Projektphase bei Cognidata

Michael Eckel

Fachhochschule Gießen-Friedberg
Fachbereich Mathematik, Naturwissenschaften und Informatik

7. Oktober 2013

Inhalt

Inhalt

Inhalt

Inhalt

Inhalt

Projekt „PARASUITE“

- „PARASUITE“
  - **P**roduct **A**nalysis and **R**eporting **A**pplication **SUITE**
- Forschungsprojekt
  - Cognidata GmbH (`http://www.cognidata.de/`)
  - FH Gießen
  - Uni Marburg

Projekt „PARASUITE“

- „PARASUITE“
  - **P**roduct **A**nalysis and **R**eporting **A**pplication **SUITE**
- Forschungsprojekt
  - Cognidata GmbH (`http://www.cognidata.de/`)
  - FH Gießen
  - Uni Marburg
- Entscheidungsunterstützendes System
  (engl. *Decision Support System* – DSS)
- . . . im Bereich Produktlebenszyklus
  - Beginning of Life
  - Middle of Life

Kunden

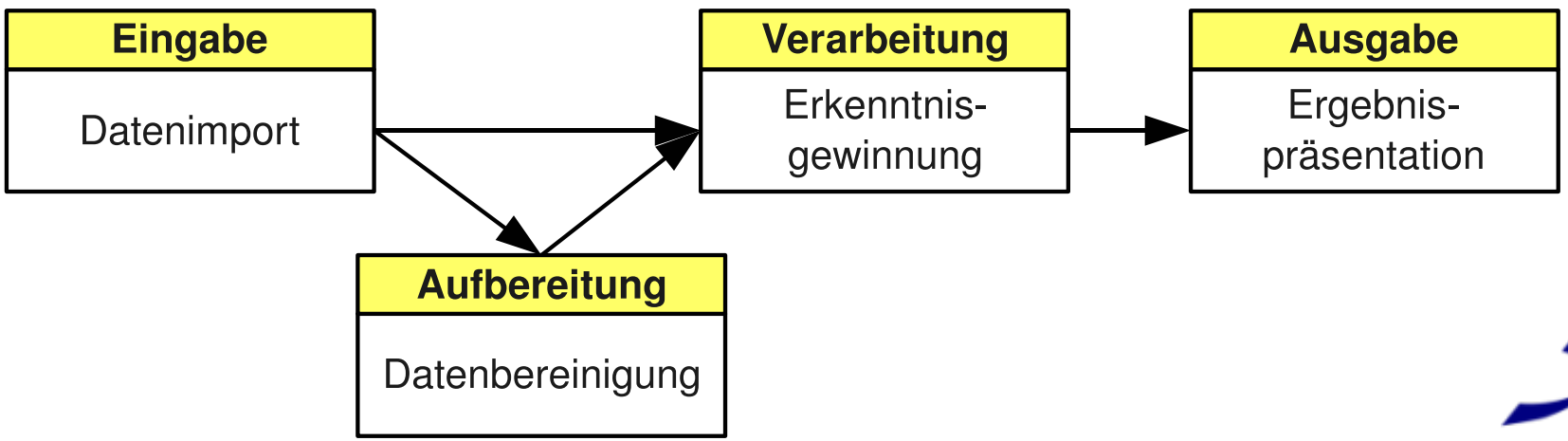
- Bombadier Transportation
  - Hersteller von Lokomotiven
  - PARASUITE unterstützt bei *Beginning of Life*
  - bessere Produkte herstellen
  - Fehlerquellen finden
  - Schlechte Teile ausfindig machen

Kunden

- Bombadier Transportation
  - Hersteller von Lokomotiven
  - PARASUITE unterstützt bei *Beginning of Life*
  - bessere Produkte herstellen
  - Fehlerquellen finden
  - Schlechte Teile ausfindig machen
- Thyssen Elevators
  - Hersteller von Aufzügen
  - PARASUITE unterstützt bei *Middle of Life*
  - Vorausschauende Produktwartung
    engl. *Predictive Maintainance*
  - Planen von Wartungsarbeiten
  - Wann fällt wo welches Teil aus?

Wie arbeitet PARASUITE?

