OUT", "consumer.IN");

initialize({ "PK1", "PK2", "PK3" },
    "producer.CONFIG");
...
```

FBP-Netzwerke erstellen

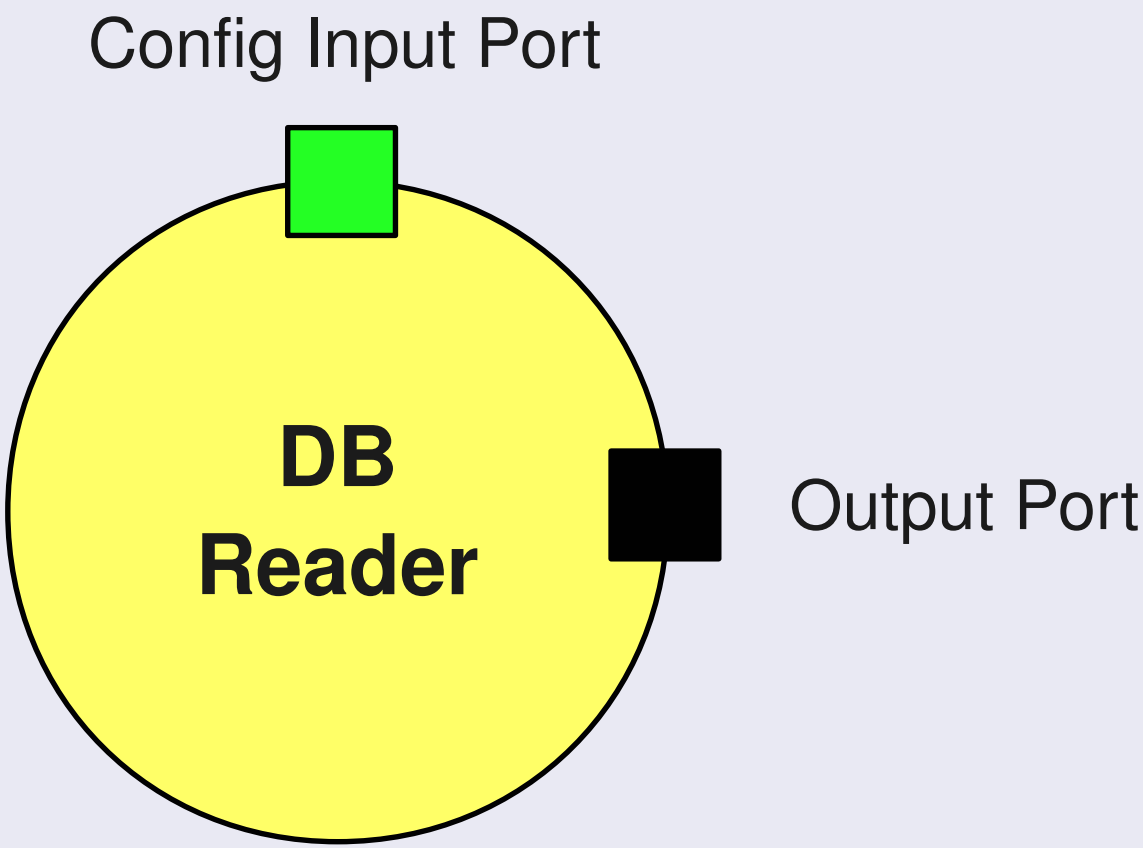
FBP-Netzwerk in XML

