Seminar Datenbanken und Informationssystem

# XML-Databinding

Sven Welte
31.01.2003

# Überblick

- XML-Databinding
  - Grundlagen
  - JAXB
  - weitere Produkte
- Java XML-Serialisierung
- Spezielle Speicherungslösungen
  - Persistent DOM
  - Natix
- Zusammenfassung
  - Vor/Nachteile XML-Databinding
  - Ausblick

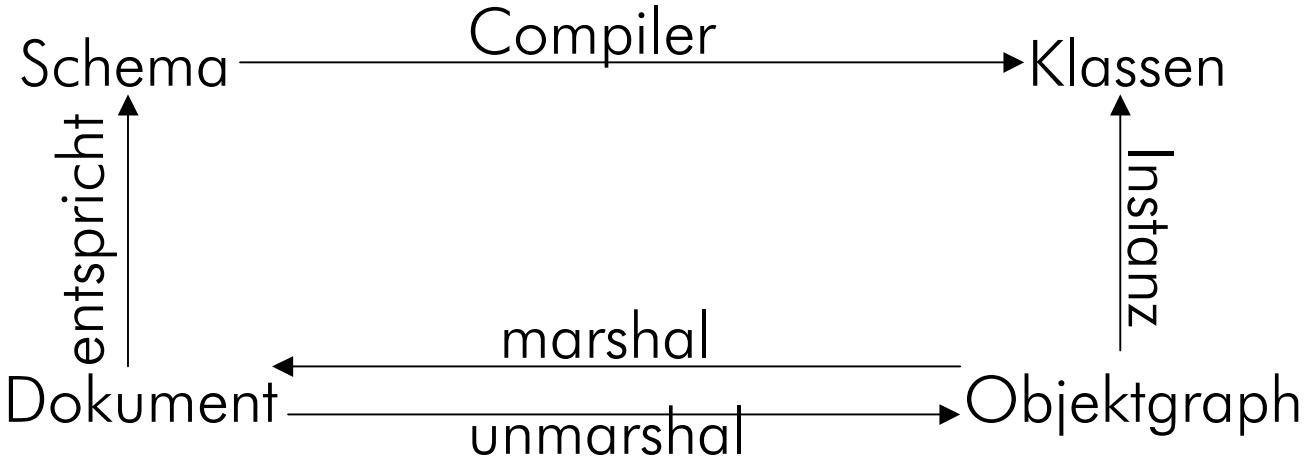
# Motivation

Aufgabe: (XML-Datei+Schema gegeben)
Einlesen eines XML-Dokuments und Speicherung auf Javaseite in Geschäftsobjekten (z.B. Movie, Cast). Nach Verarbeitung auf Javaseite Speicherung der Geschäftsobjekte in neuem XML-Dokument.

```
<movie>
   <title>Pitch Black</title>
   <cast>
     <actor headliner="true">Vin Diesel</actor>
     <actor headliner="true">Radha Mitchell</actor>
     <actor>Vic Wilson</actor>
   </cast>
   <producer>Tom Engelman</producer>
 </movie>
 <movie> .... </movie>
```

Lösung mit SAX: Schreiben von mind. 200 Zeilen Code (Geschäftsobjekte, Speicherungslogik, Ladelogik)
Lösung mit JAXB: Schreiben von 4 Zeilen Code

