

18. Mai 2005

Regelsprachen

Seminar Praktische Modellierung SS 2005
Institut für Informatik, FU-Berlin

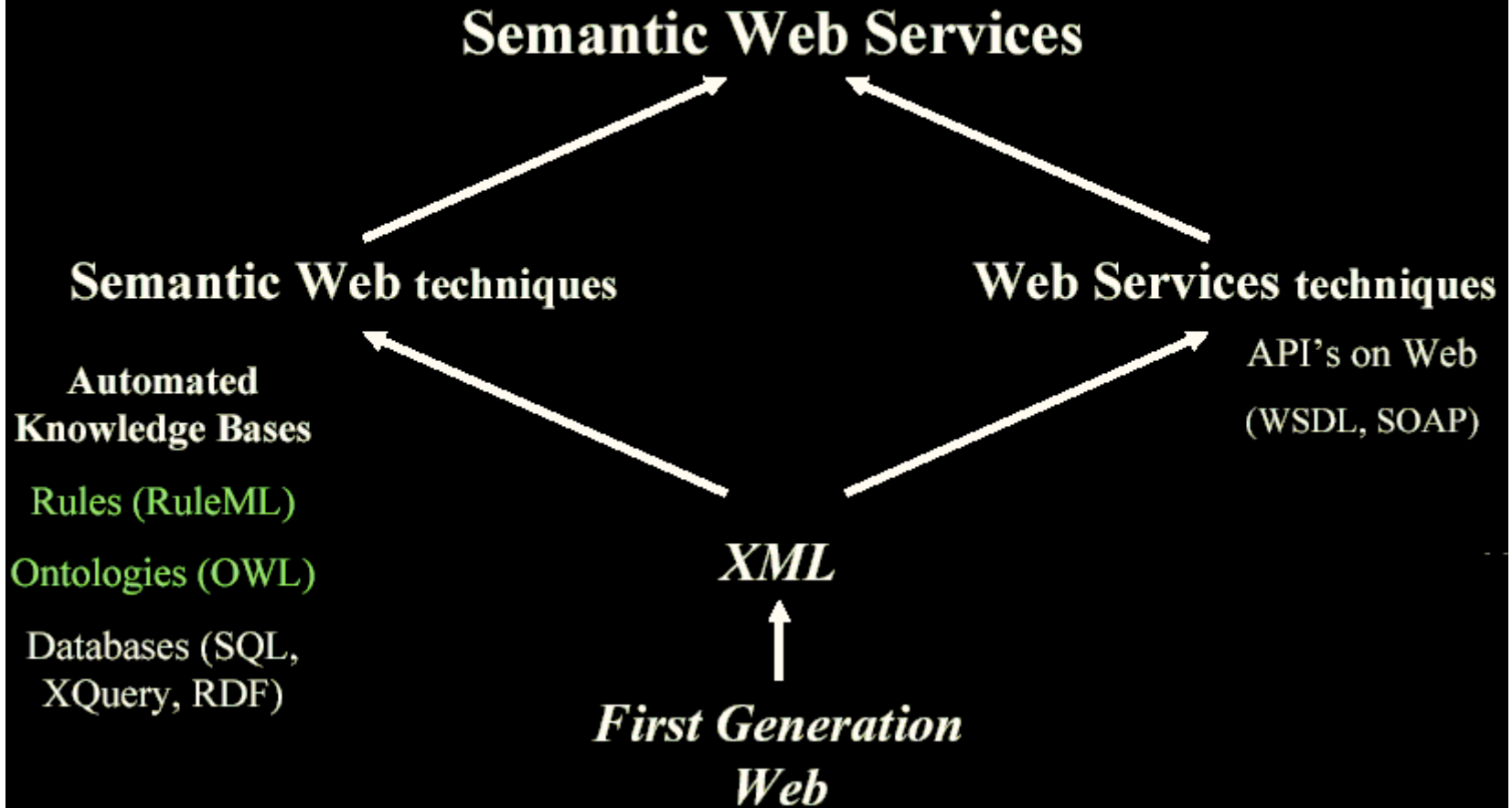
von Jamal Baydoun

baydoun@inf.fu-berlin.de

Referatsinhalt

- **Regeln und das Semantic Web**
Kategorien, Hierarchie, Beispiele
- **RuleML**
Grundidee, Kategorien, Hierarchie, Syntax, Datalog, Negation, Prioritäten
- **SWRL**
Grundidee, Abstrak Syntax, XML Syntax, Built-Ins, Beispiel
- **Tools**
SweetJess, OWLTrans
- **Literatur**

