

Extending an OWL Web Node with Reactive Behavior

Wolfgang May¹ Franz Schenk¹
Elke von Lienen²

Institut für Informatik, Universität Göttingen, Germany

Institut für Informatik, TU Clausthal, Germany

PPSWR 2006, Budva, Montenegro, June 10/11, 2006

Structure of the Talk

- Motivation
- What are triggers and what are they good for in the Semantic Web?
- Differences between events and actions
- Triggers in OWL:

$$\begin{array}{r} \text{simple reactivity} \\ \text{reasoning} \\ + \\ \text{more reactivity} \\ \hline \text{reactive behavior on the semantic level} \end{array}$$

- Implementation
- Conclusion

Motivation

- Use as much existing solutions as possible (be-lazy-approach).
- Extend an information-serving RDF web node with reactive behaviour.
- Integrate this web node into an event-driven semantic web architecture.
- Apply existing work on active rules (from Active Databases) to Semantic Web Application.
- Use existing implementations for reasoning about and handling of rdf/owl-data

Triggers

Simple kind of Active Rules following a simple Event-Condition-Action Pattern:

ON *event* **WHEN** *condition* **DO** BEGIN *action* END

SQL Triggers

Example:

```
ON UPDATE OF actual_schedule
WHEN $NEW.arrival > $OLD.arrival + 30
DO
  BEGIN ... END
```

- *event* is an event in the database
- immediately caused by and identical with an update action

Applications/Use of Triggers

Tasks of triggers:

Local behavior of a node

- consistency maintenance wrt. data model (SQL: e.g. referential integrity after a deletion)
- consistency preservation wrt. application semantics, logging, monitoring

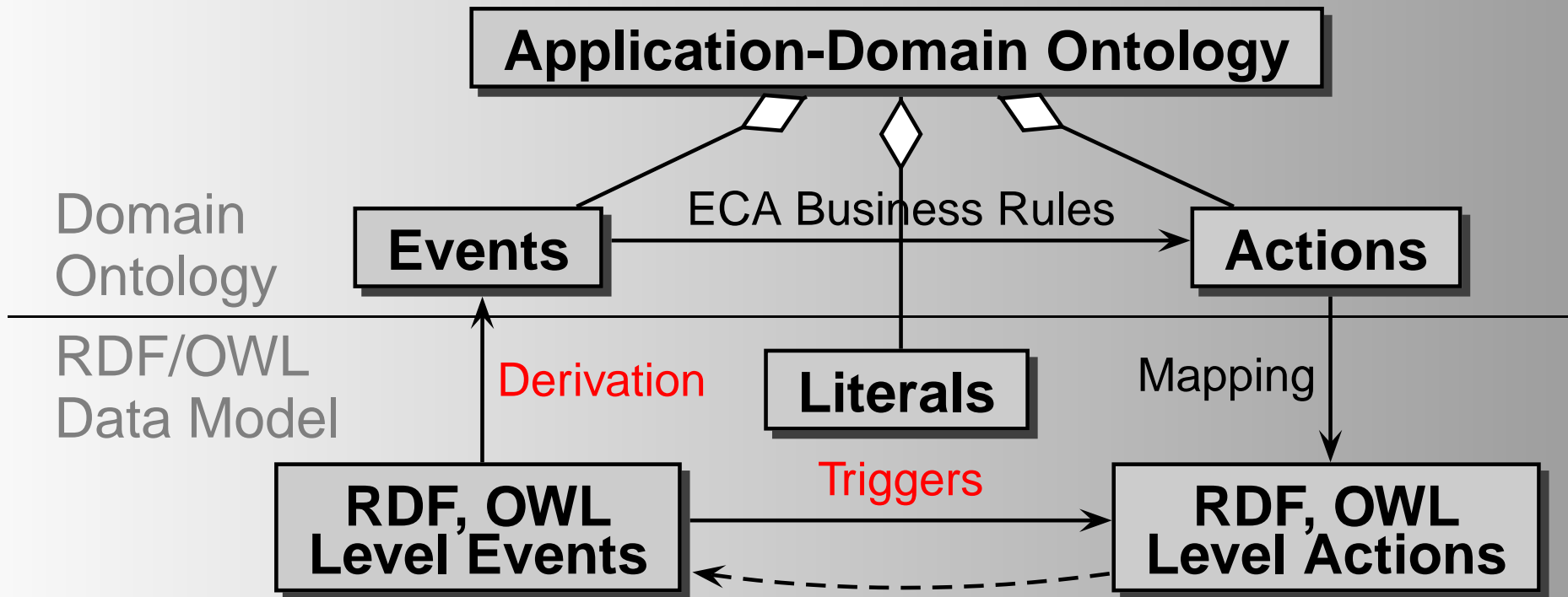
Global

In an Event-Driven Semantic Web Architecture

- raise (=derive) application-level events and make them visible to the outside

Behavior in the Semantic Web

- Domain languages also describe behavior



- triggers needed for **actions+OWL reasoning \rightsquigarrow events**
 - derivation of application-level events
 - triggering OWL-level actions

Updates on the RDF Level

... similar to SQL, under consideration of triple and resource semantics:

- insert a statement
- delete a statement
- update a statement (object, predicate, subject)
- delete a resource (and all statements referring to it)

similar to SQL: events == update actions. RDFS/OWL then adds *inference* to actions ...