# Grundlagen: XML-Databinding

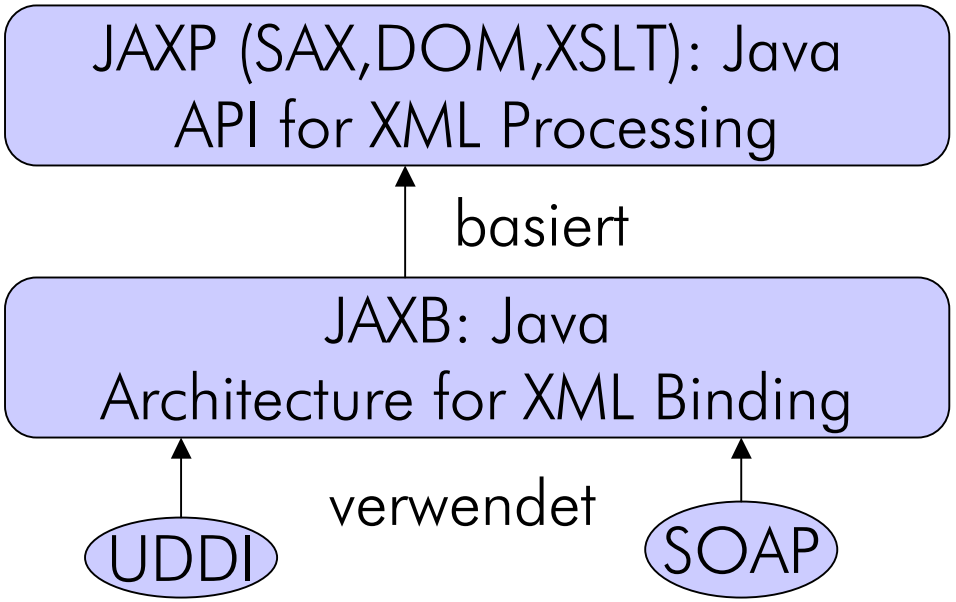



- XML-Databinding-Lösung besteht aus:
  - Klassengenerator
  - Marshalling Framework

| Abbildung | XML | Java |
|---|---|---|
| | Schema | Klassen |
| | Element | Klasse |
| | Attribut | Klassenattribut |

# Produkte: JAXB

- neuer Standard: Java Architecture for XML Binding (JAXB)
- von Sun entwickelt (JSR31)
- aktuelle Version 1.0 beta
- prinzipielles Vorgehen auf andere Produkte übertragbar
- Einordnung:



# JAXB: Beispiel

## movies.dtd

```
<!ELEMENT movies (movie+)>
<!ATTLIST movies
        version CDATA    #REQUIRED>
<!ELEMENT movie (title, cast,
            director?, producer*)>
<!ELEMENT cast (actor+)>
<!ELEMENT title (#PCDATA)>
<!ELEMENT director (#PCDATA)>
<!ELEMENT producer (#PCDATA)>
<!ELEMENT actor (#PCDATA)>
<!ATTLIST actor  headliner
         (true | false) 'false'>
```

## Bindungsschema movies.xjs

```
<xml-java-binding-schema
          version="1.0-ea">
 <options package=
          "generated.movies"/>
 <element name="movies"
          type="class"
          root="true"/>
</xml-java-binding-schema>
```

## Aufruf Codegenerator:

```
xjc movies.dtd movies.xjs
```

## Erzeugte Klasse Movie.java

```
package generated.movies;
...
class Movie extends
            MarshallableObject
  String getTitle() {...}
  void setTitle(String _Title)
   {...}
  Cast getCast() {...}
  void setCast(Cast _Cast) {...}
  List getProducer() {...}
  void deleteProducer() {...}
  void emptyProducer() {...}
  void validate() throws
   StructureValidationException
  {...}
  void marshal(OutputStream out)
  {...}

  static Movie unmarshal
         (InputStream in) {...}
  boolean equals(Object ob){...}
  int hashCode() {...}
  String toString() {...}
  ...
```

## Anwendung:

```
// Filme einlesen
InputStream in =
   new FileInputStream (”input.xml”);
Movies movies = Movies.unmarshal(in);
// Filme bearbeiten
....
// Filme in XML-Datei speichern
File f = new File(”output.xml”);
OutputStream out =
  new FileOutputStream (f);
Movies.validate();
Movies.marshal(out);
```

# JAXB: Bindungsschema

Anpassung der generierten Klassen durch Bindungsschema

- Verwendung eines Elements als Java-Attribut

```
<element name="title" type="value"/>
```

- Umbenennung der Klasse „movie-data“ nach „Movies“

```
<element name="movie-data"  type="class"
         class="Movies"     root=true/>
```

- Abbildung vom Element „copyrightYear“ auf Attribut vom Typ int

```
<element name="copyrightYear" type="value"
         convert="int"/>
```

# JAXB: Bindungsschema

- Verwendung benutzerdefinierter Typen
  - Aufzählungen werden ähnlich realisiert

