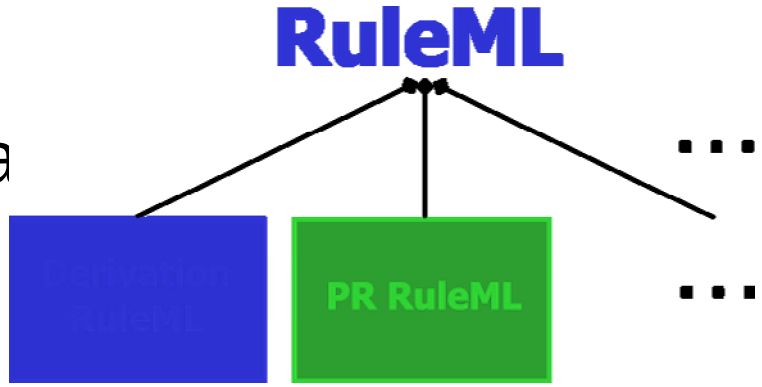
The OO jDREW Reference Implementation of RuleML

RuleML-2005, 10-12 November 2005

Marcel Ball[1], Harold Boley[2], David Hirtle[1,2],
Jing Mei[1,2], Bruce Spencer[2]
[1]University of New Brunswick, Canada
[2]National Research Council of Canada

Contents

1. Modular Schemas

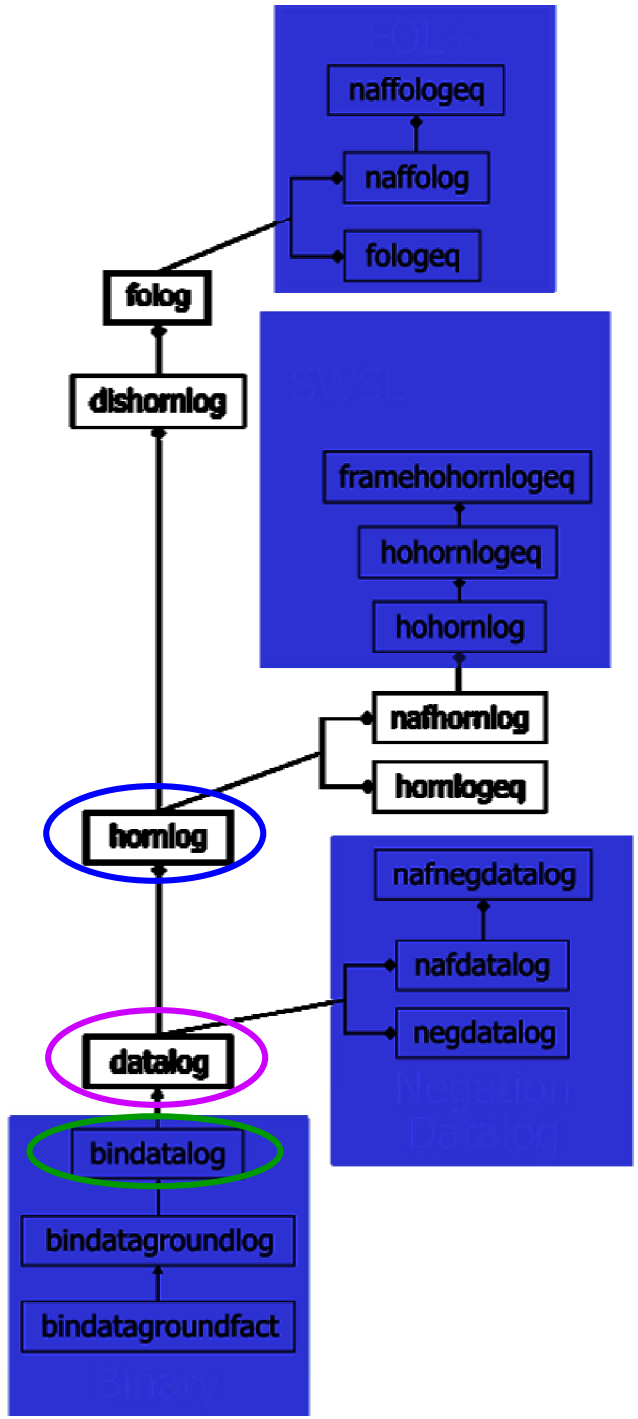


"RuleML is a **family** of sublangua
whose **root** allows access to
the language as a whole and
whose **members** allow to identify
customized subsets of the language."