```
<component type='PacketProducer' name='producer'/>
<component type='PacketConsumer' name='consumer'/>
<connection sender='producer' sourcePort='OUT'
    receiver='consumer' destinationPort='IN'/>
<init component='producer' port='CONFIG'>
  <parameter name='VALUES'>
    <list type='STRING'>
      <item>PK1</item>
      <item>PK2</item>
      <item>PK3</item>
    </list>
  </parameter>
</init>
```

Inhalt I

Datenbanklesekomponente

Layout



Datenbanklesekomponente

Arbeitsweise

1. Lesen der Konfiguration vom Konfigurationsport
2. Verarbeiten der Konfiguration
3. Schließen des Konfigurationsports
4. Hauptverarbeitungsschleife
   Ende-Bedingung: alle Datensätze sind gelesen
   1. Lesen eines Datensatzes aus der Datenbank
   2. Datensatz transformieren und in ein Informationspaket stecken
   3. Informationspaket über den Ausgangsport senden
5. Schließen des Ausgangsports

Inhalt I

Konfiguration der Datenbanklesekomponente

- **Anforderungen:** textuell, menschenlesbar

Konfiguration der Datenbanklesekomponente

- **Anforderungen:** textuell, menschenlesbar

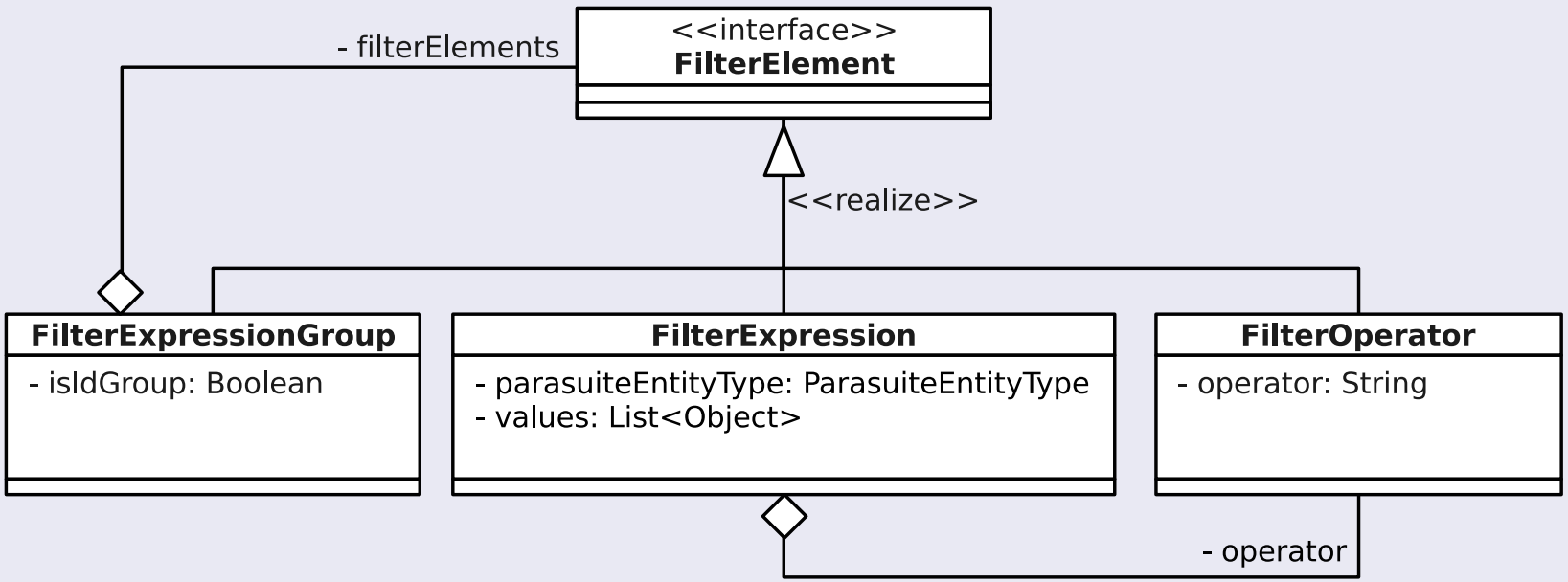
Konfigurationsparameter

Name	Funktion	SQL-Äquivalent
`LOCALE`	Angabe der Lokalisierung	–
`TABLE`	auszulesende Tabelle	`FROM`
`COLUMNS`	auszulesende(r) Spaltenname(n)	`SELECT`
`FILTER`	Datenfilter	`WHERE`
`ORDER`	Sortierung	`ORDER BY`