## Hilfsklasse

```
class DateConversion {
 static SimpelDateFormat df =
   new SimpleDateFormat (”dd.mm.yy”);

 static Date parseDate (String d) {
   return df.parse(d);
 }

 static String printDate(Date d) {
   return df.format(d);
 }
}
```

## Bindungsschema

```
<conversion name=”Date”
  type=”java.util.Date”
  parse=”DateConversion.parseDate”
  print=”DateConversion.printDate”
/>

<element name=”movie”
         type=”class”>
  <attribute name=”releaseYear”
             convert=”Date”/>
</element>
```

# XML-Serialisierung in Java

- JSR57: Long-Term Persistence for JavaBeans
- seit Java 1.4 enthalten
- ursprünglich entwickelt zur Speicherung von Benutzeroberflächen
- Probleme mit bisheriger Serialisierung
  - Erstellte Dateien inkompatibel mit unterschiedlichen JVMs
  - Aufblähung der Dateien durch Serialisierung unnötiger Attribute
  - keine Fehlertoleranz
  - keine Migration möglich

# Weitere Produkte

Databinding:

- jBind (XML-Schema)
- Castor (XML-Schema)
- Zeus (DTD, XML-Schema)
- Quick (DTD, QDML)
- Microsoft .NET
- Borland Delphi/C++Builder (DTD, XML-Schema, XDR)
- ... Produkte für C++, Python

Serialisierung

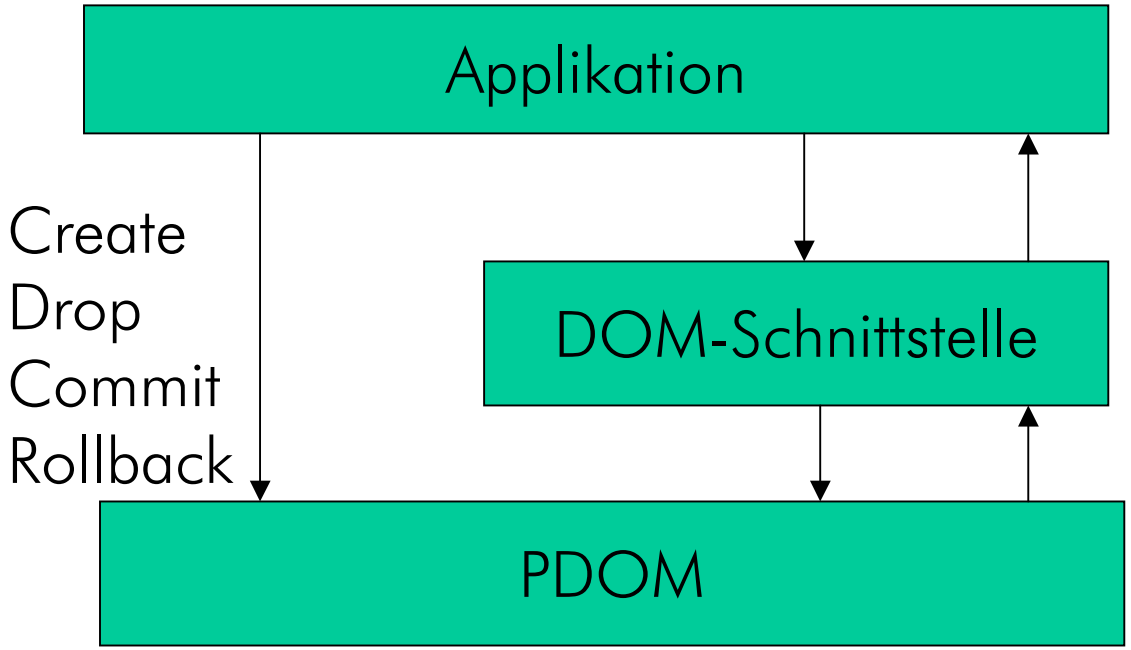
- JSX: Java Serialization to XML
- Apache Jakarta: Betwixt („Bean To XML“ + „between“)
- ...

