A Hands-on PSOA RuleML Tutorial

Relationship & Framepoint Facts and Rules

Theodoros Mitsikas
National Technical University of Athens | RuleML
mitsikas[AT]central[DOT]ntua[DOT]gr

4th International Joint Conference on Rules and Reasoning,
Virtual, June 29th - July 1st, 2020

Available at: http://ruleml.org/talks//TheodorosMitsikas-PSOARuleMLTutorial-RuleMLRR2020.pdf
Updated from: 2nd Workshop on Rules: Logic and Applications (RulesLogApps 2019)

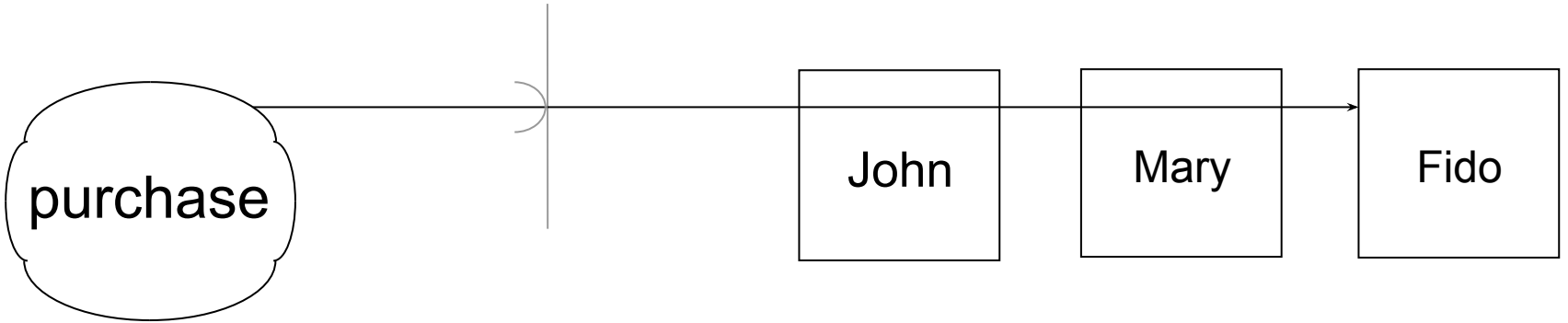
PSOA RuleML

- An **object-relational** Web rule language for defining and querying **relational**-style *table rows* and **object**/node-describing *key-value pairs*
- Integrates spectrum of atomic formulas (atoms), from Prolog-like **relationships** to F-logic-like **frames**, as well as blended ones, in a systematics of **p**ositional-**s**lotted **o**bject-**a**pplicative (psoa) atoms

- Use cases of Knowledge Bases (KBs):
  - Port Clearance Rules, Medical Devices Rules, Air Traffic Control KB, ...

Relationships: From English & RCL to Visualization (Grailog)

English: "A purchase involving John, Mary, and Fido"
RCL: "a purchase of John, Mary, thru Fido"



Relationships: Oidless, Single-Tupled, Dependent Atoms

`purchase(John Mary Fido)` PSOA RuleML

`purchase(john,mary,fido)` Prolog

PSOA RuleML:

- Predicate arguments are separated by whitespace, not by commas
- `John`, ... are individual constants (variables denoted by '?' prefix)
  - constants include `Top` (the root of the predicate hierarchy), numbers, strings, and Internationalized Resource Identifiers (IRIs)

Relationships: Oidless, Single-Tupled, Dependent Atoms

`purchase(John Mary Fido)` PSOA RuleML

- the order of the arguments is significant
- we can have n-ary relationships (here: n=3)
- the argument tuple of a relationship is predicate-dependent (predicate-scope-sensitive)

(Fact) Queries

Ground queries (no variables):

```
purchase(John Mary Fido)     % Yes
```

Non-ground queries (at least one variable, bound to "_"-prefixed explicit local constants):