- Kunden sammeln Daten
  - Fehler- und Ausfallmeldungen (FAMs)
  - Status von Komponenten (Snapshot)
  - ...
- PARASUITE wertet diese Daten aus
  - Data Mining (Erkenntnisgewinnung aus Daten)
  - Data Cleaning (Datenbereinigung)
  - Grafische Präsentation der Ergebnisse (Diagramme)



Eingesetzte Technologien und Tools

Programmiersprachen Java EE (EJB, JPA, Servlets und JSP), XML

Application Server JBoss

Datenbank-System MySQL

GUI-Framework Eclipse RCP mit BIRT (Reporting, Grafische Datenauswertung)

Inhalt

Inhalt

Datenvorberechnung

- BIRT kann Daten auswerten
- dauert aber viel zu lange
- deshalb müssen Daten vorberechnet werden
- sehr große Datenmengen
- lange Berechnungszeiten

Datenvorberechnung

- **Früher**
  - Direkte Datenbankabfragen
  - hart-kodierte SQL-Abfragen
  - unflexibel
  - Codeänderungen
- **Jetzt** (in Zukunft)
  - Benutzer soll Berechnungen zusammenklicken können
  - Berechnungsnetzwerke
    - Knoten
    - Verbindungen
  - Flow Based Programming

Inhalt

Flow Based Programming (FBP)

Was ist Flow Based Programming?

- Programmierparadigma
- Erfinder: John Paul Morrison (70er Jahre)
- Eigenschaften
  - Netzwerke von „black-box“ Prozessen
  - Prozess = Komponente
  - Verbindungen zwischen Prozessen
  - Informationspakete
- als Framework verfügbar für Java und C#.NET

Bestandteile

Komponente (auch: Knoten) I

- für Verarbeitung zuständig
- meist eine spezielle Aufgabe
- als Thread realisiert
- hat *Ports* (beliebig viele)
- Port-Typen:
  - Input Port
  - Output Port
  - Array Input Port
  - Array Output Port