# Speicherungslösungen

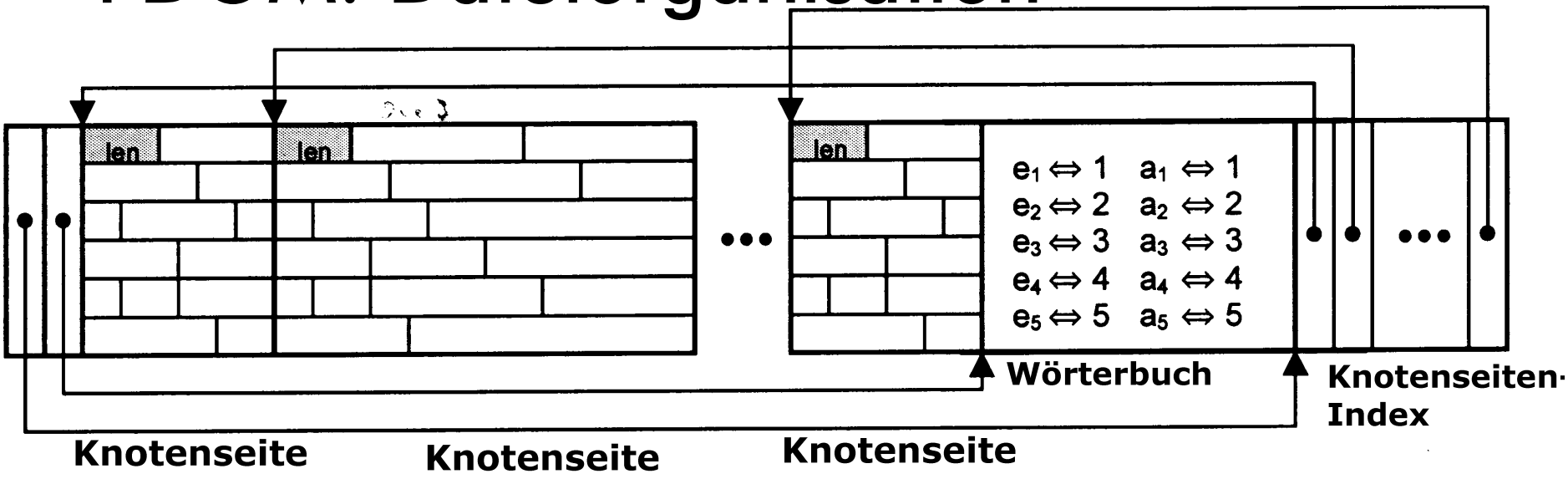
Nachteile von XML-Dokumenten

- XML-Dokumente durch wiederholende Nutzung von Tags recht aufgebläht
- Umwandlung von XML-Dokumenten vor dem Bearbeiten notwendig
- Probleme bei sehr großen XML-Dokumenten