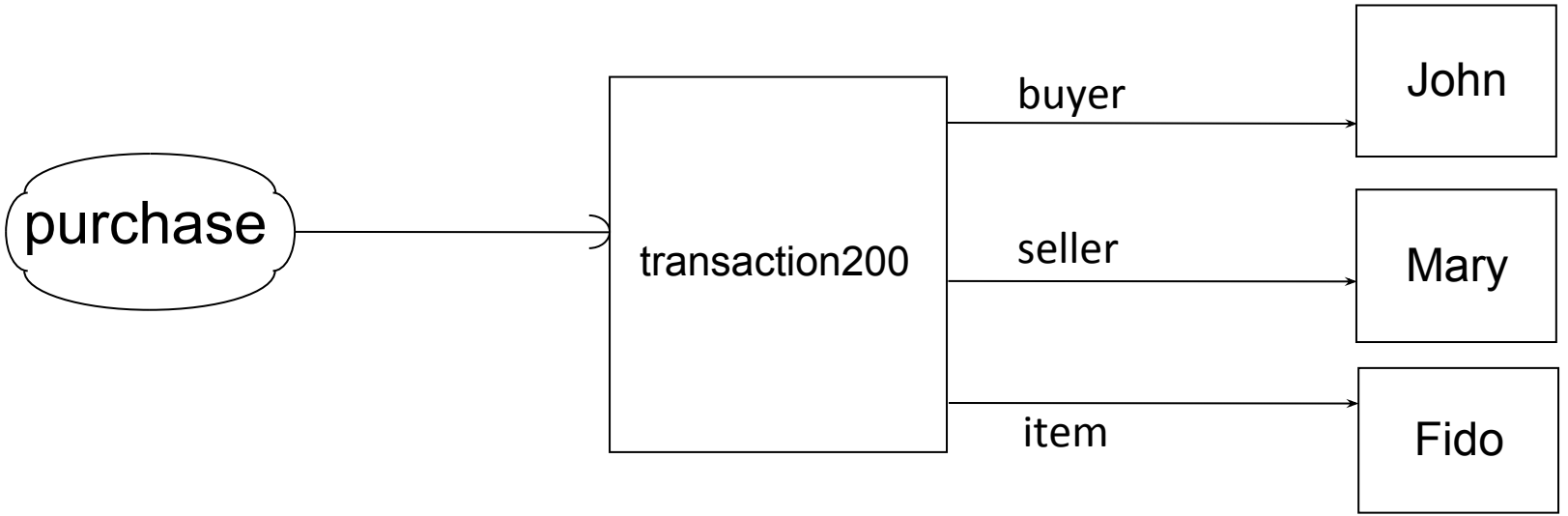
```
purchase(?b ?s ?i) % ?b=_John ?s=_Mary ?i=_Fido

purchase(?b ?s)    % No (there can be no bindings)

?p(John Mary Fido) % ?p=_purchase (predicate variable)
```

Framepoints: From Engl.-like RCL to Visualization (Grailog)

"transaction200, a purchase with buyer John, seller Mary, plus item Fido"



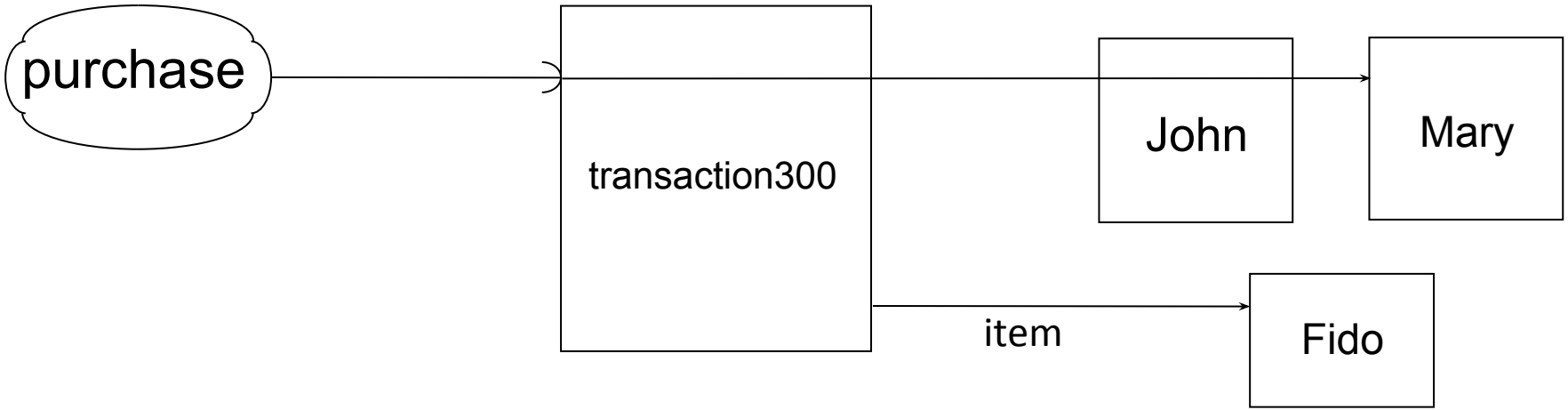
Framepoints: Oidful, Slotted, Independent Atoms

```
transaction200#purchase(buyer->John
                        seller->Mary
                        item->Fido)
```