`DISTINCT`	Duplikate oder nicht	`DISTINCT`

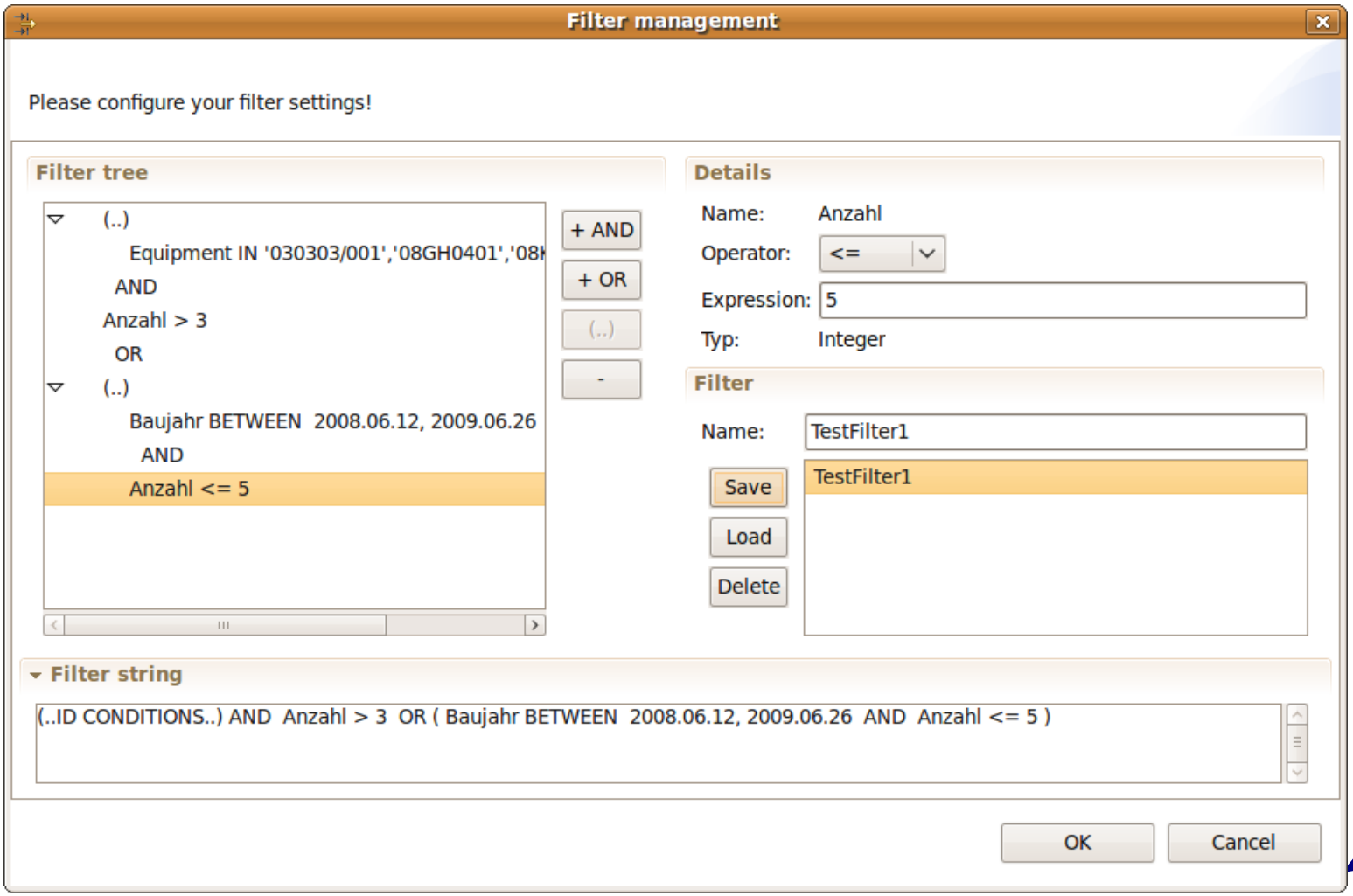
Inhalt I

Filter-Konfiguration

Datenstruktur für Filterausdrücke



Filter-Konfiguration



Filter-Konfiguration

Serialisierung mittels BASE-64

Client: Filterausdruck binär serialisieren und in BASE-64 umwandeln

Server: Rückwandlung und Deserialisierung

- **Pro**
  - Textbasiert
  - Einfach
  - Funktioniert
- **Contra**
  - Debugging (so gut wie) unmöglich

Fazit: keine gute Lösung

Filter-Konfiguration

Serialisierung mittels SQL

Client: Filterausdruck in SQL umwandeln

Server: SQL parsen

- Wäre eine gute Lösung
- Aber: SQL enthält nicht genügend Informationen
  - `isIdGroup`-Feld nicht vorhanden
  - SQL-Code enthält Unterabfragen
- Umgehung:
  - `isIdGroup` als SQL-Kommentar
  - Für Unterabfragen: Datenbank direkt ansprechen

Fazit: keine akzeptable Lösung

Filter-Konfiguration

Serialisierung mittels PFEL

Client: Filterausdruck in die eigene Sprache PFEL (**P**ARASUITE **F**ilter **E**xpression **L**anguage) umwandeln

Server: PFEL parsen

- **Pro**
  - Textbasiert
  - Gut lesbar
  - Funktioniert
- **Contra**
  - Enthält nicht alle Informationen des `FilterExpression`-Objekts, ABER: können aus DB geholt werden
  - zu Testzwecken nicht so gut

Fazit: akzeptable, funktionierende Lösung

PFEL

Beispiel PFEL (pretty-printed)