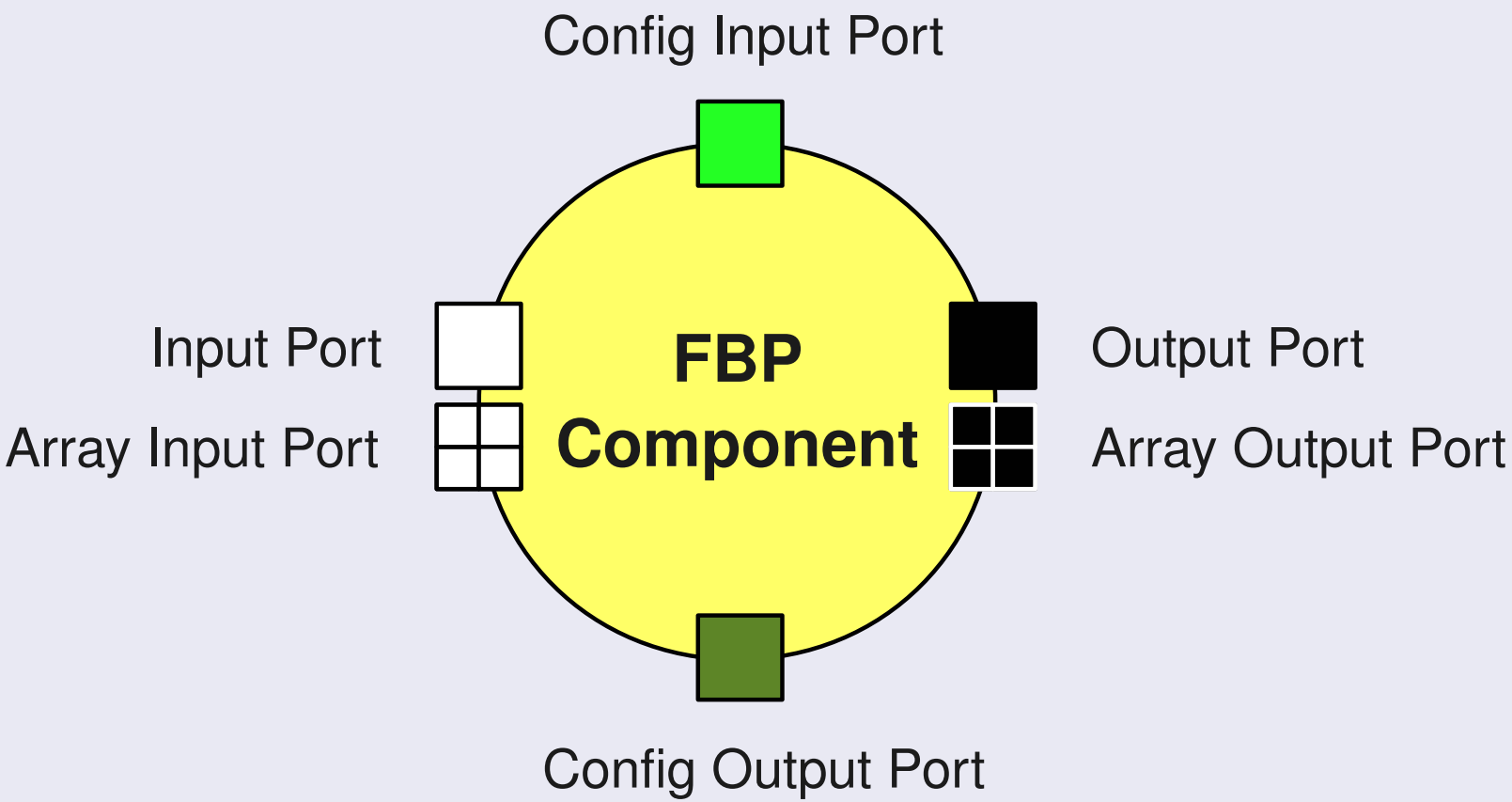
Bestandteile

Komponente (auch: Knoten) I

- für Verarbeitung zuständig
- meist eine spezielle Aufgabe
- als Thread realisiert
- hat *Ports* (beliebig viele)
- 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

Komponente (auch: Knoten) II



Bestandteile

Verbindung

- gemeinsam genutzte Datenstruktur
- ähnlich einer `BlockingQueue` in Java
- verbindet Ausgangsport mit Eingangsport
- uni-direktional (vgl. UNIX Pipe)
- transportiert *Information Packets* (IPs)