# Persistent DOM: Architektur



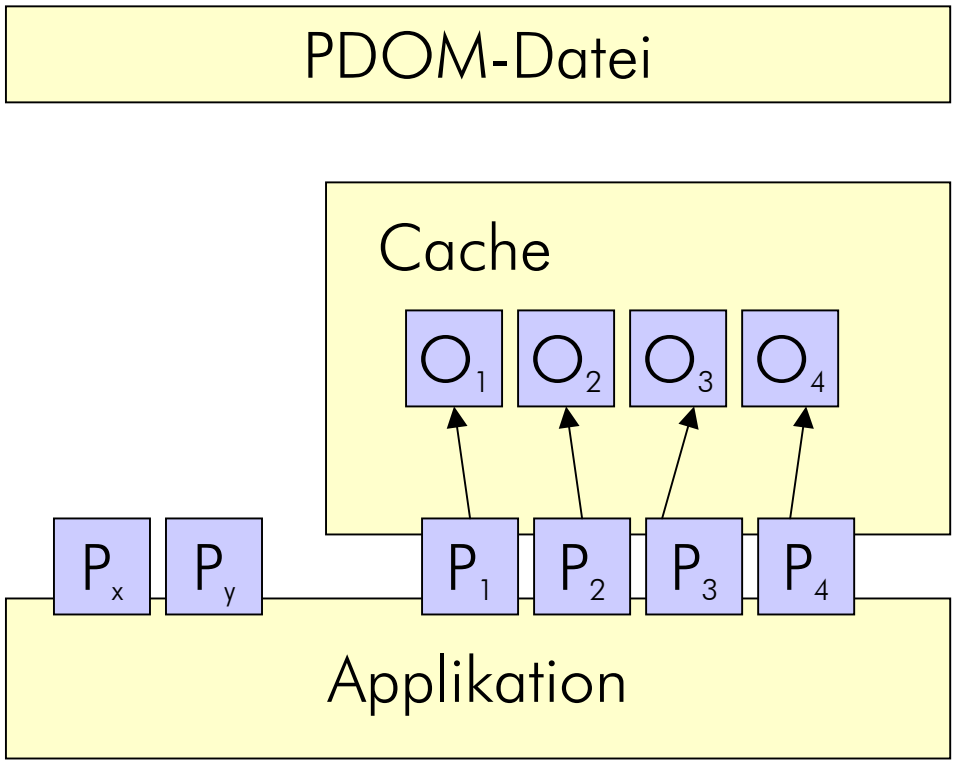
# PDOM: Dateiorganisation




Vorteile

- Kompaktheit (zus. Komprimierung, Wörterbuch)
- Schneller Zugriff durch geringe E/A
- Einfacher Wiederanlauf (Schattenspeicherkonzept)

# PDOM: Cacheverwaltung




Proxy Objekt
- Enthält eindeutige Objektadresse
- Enthält Cachezustand

Zugriff auf Objekt nur durch Proxy Objekt
- bei Bedarf nachladen

- Verwendet abgewandelten LRU-Algorithmus
  - Bei Baumtraversierung gelten Annahmen von LRU nicht
  - Deshalb älteste Einträge länger im Speicher belassen

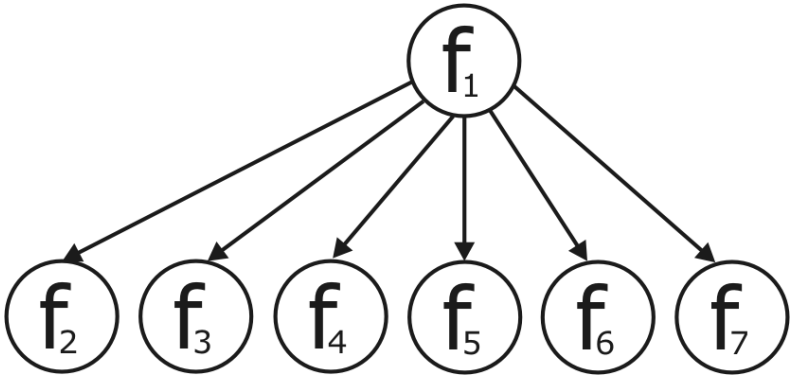
# PDOM: Performance

Aufgabe: Traversierung einer XML-Datei (7.5MB, 327.145 DOM-Objekten)

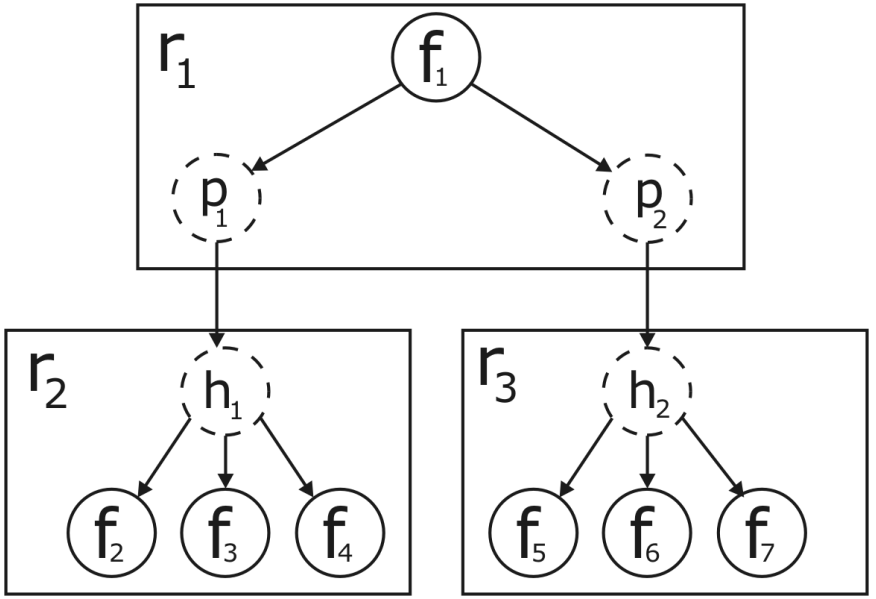
| | Einmalige Traversierung | | Median von 10 Traversierungen | |
|---|---|---|---|---|
| | Zeit[ms] | Speicher[KB] | Zeit[ms] | Speicher[KB] |
| IDOM | 17115 | 38219 | 412 | 38219 |
| PDOM, Cache | 6208 | 43176 | 705 | 43176 |
| PDOM, ohne Cache | 3065 | 839 | 2133 | 840 |