Next Generation Web

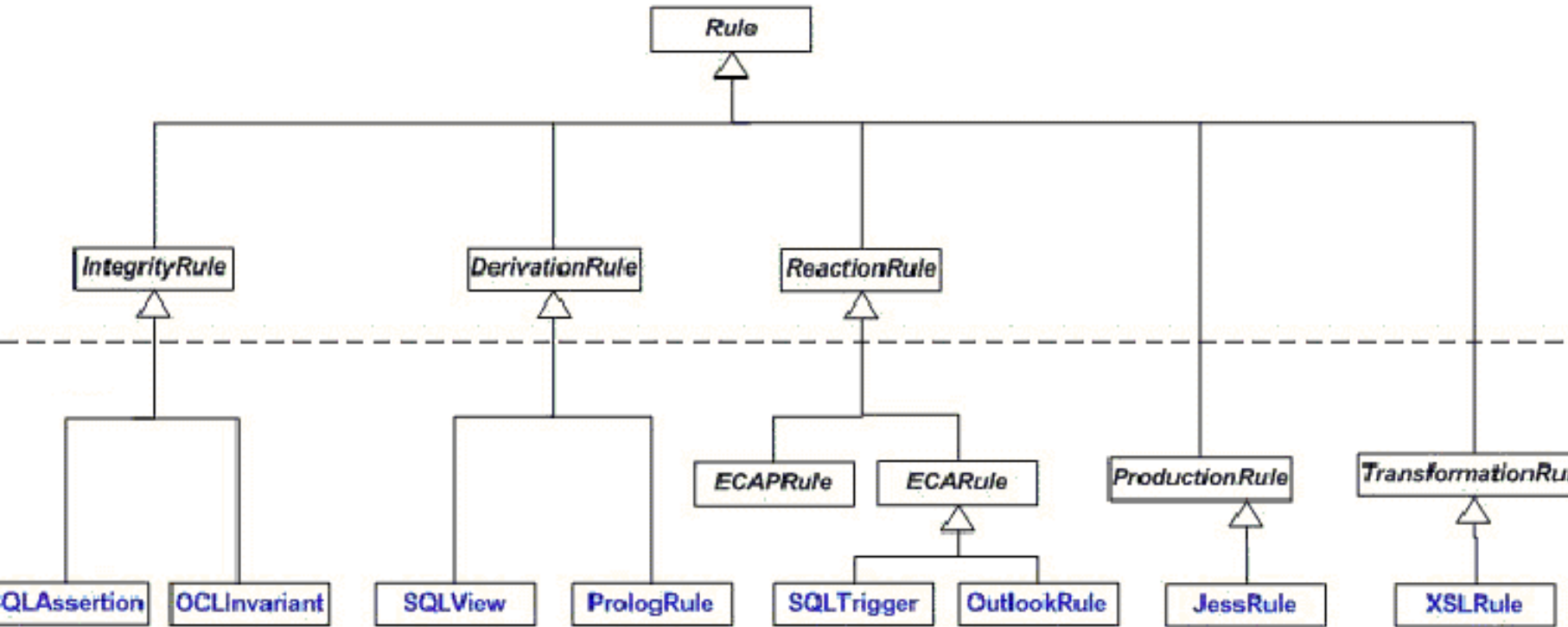


Benjamin Grosf and Mike Dean. 3rd International Semantic Web Conference, Nov. 7, 2004, Hiroshima, Japan

Mögliche Anwendungsgebiete für Regeln

- eCommerce: B2B Anwendungen
Business Rule Markup Language (BRML)
Wer seit mehr als 5 Jahren ein Kunde bei unserer Firma ist, bekommt 20% Rabatt.
- Internet-Zugriffsberechtigung
Wer in der letzten 2 Monaten ein Angestellter bei einer organization war, die irgendwann in der letzten 2 Monaten ein W3C-Mitglied war, kann sich zur W3C-mitgliedschaft anmelden.

Kategorien von Regeln



Was unterscheidet diese Kategorien voneinander?