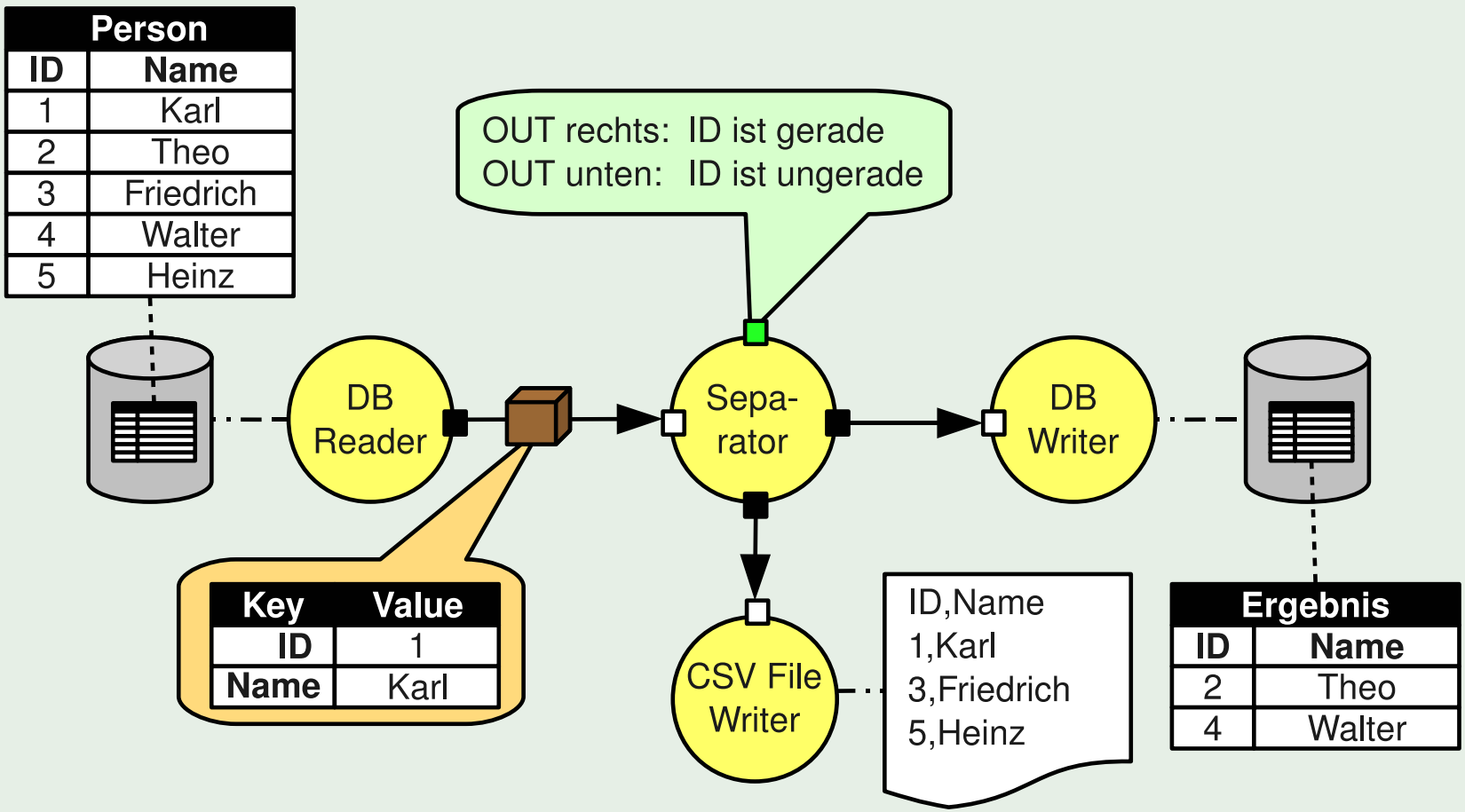
Bestandteile

Subnetz

- enthält ein Teilnetzwerk
- besteht aus Komponenten und Verbindungen
- hat eigene Ports
- Modularisierung
- Wiederverwendbarkeit
- Abstraktion
- Vereinfachung der Komplexität
- beliebig tiefe Schachtelung

Bestandteile

Beispiel



Inhalt

Inhalt

Meine Aufgaben in der Projektphase

Primäre Aufgabe

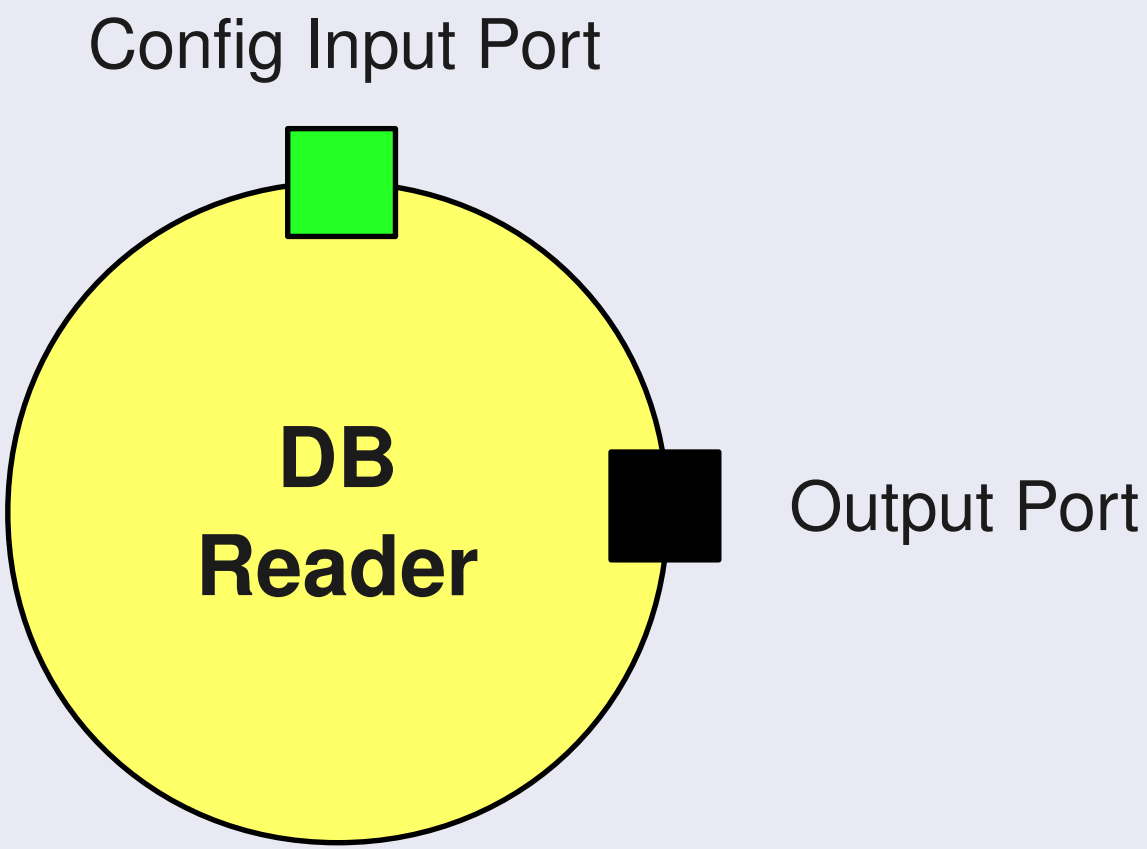
Entwicklung eines Datenbank-Leseknotens für das neue Datenbankschema „PARASUITE2“