# NATIX: Struktur

Logische Struktur



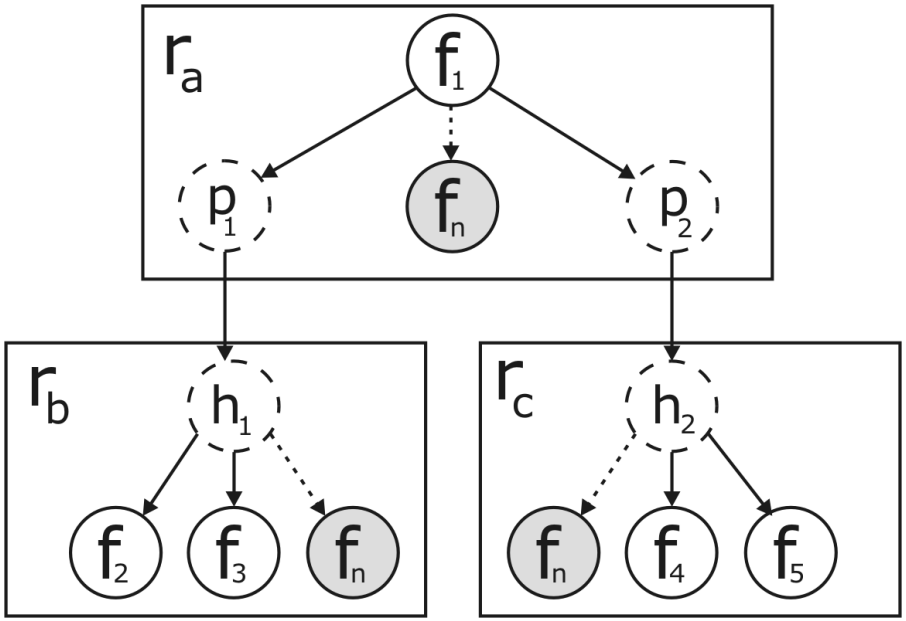

Physikalische Struktur




Aufteilung der Knoten auf Datensätze mit

- Proxy Knoten
- Hilfsknoten

# Natix: Einfügen




Steuerung durch Splitmatrix S

$p(\sum^{*}_{DTD}) \rightarrow N$ bijekt. Abbildung

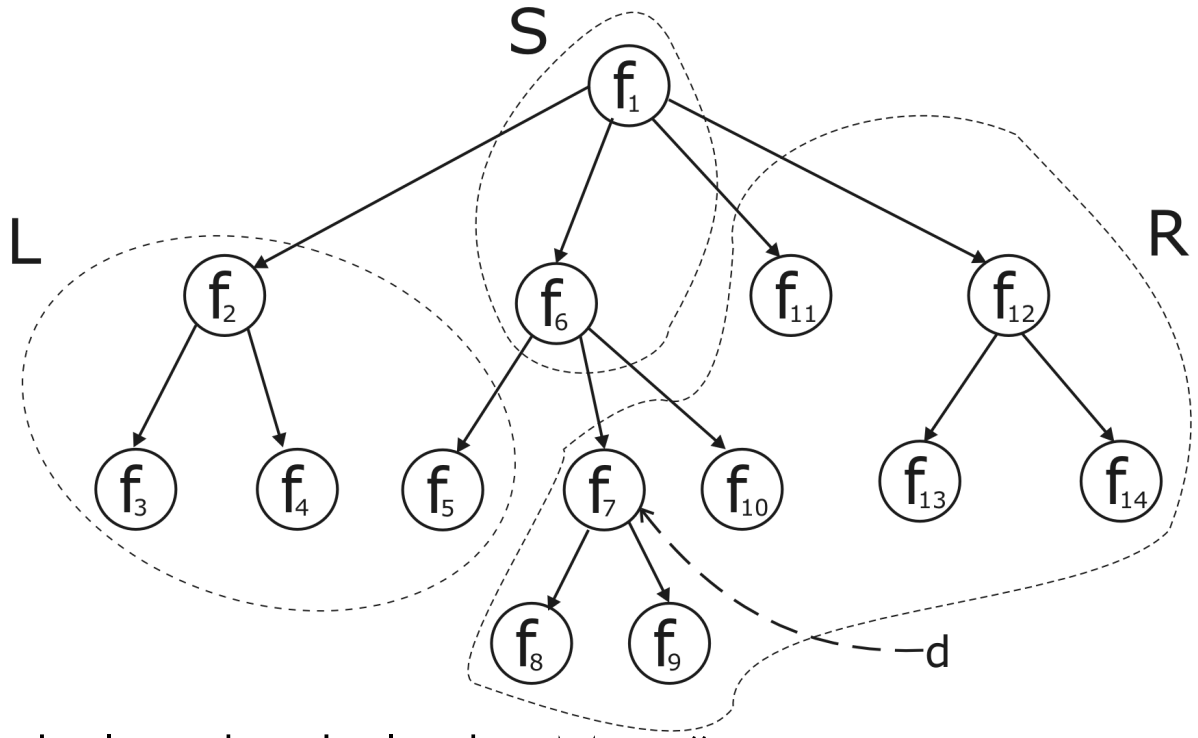
Sei $p(e_1)=i$, $p(e_2)=j$

$s_{ij}=0$ → $e_1$, $e_2$ niemals zusammen

$s_{ij}=inf$ → $e_1$, $e_2$ immer zusammen

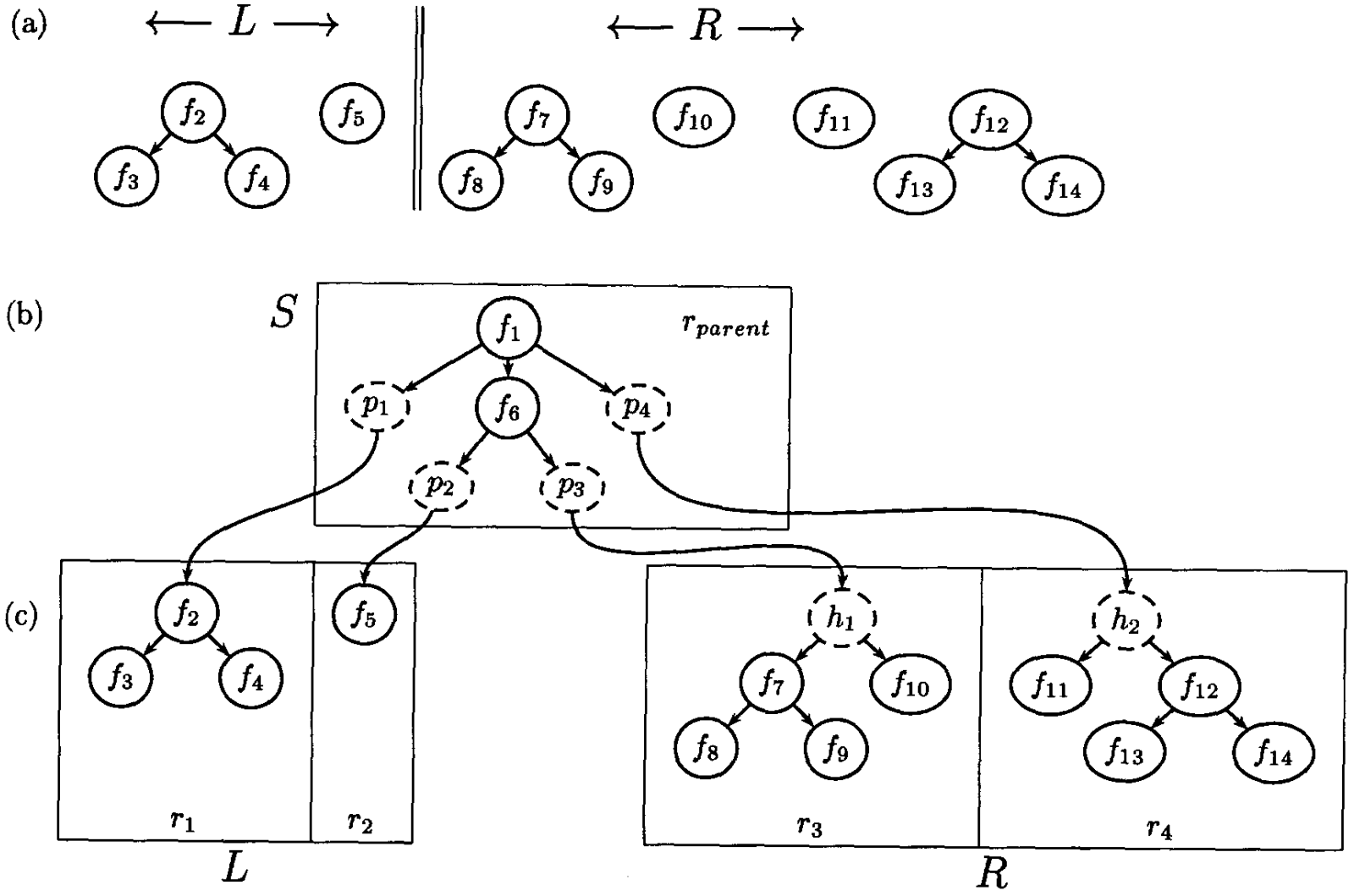
sonst → System entscheidet

# Natix: Splitting




- Wähle d als „physikalische Mitte“
- Separator S={Pfad von d zur Wurzel ohne d}
- R={Unterbaum von d; UB von d‘s rechten Geschwistern;