- RuleML: Rule Markup Language
  - RuleML derivation rules (shown here) and production rules defined in XML Schema Definition (XSD)
  - Each XSD of the family corresponds to the expressive class of a specific RuleML sublanguage
- The most recent schema specification of RuleML is always available at http://www.ruleml.org/spec
- Current release: RuleML 0.89
- Pre-release: RuleML 0.9

1.1 Schema Modulari

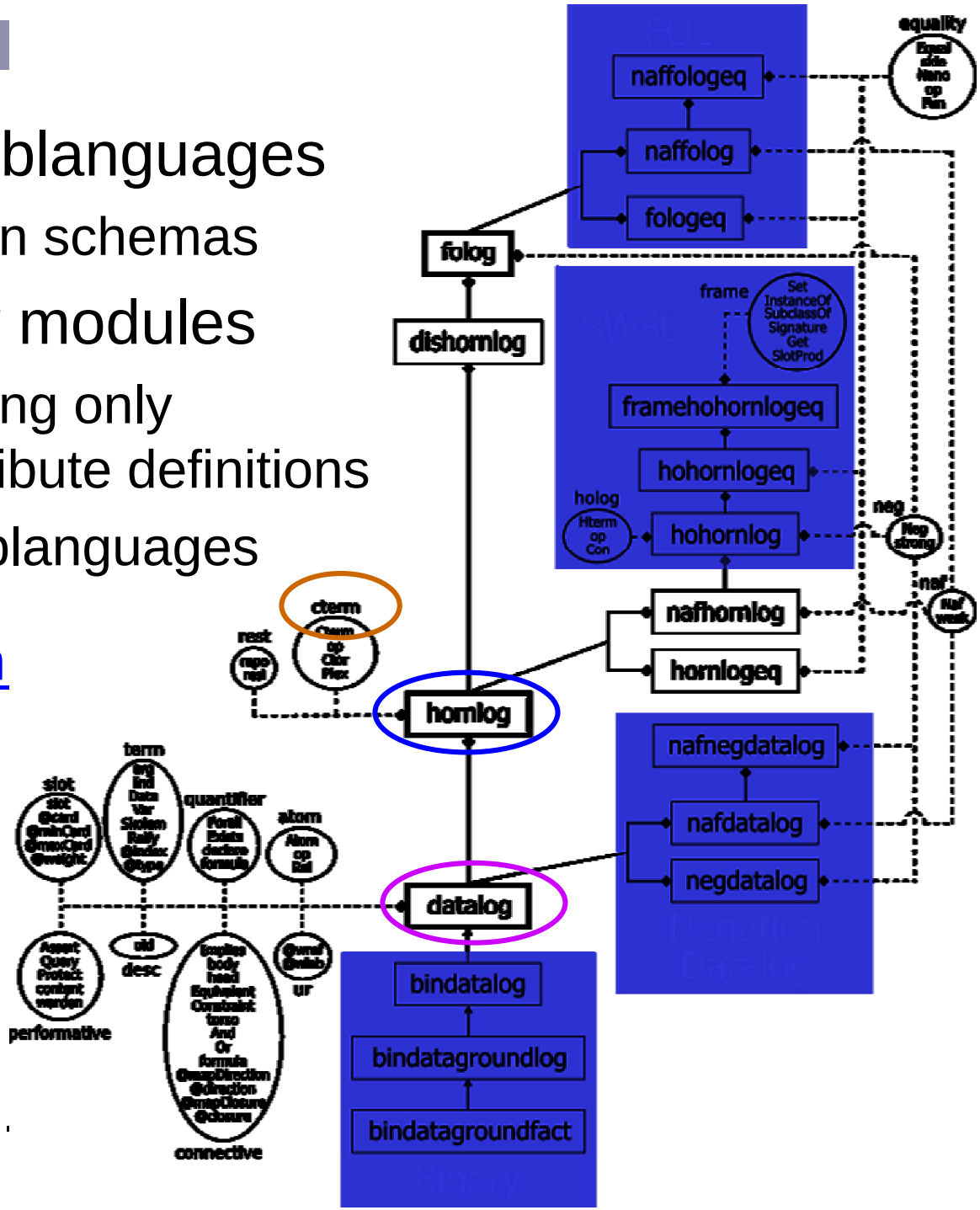
- XSD URIs identify expressive
  - Receivers of a rulebase ca applicability of tools (such as Datalog vs. Horn
  - Associated with semantic (such as function-free vs. Herbrand models)
- Modularization (Official Mode
  - Aggregation: e.g., Datalog *part of* Hornl
  - Generalization: e.g., Bindatalog *is a* Datal