1. Ableitungsregeln

- *Schlussregeln*
- Bestehen aus einer oder mehreren *Bedingungen* (Körper) und einer *Konklusion* (Kopf)

Prolog

availableCar(X) :-
rentalCar(X), not requiresService(X),
not isAssignedToSomeRental(X).

"Ein Auto ist zur Verfügung steht, wenn es ein Mietauto ist, nicht reparaturbedürftig ist, und nicht anderem Mieter zugewiesen ist"

2. Integritätsregeln

- Bestehen nur aus einem *Kopf*
- Die bekannteste Sprache, die Einschränkungsregel ausdrücken kann, ist **SQL**

```
CREATE TABLE RentalOrder(OrderNo
INTEGER PRIMARY KEY,Driver CHAR(10)
REFERENCES Person CHECK( 24 < (SELECT
Age FROM Person WHERE PersID = Driver)))
```

"Ein Mietautofahrer muss mindestens 25 Jahre alt sein"

- Anderes Beispiel **OCL**

3. Reaktionsregeln

- Bestehen aus einem *Ereignis*, einer *Bedingung*, einer *Aktion*, und möglicherweise einer *Nach-Bedingung*
- 2 Sorten **ECA-Regeln** und **ECAP-Regeln**

Trigger in SQL (Auslöser)

```
CREATE TRIGGER alert_service_station  
AFTER UPDATE ON RentalCar( requiresService)  
FOR EACH ROW WHEN requiresService  
CALL send_email('ServiceStation', 'New service required')
```

"Nach dem aktualisieren der Tabelle RentalCar wird der Autowerkstatt ein Email gesendet, falls das Auto reparaturbedürftig ist"

4. Produktionsregeln

- Bestehen aus einer *Bedingung* und einer *Aktion*

Jess Produktionsregel

```
(defrule availableCar
```

```
(and (RentalCar ?x)
```

```
(not (requiresService ?x))
```

```
(not (isAssignedToSomeRental ?x)))
```

```
⇒(assert (availableCar ?x))
```

"wenn X ein Mietauto ist, nicht reparaturbedürftig ist, und nicht anderem Mieter zugewiesen ist, dann gilt, dass es zur Verfügung steht"

5. Transformationsregeln

- Bestehen aus einer *Bedingung*, einer *Konklusion* und einer *transformationsausgabe*

XSL Regel

```
<xsl:template match="title">  
<h3><xsl:value-of select="text()" /></h3>  
</xsl:template>
```