Sekundäre Aufgabe

Entwicklung eines Datenbank-Schreibknotens

DB-Leseknoten

Layout des DB-Leseknotens



DB-Leseknoten

Arbeitsweise

1. Lesen der Konfiguration vom Config Input Port
2. Schließen des Config Input Ports
3. Hauptverarbeitungsschleife (Ende: alle Datensätze gelesen)
   1. Datensatz lesen aus DB
   2. Datensatz in *Information Packet* transformieren
   3. *Information Packet* aus Output Port senden
4. Schließen des Output Port

Inhalt

Konfiguration des DB-Leseknotens

- Datenstruktur in *Information Packets* ist eine `HashMap`
  - **Key:** `String`
  - **Value:** `Object`
- Ein Konfigurationspaket wird eingelesen

Konfiguration des DB-Leseknotens

- Datenstruktur in *Information Packets* ist eine `HashMap`

  Key: `String`
  Value: `Object`

- Ein Konfigurationspaket wird eingelesen

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`

Filter-Konfiguration

- PARASUITE-GUI: Filter-Dialog (`FilterExpression`)
- DB-Schnittstelle erwartet ein solches Objekt
- Objekt muss vom Client zum Server
- Erster Gedanke: Binäre Serialisierung

Filter-Konfiguration

- PARASUITE-GUI: Filter-Dialog (`FilterExpression`)
- DB-Schnittstelle erwartet ein solches Objekt
- Objekt muss vom Client zum Server
- Erster Gedanke: Binäre Serialisierung
- Aber: Serialisierung muss textbasiert sein
  - Netzwerkbeschreibung ist in XML
  - für Debugging-Zwecke (Menschenlesbarkeit)

Filter-Konfiguration

Erster Ansatz (BASE-64)

Client: `FilterExpression`-Objekt serialisieren und in BASE-64 umwandeln

Server: Rückwandlung und De-Serialisierung

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

Fazit: keine gute Lösung

Filter-Konfiguration

Zweiter Ansatz (SQL)

Client: `FilterExpression`-Objekt in SQL umwandeln (Funktionalität vorhanden)

Server: SQL parsen

- Wäre eine gute Lösung
- Aber: SQL enthält nicht genügend Informationen

Fazit: keine Lösung

Filter-Konfiguration

Dritter Ansatz (Eigene Sprache)

Client: `FilterExpression`-Objekt 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

Inhalt

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 ::= <TEXT> | <INTEGER> | <TIME> | ...

     Values ::= Value | Value "," Values

  ValueList ::= "VALUES(" Values ")"

 FilterExpr ::= "{" <IDENTIFIER> <OPERATOR>
                ValueList "}"

    ...
```

Realisierung von PFEL

- Selbstgeschriebene Komponenten
  - PFELScanner
  - PFELParser
  - PFELPrinter
- **Alternative:** Parser-Generator, z. B. JavaCC
  - automatisch generierter Code
  - nicht so fehleranfällig
  - schneller?
- ABER: Für kleine Sprachen – wie PFEL – ist ein selbstgeschriebener Parser völlig in Ordnung
  - übersichtlicher
  - verständlicher

Inhalt

Weitere Aufgabe in der Projektphase

- Value List Handler
  - Effizientes Abfragen der DB
  - Portionsweises auslesen (Paging-Prinzip)
  - . . . anstatt alle Datensätze auf einmal
- Lokalisierung des Daten-Caches
  - **Vorher:** Cache hält Daten einer Lokalisierung
  - **Jetzt:** Pro Lokalisierung „ge-cache-te“ Daten
- . . .

Inhalt

Ausblick

- Generischer Leseknoten
  - Lesen aus Datei, URL, DB, Arbeitsspeicher, ...
- Generischer Schreibknoten
  - Schreiben in Datei, URL, DB, Arbeitsspeicher, ...

Ausblick

- Generischer Leseknoten
  - Lesen aus Datei, URL, DB, Arbeitsspeicher, ...
- Generischer Schreibknoten
  - Schreiben in Datei, URL, DB, Arbeitsspeicher, ...
- XML anstatt PFEL
  - Zum Testen besser (Alle Infos vorhanden)
  - Lesbarkeit wird schlechter (viel Overhead)
  - komplette Serialisierung möglich
- Grafischer Konfigurationsdialog für Komponenten

Literatur


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


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


...

Fragen?

Danke für die Aufmerksamkeit!