- **Rectangles** are sublanguages
  - Inheritance between schemas
- **Ovals** are auxiliary modules
  - Elementary, including only element and/or attribute definitions
  - Become *part of* sublanguages

E.g., in http://www.ruleml.org/0.9/xsd/hornlog.xsd

```
<xs:redefine
schemaLocation=
"datalog.xsd">
<xs:include
schemaLocation=
"modules/cterm_module.
```



1.2 RDF Rules

- RDF-like Rules: Important RuleML sublanguage
  - Datalog: Relational databases augmented by views
  - RDF Properties: Slots permit non-positional, keyed arguments
  - RDF URIs: Anchors provide **`Ob`**ject **`id`**entity via webzing through URIs
    - **`oid`**s: Can be **`Ind`**ividuals, **`Var`**iables, etc.
    - **`uri`**s: Now used for both RDF's **`about`** and **`resource`**
  - RDF Blank Nodes: F-logic/Flora-2 Skolem-constant approach
    - E.g., Skolem generator '_' becomes **`<Skolem/>`**

“For a product whose price is greater than 200 and whose weight is less than 50, no shipping is billed.”

```
<Implies>
 <body>
  <And>
   <Atom>
    <oid><Var>x</Var></oid>
    <Rel>product</Rel>
    <slot><Ind uri=":price"/><Var>y</Var></slot>
    <slot><Ind uri=":weight"/><Var>z</Var></slot>
   </Atom>
   <Atom>
    <Rel uri="swrlb:greaterThan"/><Var>y</Var><Data>200</Data>
   </Atom>
   <Atom>
    <Rel uri="swrlb:lessThan"/><Var>z</Var><Data>50</Data>
   </Atom>
  </And>
 </body>
 <head>
  <Atom>
   <oid><Var>x</Var></oid>
   <Rel>product</Rel>
   <slot><Ind uri=":shipping"/><Data>0</Data></slot>
  </Atom>
 </head>
</Implies>
```