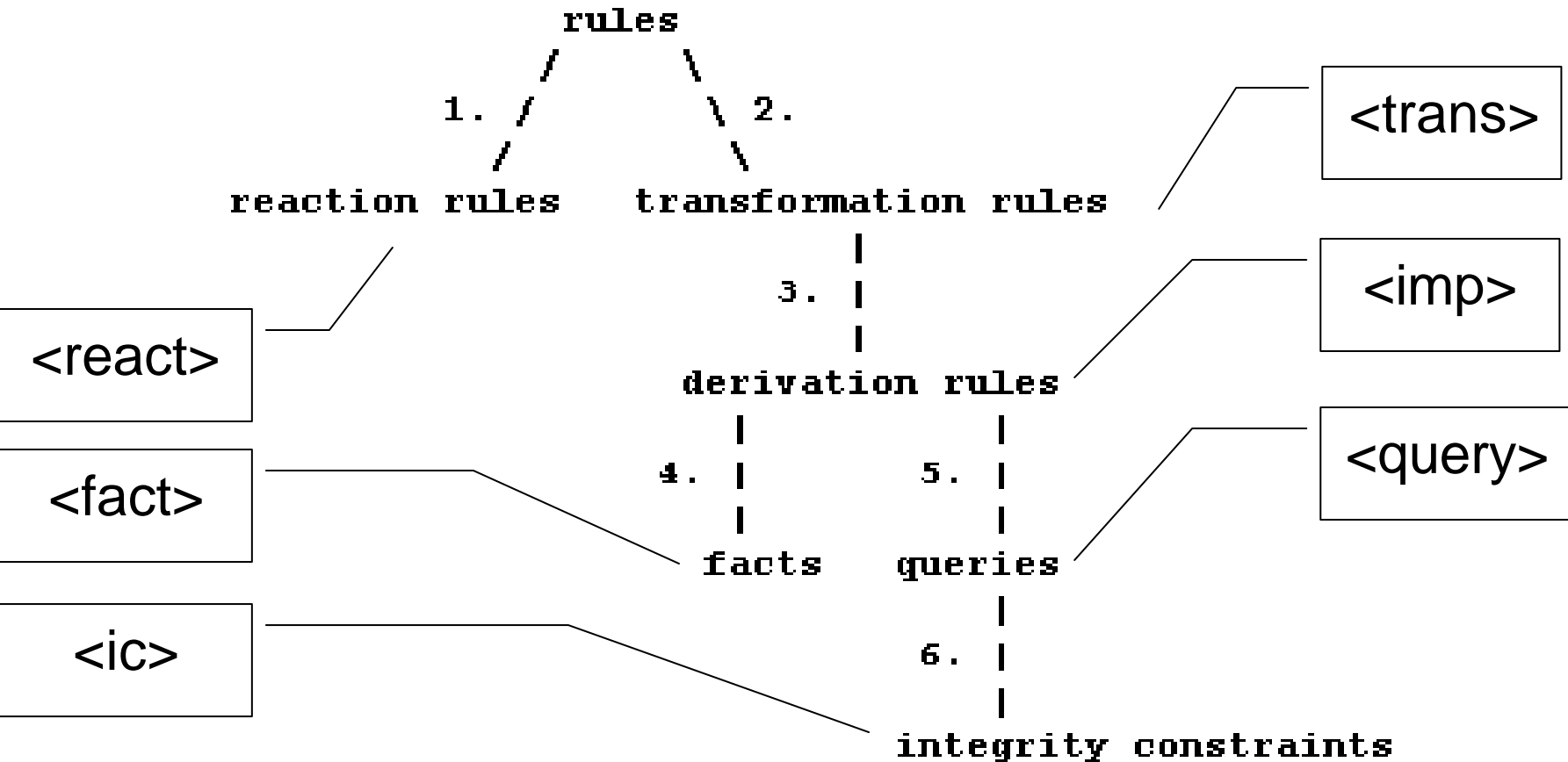
"transform die lange Beschreibung eines Buches in eine kurze Beschreibung, die nur den Titel dieses Buches enthält"

RuleML

- Regel-Auszeichnungssprache.
- Bisher kein W3C Standard
- **Die Grundidee**
XML-Namespace *RuleML* zur Verfügung zu stellen. Wie Z.B *MathML*

ermöglicht Regelaustausch

RuleML Hierarchy



RuleML Syntax

- Transformationsregel

<trans>

<_head>*Konklusion***</_head>**

<_body>**<and>***Bed1...BedN***</and>****</_body>**

<_foot>*Transformationsausgabe***</_foot>**

</trans>

- Ableitungsregel

<imp>

<_head>*Konklusion***</_head>**

<_body>**<and>***Bed1...BedN***</and>****</_body>**

</imp>

- **Fakten**

<fact><_head>*Konklusion*</_head>**</fact>**

Oder

<imp>

<_head>*Konklusion*</_head>

<_body><and></and></_body>

</imp>

- **Anfragen**

<query>

<_body><and>*Bed1...BedN*</and></_body>

</query>

- Integritätsregeln

<ic>

<_body><and>*Bed1...BedN*</and></_body>

</ic>

- Reaktionsregeln

<react>

<_event>*Auslöser*</_event>

<_body><and>*Bed1...BedN*</and></_body>

<_head>*Aktion*</_head>

</react>

"Datalog" RuleML Untersprache

- Untermenge von RuleML
- Entscheidbar
- Andere Untersprachen: ur-datalog, hornlog...
- Datalog XML Schema
- Datalog DTD

"Datalog" RuleML Beispiel

Eine Person besitzt ein Objekt, wenn diese Person das Objekt von einem Händler kauft, und die Person das Objekt aufbewahrt.