Update Operations on OWL Data

- Events on the OWL-Level can be derived events, that result from some (syntactically different) update operation.
- There is no distinction between base and derived relations,
⇒ Redundancy is possible

Additional update operations:

- **RETRACT**(*statement*): intensional update, implemented by delete+triggers; it is considered to be executed successfully, if *statement* does not hold after update
- **ASSERT**(*statement*): intensional update; if a statement already holds, nothing has to be done .

Example (I)

- facts: (hasHusband, inverseOf, hasWife), (Alice, hasHusband, Bob), (Dan, hasWife, Carol), (Emmy, hasHusband, Frank), (Frank, hasWife, Emmy).
- derived: (Bob, hasWife, Alice), (Carol, hasHusband, Dan).
- DELETE(Emmy, hasHusband, Frank) no effect!
- requires a *trigger*

```
ON DELETE OF hasHusband DO  
BEGIN DELETE (OLD:object, hasWife, OLD:subject) END
```

for *supporting an intensional update*

⇒ reacts upon the action

Example (II)

- facts: (hasPresident, rdf:type, owl:FunctionalProperty), (Atlantis, hasPresident, JohnDoe).
- INSERT (Atlantis, hasPresident, ScottTiger) (having also the fact (*ScottTiger differentFrom JohnDoe*))
- model is inconsistent
- requires a trigger

```
ON INSERT OF hasPresident
WHERE {?c hasPresident ?x.} AND
      ?c = NEW.subject AND ?x <> NEW.object
DO BEGIN DELETE (?c, hasPresident, ?x) END
```

for *supporting the update before reasoning*

⇒ reacts upon the action

Example (III)

Mapping Events on the OWL level to the application domain

- if a statement (*dept*, hasProfessor, *prof*) is added to the model, then raise an external event:

```
ON INSERTION OF hasEmployee OF department  
RAISE EVENT  
  (new_employee($object, $subject, $university))
```

- ⇒ reacts upon a change in the model
(i.e., a fact that did not hold before now holds)
- independent which actual update caused this
- uses additional knowledge added by the reasoner
- ⇒ after reasoning

Solution

- ⇒ distinguish between *actions* and *events=changes*
- *pre-reasoning* triggers supporting updates, reacting on intensional update *operations*,
 - *post-reasoning* triggers reacting on actual *changes*.

Pre-Reasoning Triggers

- React on actions, support for intensional updates
- ON {INSERT | UPDATE | RETRACT} OF *property* OF INSTANCE [OF *class*]
- INSTEAD OF {ASSERT | UPDATE} OF *property* OF INSTANCE [OF *class*]
can be used for specifying how to execute an ASSERT or UPDATE instead of straightforwardly materializing the operation.
- remove inverse, care for cardinalities etc.