2. Bidirectional Interpreters in Java

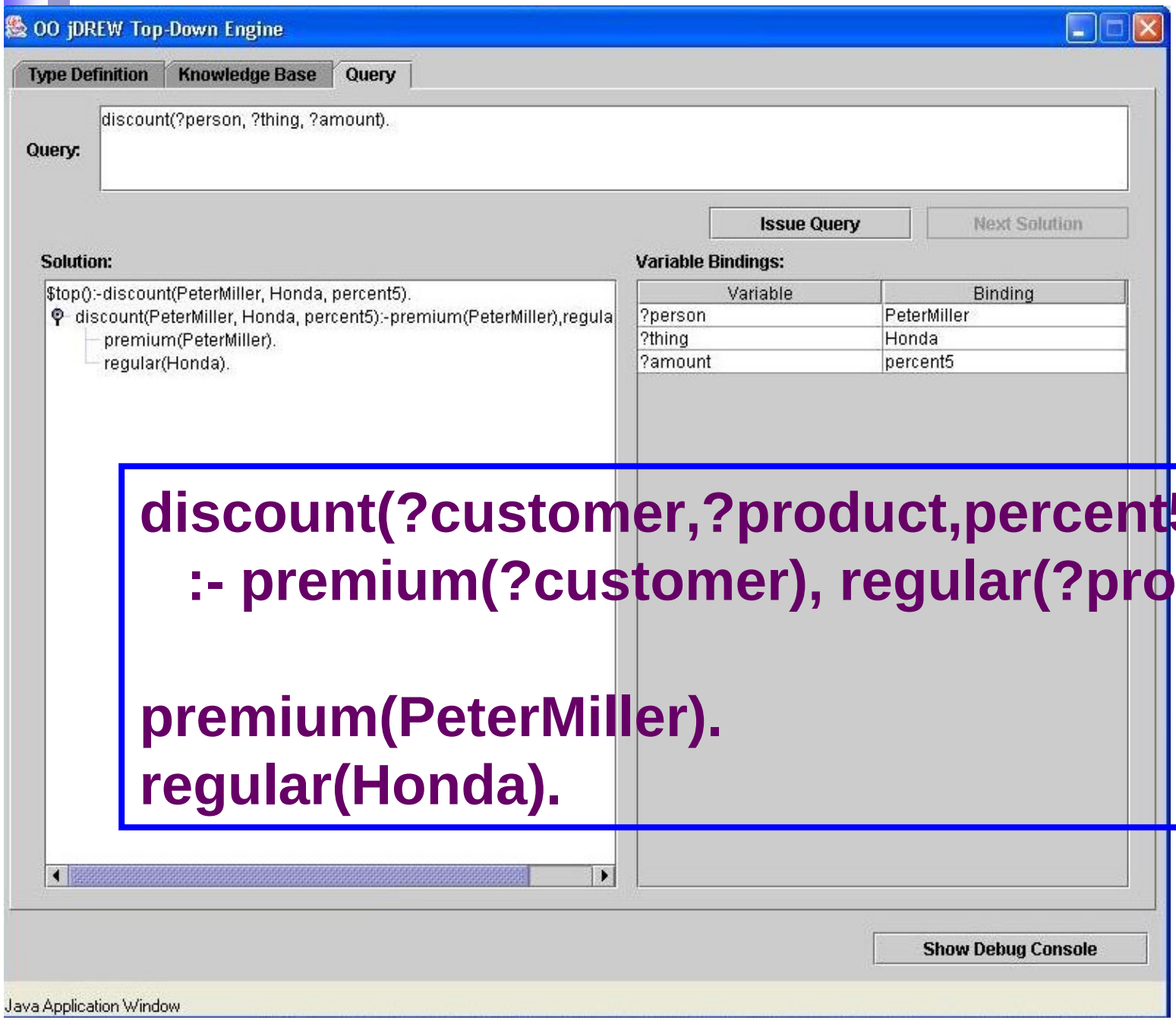
- Two varieties of reasoning engines
  - **T**op-**D**own: backward chaining
  - **B**ottom-**U**p: forward chaining
- jDREW: *Java Deductive Reasoning Engine for the Web* includes both TD and BU
  http://www.jdrew.org
- OO jDREW: *Object-Oriented* extension to jDREW
  http://www.jdrew.org/oojdrew
- Demos: November 11th, 2005; on demand today
  - Java Web Start online demo available at
    http://www.jdrew.org/oojdrew/demo.html

2.1 jDREW Principles

- Utilities:
  - Reading files of RuleML statements into the internal clause data structure
  - Storing and manipulating clauses
  - Unification of clauses according to the positions of the selected literals
  - Basic resolution engine
    - Clause to clause subsumption
    - Clause to clause-list subsumption
  - Choice point managers
  - Priority queues for various reasoning tasks
  - Readable top-level procedures
- Control flow:
  - Oriented around iterators
  - Pay as you go
  - Top-down: choice point to the next solution
  - Bottom-up: generating solutions one-at-a-time