[own.ruleml](#)

[ur-own.ruleml](#)

Negation in RuleML

- Schwache Negation

"Ich mag nicht Schnee"

"I don't like snow"

Durch <not>.. </not> ausgedrückt

- Starke Negation

"Ich hasse Schnee"

"I dislike snow"

Durch <neg>.. </neg> ausgedrückt

Beispiel

<imp><_head>

<neg><atom>

<_opr><rel>istOffiziellesDokument</rel></_opr>

<var>DokumentName</var>

</atom></neg>

</_head><_body>

<not><atom>

<_opr>

<rel href="http://www.ebizz.com/rdf-sch#offiziell"/>

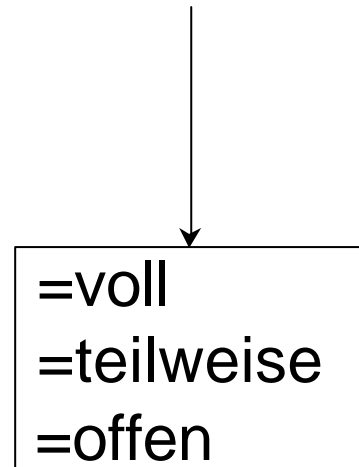
</_opr>

<var>DokumentName</var>

<ind>voll</ind>

</atom></not>

</_body></imp>



Prioritäten in RuleML

- Bei Museumseintritt bekommen Studenten 50% Rabatt
- Bei Museumseintritt bekommen Studenten, die älter als 30 Jahre sind, keinen Rabatt

Konflikt

Konfliktlösung

- Quantitative Lösung
- Qualitative Lösung

[student.ruleml](#)

RuleML Zusammenfassung

- Grundidee
- Syntax
- Datalog
- Negation
- Konfliktlösung
- Beispiele

SWRL

- Semantic Web Rule Language
- Kombiniert OWL DL und OWL Lite mit Datalog RuleML

SWRL = OWL DL/OWL Lite + Datalog RuleML

- Relativ neu
- Bisher kein W3C Standard
- Eventuelles Input in W3C Prozess

SWRL Vorteil

- Verbessert OWL Ausdrucksmächtigkeit
- Die Regel drücken Dinge aus, die in OWL nicht ausdrückbar sind

Beispiele:

- Ein Onkel ist der Bruder eines Vaters
- 2 Geschwister haben denselbe Vater
- Ein Erwachsene ist eine Person, die $\text{Alter} > 17$
- Ein Internationaler Flug erfordert Flughäfen in verschiedenen Ländern

SWRL Abstrakt Syntax

- Ontologie ist eine Sammlung von Axiomen und Fakten: *SubClass, equivalentClass...*
- SWRL stellt *Rule* als neues Axiom zur Verfügung
axiom ::= rule
- rule ::= Implies(body, head)
body ::= { atom }
head ::= { atom }

- atom ::= description (x)
| dataRange (x)
| individualvaluedPropertyID (x, y)
| datavaluedPropertyID (x, y)
| sameAs (x, y)
| differentFrom (x, y)
| builtIn ()
- Der Kopf einer Regel ist erfüllt, wenn alle Atomen des Körpers erfüllen sind

XML Syntax

- benutzte Namespaces:
<http://www.w3.org/2003/11/swrlx>
<http://www.w3.org/2003/11/ruleml>
<http://www.w3.org/2003/05/owl-xml> (owlx)
<http://www.w3.org/2001/XMLSchema> (xsd)
- <Ontology> Element hat zwei neue Elemente
[<ruleml:imp>](#) und [<ruleml:var>](#)

- `<ruleml:var>xsd:string</ruleml:var>`

Beispiel: `<ruleml:var>x1</ruleml:var>`

- `<ruleml:imp>`
`<ruleml:_rlab ruleml:href = xsd:anyURI>`
Beschreibung
`</ruleml:_rlab>`
`<ruleml:_body> (swrlx:atom) </ruleml:_body>`
`<ruleml:_head> (swrlx:atom) </ruleml:_head>`
`</ruleml:imp>`

- swrlx:classAtom
(*description* in der abstrakten Syntax)

<swrlx:classAtom>

<owlx:Class owlx:name="Person" />

<ruleml:var>x1</ruleml:var>

</swrlx:classAtom>

- *classAtom* ist erfüllt, wenn *x1* eine Instanz der Klasse *Person* ist.