  UB von Knoten, die rechte Geschwister von Knoten aus S sind}
- L=Rest

# Natix: Aufteilen in Datensätze



# Zusammenfassung

- Umwandlung von XML-Dokument in Graph von Geschäftsobjekten recht einfach möglich
- Databindingprozess am Beispiel von JAXB betrachtet
  - Prozess mit Bindungsschema steuerbar
- PDOM ermöglicht effiziente Speicherung von XML-Daten
  - mit Cache fast so schnell wie DOM
  - ohne Cache deutlich geringerer Speicherplatz
- Datenstruktur von NATIX verhält sich ähnlich B*-Baum

# Zusammenfassung: Bewertung

- Vorteile XML-Databinding
  - automatische Erzeugung
  - minimale konsistente Schnittstelle
  - kaum Laufzeitnachteile
  - einfache Erlernbarkeit
  - Vereinfachung des Softwareentwicklungsprozesses (Design, Implementierung, Tests, Reviews, Verwaltung, Wartung)
- Nachteile XML-Databinding
  - kompletter Objektgraph im Hauptspeicher
  - keine Unterstützung von heterogenen XML-Dokumenten
  - keine Unterstützung von bestehenden Klassen
  - kaum Änderungen zur Laufzeit möglich
  - dokumententzentrierte XML-Dateien (z.B. bei Workflows) nicht gut unterstützt
  - Probleme mit selbstdefinierten Methoden

# Zusammenfassung: Ausblick

- JAXB wird Standardtechnologie
- Fokus von SAX/DOM verlagert sich auf XML-Databinding
- starke Verbreitung von XML-Databinding in andere Sprachen durch WebServices