2.2 OO jDREW Slots

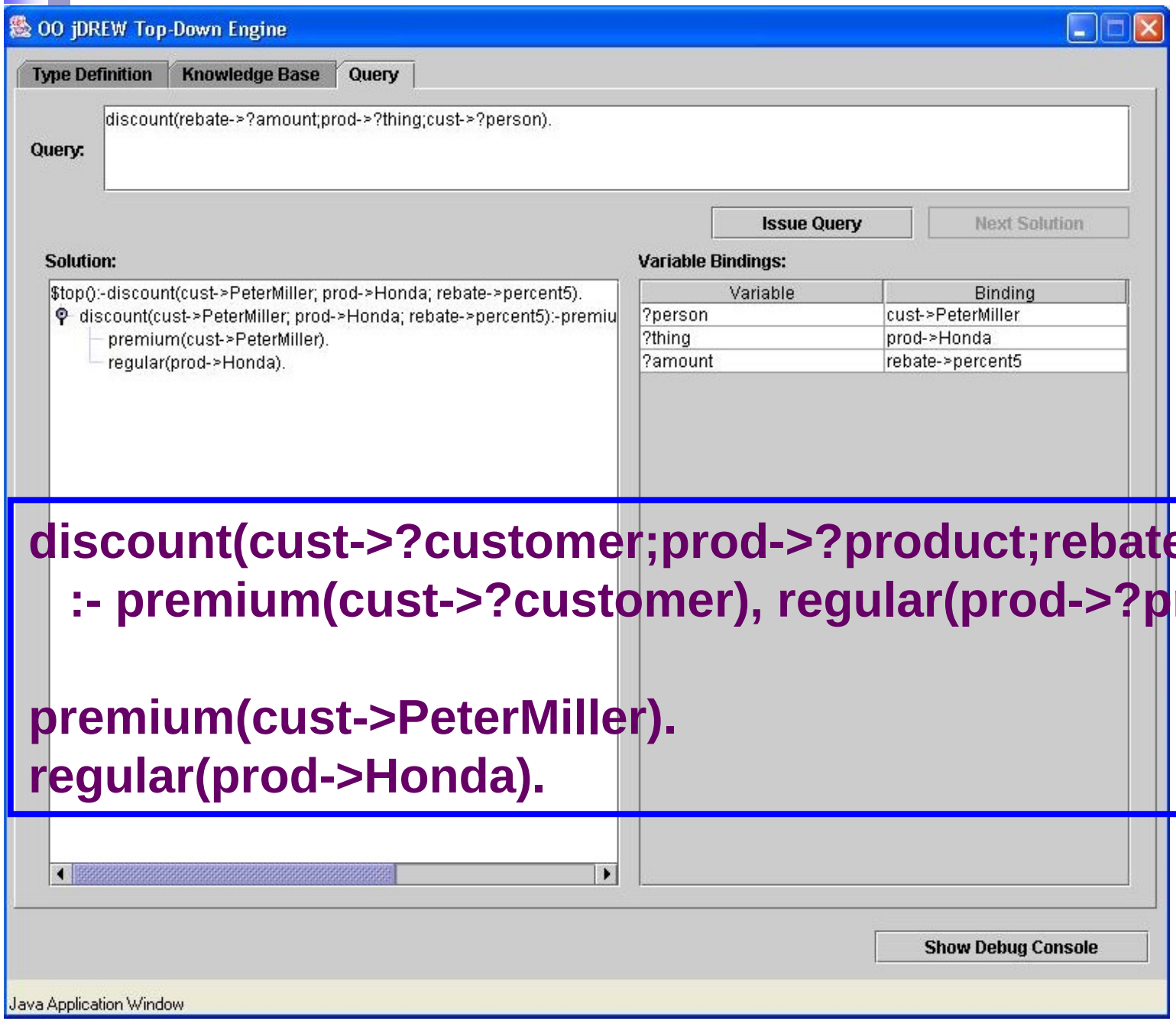
- Normalized atoms and complex terms
  - **`oid`**s (object identifier)
  - Positional parameters (in their original order)
  - Positional rest terms
  - Slotted parameters (in the order encountered)
  - Slotted rest terms
- Efficient unification algorithm
  - Scan two lists of parameters
    - Matching up roles and positions for positional parameters
    - Unifying those parameters
  - Add unmatched roles to list of rest terms
  - Generate dynamically a Plex (RuleML's closest equivalent to a list) for a collection of rest terms



positional

discount(?customer,?product,percent5)
:- premium(?customer), regular(?product).

premium(PeterMiller).
regular(Honda).