- swrlx:datarangeAtom

```
<swrlx:datarangeAtom>
```

```
<owlx:OneOf>
```

```
<owlx:DataValue owlx:datatype="&xsd:int">
```

```
5
```

```
</owlx:DataValue>
```

```
<owlx:DataValue owlx:datatype="&xsd:int">
```

```
10
```

```
</owlx:DataValue>
```

```
</owlx:OneOf>
```

```
<ruleml:var>x2</ruleml:var>
```

```
</swrlx:datarangeAtom>
```

- *datarangeAtom* ist erfüllt, wenn *x2* entweder den Wert 5 oder 10 hat.

- swrlx:individualPropertyAtom

```
<swrlx:individualPropertyAtom swrlx:property="hasParent">  
  <ruleml:var>x3</ruleml:var>  
  <owlx:Individual owlx:name="John" />  
</swrlx:individualPropertyAtom>
```

- *individualPropertyAtom* ist erfüllt, wenn *x3* ein Individual ist, das das Individual John als Wert der Attribut *hasParent* hat.

- swrlx:datavaluedPropertyAtom

```
<swrlx:datavaluedPropertyAtom swrlx:property="age">  
  <ruleml:var>x4</ruleml:var>  
  <owlx:DataValue  
    owlx:datatype="&xsd;int">24</owlx:DataValue>  
</swrlx:datavaluedPropertyAtom>
```

- *datavaluedPropertyAtom* ist erfüllt, wenn *x4* ein Individual ist, das 24 als Wert der Attribut *age* hat.

- swrlx:sameIndividualAtom
/ swrlx:differentIndividualAtom

```
<swrlx:sameIndividualAtom>  
  <ruleml:var>x1</ruleml:var>  
  <ruleml:var>x2</ruleml:var>  
  <owlx:Individual owlx:name="Clinton" />  
  <owlx:Individual owlx:name="Bill_Clinton" />  
</swrlx:sameIndividualAtom>
```

- *sameIndividualAtom* ist erfüllt, wenn *x1 Clinton* entspricht und *x2 Bill_Clinton* entspricht oder umgekehrt.

Built-Ins

- Benutzt als Atome in SWRL swrlx:builtinAtom
- Stammen aus XQuery and Xpath
- Identifiziert durch einen Namensraum
<http://www.w3.org/2003/11/swrlb>
- Comparisons Built-Ins
- Math Built-Ins
- Boolean Values Built-Ins
- Strings Built-Ins
- Date, Time Built-Ins
- URIs Built-Ins
- Lists Built-Ins

Comparisons	Math	String	Date, Time	URIs
equal	add	stringEqualIgnore Case	dateTime	resolveURI
notEqual	subtract	stringConcat	date	anyURI
lessThan	multiply	substring	time	List
lessThanOrEqualTo	divide	stringLength	yearMonth Duration	member
greaterThan	integerDivide	upperCase	subtractDates	length
greaterThanOr Equal	mod	lowerCase	subtractTimes	first
	round	translate		sublist
	sin	substringBefore		empty
	cos	substringAfter		Boolean
	tan	matches		booleanNot
	pow	replace		

Ein Onkel ist der Bruder eines Vaters (1)

```
<ruleml:imp>
```

```
  <ruleml:_rlab ruleml:href="#example1"/>
```

```
  <ruleml:_body>
```

```
    <swrlx:individualPropertyAtom swrlx:property="hasParent">
```

```
      <ruleml:var>x1</ruleml:var>
```

```
      <ruleml:var>x2</ruleml:var>
```

```
    </swrlx:individualPropertyAtom>
```

```
    <swrlx:individualPropertyAtom swrlx:property="hasBrother">
```

```
      <ruleml:var>x2</ruleml:var>
```

```
      <ruleml:var>x3</ruleml:var>
```

```
    </swrlx:individualPropertyAtom>
```

```
  </ruleml:_body>
```

```
  <ruleml:_head>
```

```
    <swrlx:individualPropertyAtom swrlx:property="hasUncle">
```

```
      <ruleml:var>x1</ruleml:var>
```

```
      <ruleml:var>x3</ruleml:var>
```

```
    </swrlx:individualPropertyAtom>
```

```
  </ruleml:_head>
```

```
</ruleml:imp>
```