```
(
  {
    IDENTIFIER[LOCALE=de_DE](PERSON.Id)
    OPERATOR(IN)
    VALUES[TYPE=INTEGER](
      10023, 147389, 3974299
    )
  }
  OPERATOR_LOGICAL(AND)
  (
    ...
  )
)
```

Auszüge aus der PFEL Grammatik

Tokens

```
   <BRACKET> : "(" | ")" | "{" | "}"

     <COMMA> : ","

<IDENTIFIER> : "IDENTIFIER[LOCALE=" [a-z]{2} "_"
               [A-Z]{2} "](" [A-Za-z_]* "."
               [A-Za-z_]* ")"

  <OPERATOR> : "OPERATOR(" ( "<=" | ">=" | "=" |
               "!=" | "<" | ">" | ... ) ")"

      <TIME> : [0-9]{2} ":" [0-9]{2} ":" [0-9]{2}
```

Auszüge aus der PFEL Grammatik

Grammatik (BNF)

```
      Value ::= <VALUE_TEXT> | <VALUE_INTEGER> ...

     Values ::= Value | Value "," Values

 ValueList ::= "VALUE_LIST[TYPE=...](" Values ")"

FilterExpression ::= "{" <IDENTIFIER> <OPERATOR>
                     ValueList "}"
...
```

Realisierung von PFEL

- Selbstgeschriebene Komponenten
  - PFELScanner
  - PFELParser
  - PFELPrinter
- **Alternative:** Parser-Generator, z. B. JavaCC oder ANTLR
- Für kleine Sprachen – wie PFEL – ist ein selbstgeschriebener Parser völlig in Ordnung
  - übersichtlicher
  - verständlicher
  - besser wartbar

Filter-Konfiguration

Serialisierung mittels XML

Client: Filterausdruck in XML umwandeln

Server: XML parsen

- **Pro**
  - Textbasiert
  - Gut lesbar
  - Funktioniert
  - Enthält alle Informationen
- **Contra**
  - Viel Overhead

Fazit: gute Lösung

Filterausdruck in XML (1)

```
<FilterExpressionGroup>
  <IsIdGroup>true</IsIdGroup>
  <FilterExpression>
    <Locale>de_DE</Locale>
    <ParasuiteEntityType>
      <Id>Locomotive_ID</Id>
      <LocalizedName>Lokomotiven-ID</LocalizedName>
      <Locale>de_DE</Locale>
      <ParasuiteEntity>
        <Id>LOCOMOTIVE</Id>
        <LocalizedName>Lokomotive</LocalizedName>
        <Locale>de_DE</Locale>
      </ParasuiteEntity>
...
```

Filterausdruck in XML (2)

```
...
      <IdentifierType>true</IdentifierType>
      <DataType>VALUE_TEXT</DataType>
      <MeasuringUnit isNull="true"></MeasuringUnit>
    </ParasuiteEntityType>
    <FilterOperator type="compare">
      IN
    </FilterOperator>
    <ValueList type="VALUE_TEXT">
      <Value>021100101-1</Value>
    </ValueList>
  </FilterExpression>
</FilterExpressionGroup>
```

Inhalt I

Ausblick

- Generische Lesekomponente
  - Lesen aus Datei, URL, DB, Arbeitsspeicher, ...
- Generische Schreibkomponente
  - Schreiben in Datei, URL, DB, Arbeitsspeicher, ...

Ausblick – JavaFBP ändern

Annotationen an Klassenvariablen anstatt an der Klasse

```
@InPort("IN")
@OutPort("OUT")
public class SampleFBPComponent extends Component {
  private InputPort inputPort = null;
  private OutputPort outputPort = null;
  ...
  @Override
  protected void openPorts() {
    this.inputPort = super.openInput("IN");
    this.outputPort = super.openOutput("OUT");
  }
}
```

Ausblick – JavaFBP

Annotationen an Klassenvariablen anstatt an der Klasse

```
public class SampleFBPComponent extends Component {
  @InPort("IN")
  private InputPort inputPort = null;

  @OutPort("OUT")
  private OutputPort outputPort = null;

  ...
}
```

Ausblick – JavaFBP

Callback-Methoden für Ports

```
...
/* Hauptverarbeitungsschleife */
@Override
protected void execute() {
  /* Aktiv den Port abfragen */
  while (inputPort1.read()) {
    ...
  }
}
...
```

Literatur

J. Paul Morrison
Flow-Based Programming.
van Nostrand Reinhold, 1994.

Sven Steinseifer
Evaluation and Extension of an Implementation of Flow-Based Programming.
Fachhochschule Gießen-Friedberg, Masterarbeit, 2009.

Prof. Dr. Th. Letschert
Programmiersprachen – Konzepte und Realisationen.
Fachhochschule Gießen-Friedberg, 2008.

...

Fragen?

Danke für die Aufmerksamkeit!