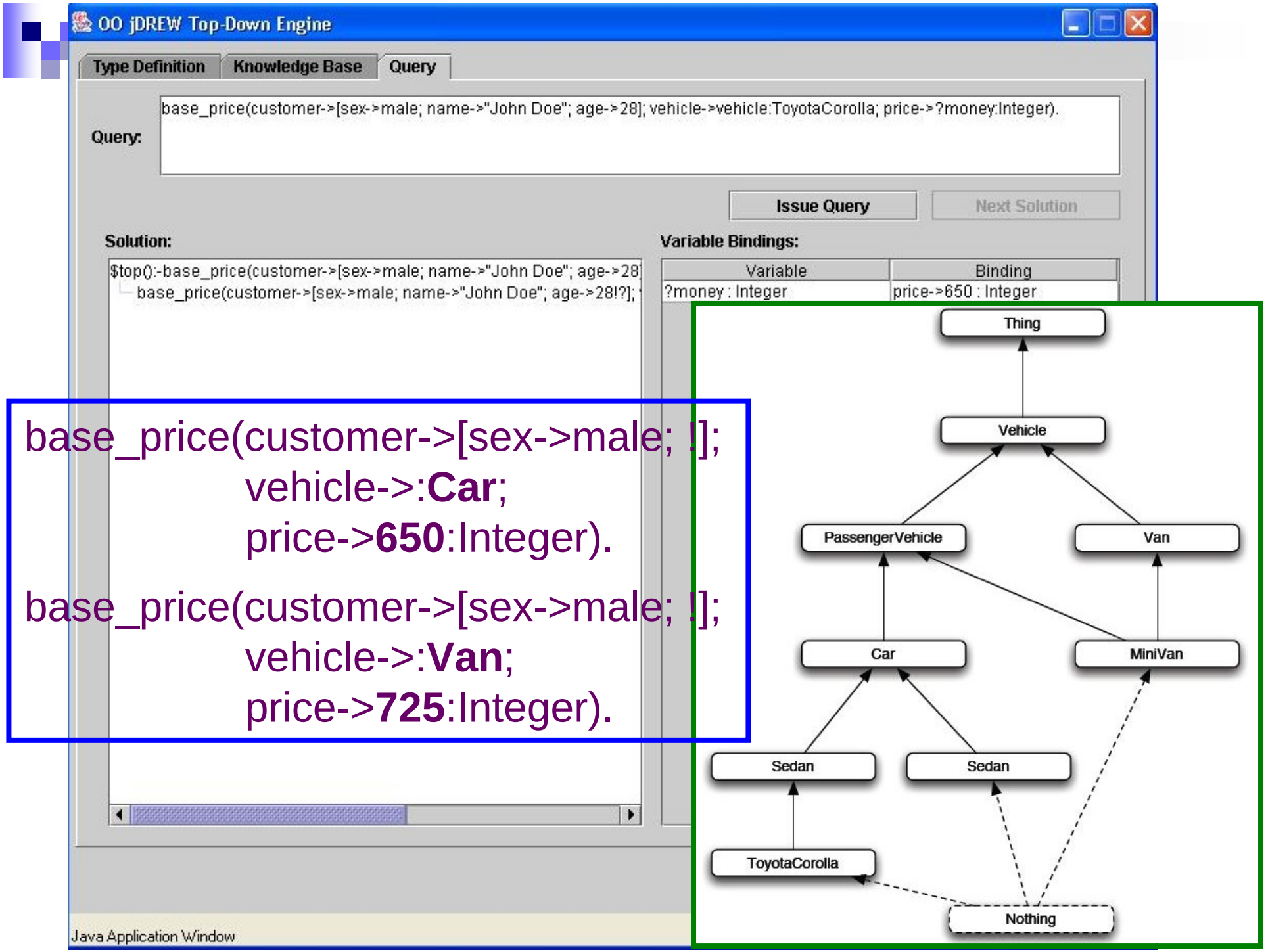
slotted

discount(cust->?customer;prod->?product;rebate->percent5)
:- premium(cust->?customer), regular(prod->?product).

premium(cust->PeterMiller).
regular(prod->Honda).

2.3 OO jDREW Types

- Order-sorted type system
  - RDF Schema: lightweight taxonomies of the Semantic Web
  - To specify a partial order for a set of classes in RDFS
- Advantages
  - Having the appropriate types specified for the parameters
  - To restrict the search space
  - Faster and more robust system than when reducing types to unary predicate calls in the body
- Limitations
  - Only modeling the taxonomic relationships between classes
  - Not modeling properties with domain and range restrictions


OO jDREW Top-Down Engine
Type Definition
Knowledge Base
Query
Query:
base_price(customer->[sex->male; name->"John Doe"; age->28]; vehicle->vehicle:ToyotaCorolla; price->?money:Integer).
Issue Query
Next Solution
Solution:
$top():-base_price(customer->[sex->male; name->"John Doe"; age->28
base_price(customer->[sex->male; name->"John Doe"; age->28!?];
Variable Bindings:
Variable
Binding
?money : Integer
price->650 : Integer
base_price(customer->[sex->male; !];
vehicle->:Car;
price->650:Integer).
base_price(customer->[sex->male; !];
vehicle->:Van;
price->725:Integer).
Thing
Vehicle
PassengerVehicle
Van
Car
MiniVan
Sedan
Sedan
ToyotaCorolla
Nothing
Java Application Window