Ein Onkel ist der Bruder eines Vaters (2)

```
<ruleml:imp>
  <ruleml:_rlab ruleml:href="#example2"/>
  <ruleml:_body>
    <swrlx:individualPropertyAtom swrlx:property="hasParent">
      <ruleml:var>x1</ruleml:var><ruleml:var>x2</ruleml:var>
    </swrlx:individualPropertyAtom>
    <swrlx:individualPropertyAtom swrlx:property="hasSibling">
      <ruleml:var>x2</ruleml:var><ruleml:var>x3</ruleml:var>
    </swrlx:individualPropertyAtom>
    <swrlx:individualPropertyAtom swrlx:property="hasSex">
      <ruleml:var>x3</ruleml:var>
      <owlx:Individual owlx:name="#male" />
    </swrlx:individualPropertyAtom>
  </ruleml:_body>
  <ruleml:_head>
    <swrlx:individualPropertyAtom swrlx:property="hasUncle">
      <ruleml:var>x1</ruleml:var><ruleml:var>x3</ruleml:var>
    </swrlx:individualPropertyAtom>
  </ruleml:_head></ruleml:imp>
```

SWRL Zusammenfassung

- Grundidee
- Abstrak Syntax
- XML Syntax
- Built-Ins
- Beispiel

Tools

- SweetJess
- OWLTrans (Jing Mei, Netzbasierte Informationssysteme, Freie Universität Berlin, Germany)