Post-Reasoning Triggers

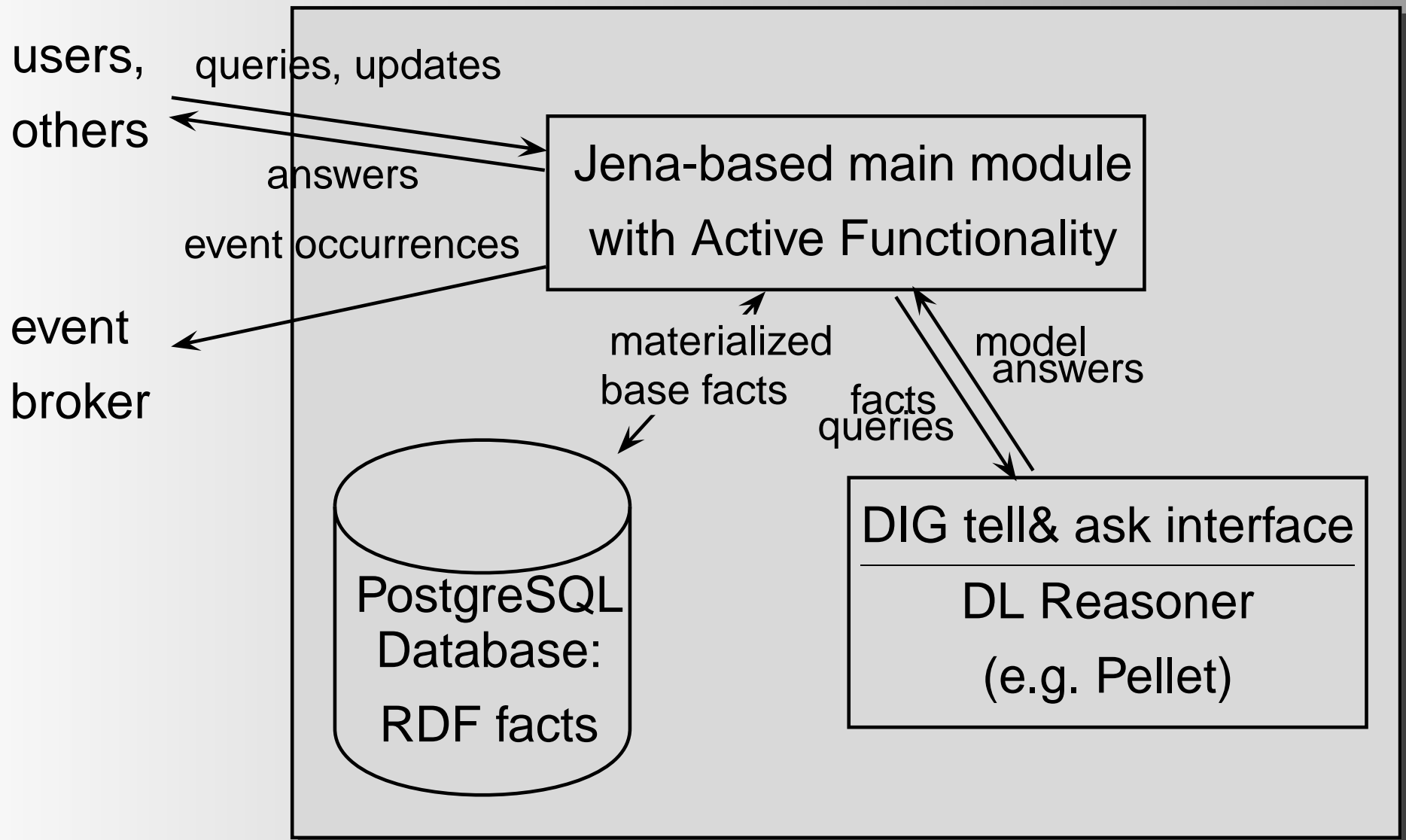
react on actual changes of the model (including derived knowledge)

- ON {INSERTION|MODIFICATION|DELETION} OF *property* OF INSTANCE [OF *class*]
- ON {CREATION|MODIFICATION|DELETION} OF INSTANCE OF *class*
- ON NEW PROPERTY OF INSTANCE [OF *class*]
- ON NEW STATEMENT ABOUT INSTANCE [OF *class*]
- ON NEW CLASS
- ON NEW PROPERTY [OF *class*]

A complete trigger example

```
CREATE TRIGGER test
ON INSERT OF hasPresident
OF INSTANCE OF country
WHEN SELECT $pres
WHERE { $new.subject $new.property $pres . }
DO
BEGIN
delete($new.subject, $new.property, $pres);
END;
```

Architecture of the Application Node



Algorithm

- Work on main model, keep a copy for potential rollback
 - Execute original update and direct triggers on main model
 - iff all direct triggers succeed and the main model is still consistent: proceed;
otherwise rollback and return
 - compute the differences between the models before and after updates+triggers
- ⇒ Set of inserted /deleted statements
- Evaluate all inserted, deleted and updated facts and fire indirect triggers (including raising of events)

Analysis

- Model operations make use only of functionality provided by the Jena RDF-Framework
- computes the whole model and exports it twice,
- computes two differences, iterates over them
- drawback: even if no trigger will fire, these differences are computed.
- but: differences usually small, actual firing of triggers efficient

Alternatives

- reasoning about updates which actual changes will be the result:
 - requires simulating OWL reasoning ,
 - actually the view maintenance problem (view = tuples relevant for the triggering events).
- comparison only of these materialized views (materialize before vs. after only for the views):
 - derive (SPARQL) view query from trigger declaration,
 - answer SPARQL queries twice (before/after) \forall triggers,
 - views are in general large,
 - for a large number of triggers less efficient than the complete difference,
 - optimization: query rewriting/query containment.

Conclusion

- Simple solution.
- Usage of active rules following the ECA-Paradigm.
- Ready-to-use implementation, easy to integrate as a webservice.
- Future work may include different approaches for trigger activation.

The end.