2.4 OO jDREW OIDs

- **`oid`**: Object Identifier
- Currently: symbolic names
  - In **`<Atom>`** & **`<Implies>`**
- Planned: **`uri`** attribute
- E.g., give name to fact keep(Mary, ?object).

```
<Atom>
  <oid><Ind>mary-12</Ind></oid>
  <Rel>keep</Rel>
  <Ind>Mary</Ind>
  <Var>object</Var>
</Atom>

<Atom>
  <oid><Ind uri="http://mkb.ca"/></oid>
  <Rel>keep</Rel>
  <Ind>Mary</Ind>
  <Var>object</Var>
</Atom>

<Atom>
  <oid><Var>object</Var></oid>
  <Rel>keep</Rel>
  <Ind>Mary</Ind>
  <Var>object</Var>
</Atom>
```

2.5 OO jDREW Extensions

- Negation-as-failure
  - Implemented both in Top-Down and Bottom-Up
- Equality ground facts
  - Mapping all equal individuals to a representative of their equivalence class

```
<Implies>
  <head>
    <Atom>
      <Rel>discount</Rel>
      <Var>customer</Var>
      <Var>product</Var>
      <Ind>5.0 percent</Ind>
    </Atom>
  </head>
  <body>
    <And>
      <Atom>
        <Rel>premium</Rel>
        <Var>customer</Var>
      </Atom>
      <Atom>
        <Rel>onsale</Rel>
        <Var>product</Var>
      </Atom>
      <Naf>
        <Atom>
          <Rel>special</Rel>
          <Var>product</Var>
        </Atom>
      </Naf>
    </And>
  </body>
</Implies>
```

```
<Equal>
  <Ind>fatherOFtom</Ind>
  <Ind>bob</Ind>
</Equal>
<Equal>
  <Ind>fatherOFtom</Ind>
  <Ind>uncleOFmary</Ind>
</Equal>
<Atom>
  <Rel>premium</Rel>
  <Ind>bob</Ind>
</Atom>
<Atom>
  <Rel>onsale</Rel>
  <Ind>clothes</Ind>
</Atom>
<Atom>
  <Rel>special</Rel>
  <Ind>clothes</Ind>
</Atom>

1: premium("uncleOFmary").
1: premium("fatherOFtom").
1: premium("bob").
2: equal("uncleOFmary",uncleOFmary).
2: equal("fatherOFtom",uncleOFmary).
2: equal("bob",uncleOFmary).
2: equal(fatherOFtom,bob).
3: onsale(clothes).
4: special(clothes).
```

```
<Equal>
  <Ind>fatherOFtom</Ind>
  <Ind>bob</Ind>
</Equal>
<Equal>
  <Ind>fatherOFtom</Ind>
  <Ind>uncleOFmary</Ind>
</Equal>
<Atom>
  <Rel>premium</Rel>
  <Ind>bob</Ind>
</Atom>
<Atom>
  <Rel>onsale</Rel>
  <Ind>clothes</Ind>
</Atom>

1: discount("uncleOFmary", clothes, "5.0 percent").
1: discount("fatherOFtom", clothes, "5.0 percent").
1: discount("bob", clothes, "5.0 percent").
2: onsale(clothes).
3: premium("uncleOFmary").
3: premium("fatherOFtom").
3: premium("bob").
4: equal("uncleOFmary",uncleOFmary).
4: equal("fatherOFtom",uncleOFmary).
4: equal("bob",uncleOFmary).
4: equal(fatherOFtom,bob).
```

3. Conclusions

- Concrete & abstract syntax of RuleML
  - Specified by modular XSD (shown here) & MOF
- Operational semantics of RuleML
  - Implemented by OO jDREW BU & TD
- Interoperability of RuleML
  - Realized by translators, primarily via XSLT; see W3C WS version: http://www.ruleml.org/w3c-ws-rules/implementing-ruleml-w3c-ws.html

Thank you

Questions?