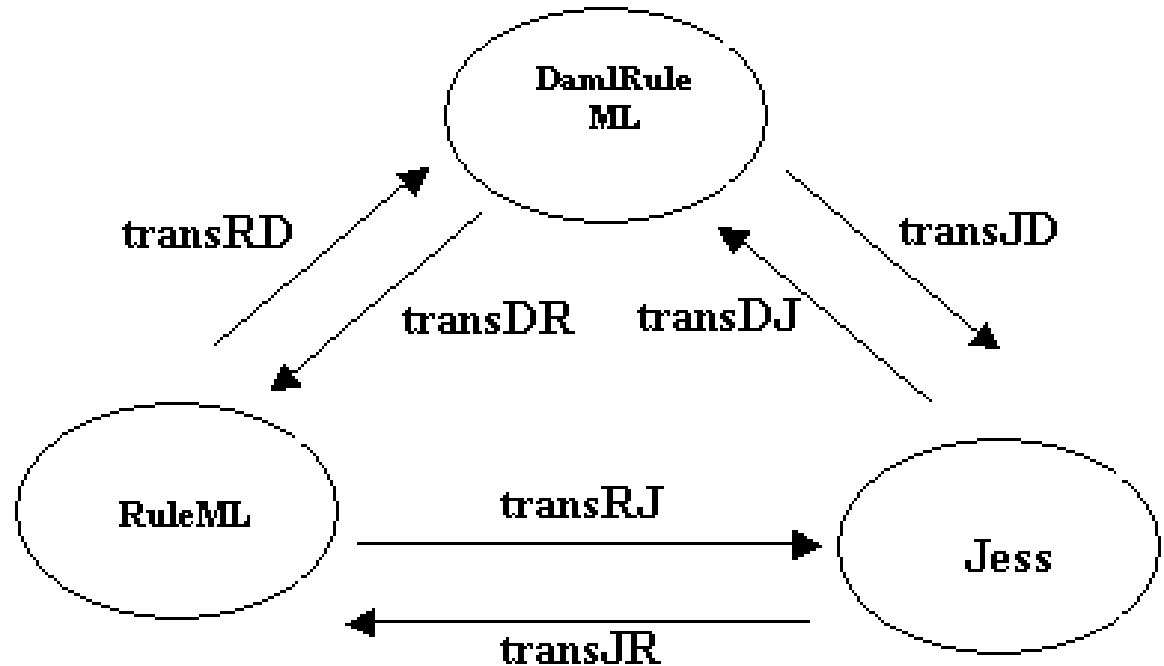
SweetJess

- 2 Aufgaben

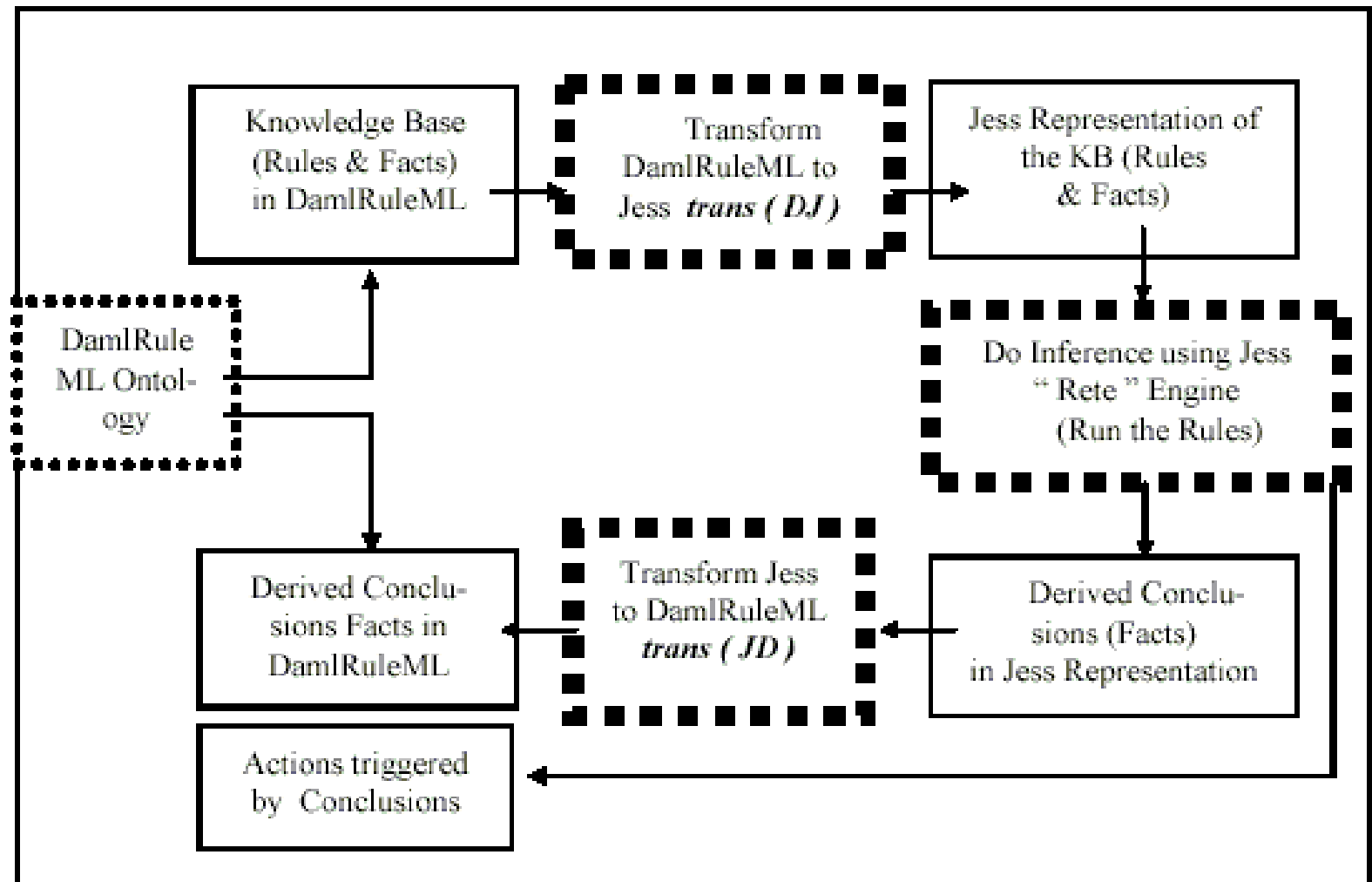
Übersetzer

+

Reasoner



Reasoning via SweetJess



OWRTrans

von OWL in FOL RuleML

family.xml → family.ruleml



OWL2RuleML.xsl

[OWLTrans Homepage](#)

Zusammenfassung

- RuleML
- SWRL
- OWLTrans und SweetJess

Literatur

- RuleML Initiative Homepage <http://www.ruleml.org/>
- SWRL Homepage
<http://www.w3.org/Submission/2004/SUBM-SWRL-20040521/>
- Harold Boley, Said Tabet and Gerd Wagner. Design Rationale of RuleML
- Gerd Wagner, Grigoris Antoniou, Said Tabet, and Harold Boley. The Abstract Syntax of RuleML – Towards a General Web Rule Language Framework
- Benjamin N. Grosz, Mahesh D. Gandhe, Timothy W. Finin. SweetJess: Translating DamlRuleML to Jess

Vielen Dank!