- hash infix, "`#`", types the Object Identifier (OID) `transaction200` with its predicate (i.e., indicates membership)
- uses slot names ('explicit roles') `buyer`, `seller`, and `item`
- independent-arrow infix, "`->`", pairs a predicate-independent slot name with its filler
- ordering between slots is not important
- framepoint atoms build a Directed Labeled Graph with predicate-typed nodes

Atoms Blending Relationships and Framepoints:
From English-like Templates (RCL) to Visualization (Grailog)

"transaction300, a purchase of John thru Mary, with item Fido"



Tuple/Slot-combining PSOA Atoms in Systematics from Relationships to Framepoints

The atom

```
transaction300#purchase(John Mary item->Fido)
```

is oidful, tupled+slotted

(Ground) Rule over Relationships

"John is liable for Fido if John purchases Fido from Mary"

```
liability(John Fido) :-
  purchase(John Mary Fido)
```

(Non-ground) Rule over Relationships

"A buyer is liable for an item if the buyer purchases the item from a seller"

```
Forall ?b ?s ?i (
  liability(?b ?i) :-
    purchase(?b ?s ?i)
)
```

Hybrid Rule over Relationships and Framepoints

Relationship conclusion, framepoint condition non-ground rule:

```
Forall ?b ?s ?i ?t (
 liability(?b ?i) :-
   ?t#purchase(buyer->?b
               seller->?s
               item->?i)     )
```

Rule over Framepoints

```
Forall ?b ?s ?i ?t (

 liabilityID(?t)#liability(bearer->?b
                           item->?i) :-
     ?t#purchase(buyer->?b
                 seller->?s
                 item->?i)
)
```

Deductive PSOA Queries

```
% KB
transaction200#purchase(
            buyer->John seller->Mary item->Fido)

Forall ?b ?s ?i ?t (
 liabilityID(?t)#liability(bearer->?b item->?i) :-
               ?t#purchase(buyer->?b seller->?s
               item->?i)
)
```

```
liability(bearer->?b item->?i)     % ?b=_John ?i=_Fido

?o#liability(bearer->?b item->?i)

          % Extra binding: ?o=_liabilityID(_transaction200)
```

Advanced PSOA Queries

```
% KB
transaction200#purchase(
            buyer->John seller->Mary item->Fido)

Forall ?b ?s ?i ?t (
 liabilityID(?t)#liability(bearer->?b item->?i) :-
              ?t#purchase(buyer->?b seller->?s
              item->?i)
)
```

```
transaction200#purchase(buyer->?b seller->Mary)     % ?b=_John
transaction200#Top(buyer->?b)          % ?b=_John
transaction200#purchase                % Yes
liabilityID(?t)#liability(bearer->?b) % ?b=_John ?t=_transaction200
Or( ?t#purchase(buyer->Theodore) ?t#purchase(buyer->John) )
                                              %?t=_transaction200
```

Live Demo

- Using PSOATransRun: the reference PSOA RuleML reasoner
- PSOATransRun translates object-relational KBs and queries from PSOA RuleML presentation syntax to relational languages

- Available online: https://psoademo-chatty-cat.eu-gb.mybluemix.net
  - ‘ready-to-use’ demo version translating to TPTP
- Available for download: http://wiki.ruleml.org/index.php/PSOA_RuleML#Prolog_Instantiation
  - runtime options allow to see, e.g., the TPTP or Prolog translation results!

Some (Further) Advanced Features of PSOA RuleML and PSOATransRun

- Built-in mathematical predicates and functions, libraries
- Dependent slots and independent tuples
- Subclasses
- Static translation
- RDF import (N3/Turtle)
- Graph modeling

Further Reading

PSOA RuleML Wiki page:

- http://wiki.ruleml.org/index.php/PSOA_RuleML#Presentation_Preview
- http://wiki.ruleml.org/index.php/PSOA_RuleML#Examples
- http://wiki.ruleml.org/index.php/PSOA_RuleML#References

Learn PSOA RuleML - a resource page on PSOA syntax, (query) semantics, and tools: http://psoa.ruleml.org/learn

Join the Open-source Project

- Develop use cases
  wiki.ruleml.org/index.php/PSOA_RuleML#Use_Cases

- Contribute to PSOATransRun development
  wiki.ruleml.org/index.php/PSOATransRun_Development_Agenda