Ontological Reasoning in Information Systems

Petr Křemen
petr.kremen@fel.cvut.cz

Outline

- Design of information systems based on ontologies
  - Integrity Constraints in OWL 2
  - Tool: JOPA
  - Application: StruFail system
- Expressive Queries in OWL 2
  - SPARQL-DL^{NOT} and its visualization
  - Tool: OWL2Query

Motivation

- **relational databases** requires **stable** data model to be stable.
- **ontologies** suitable for **rapidly changing** domains with **heterogenous** knowledge
- Basic question for the second scenario:

How to develop an application on top of an ontology ?

Options for an ontology-backed IS

- a generic „ontology editor/browser“
  - The design does not reflect the structure of the particular ontology at all
  - like Protégé, NeON Toolkit, TopBraid Composer, …
- most IS have domain-specific business logic
  - Specific user interfaces
  - Complex domain-specific computations

Ontology access in IS (in Java)

- low-level (type 1)
  - e.g. OWLAPI, Jena, …
  - Their use in targeted information systems produces **lots of boiler-plate code**

    … error-prone and hard to maintain in large systems
- high-level (type 2): object - ontology mapping
  - e.g. Sommer, Elmo, Jastor, RDFReactor, JAOB, Owl2Java, …
  - Makes assumption on the ontology structure (a „class X has a property Y with range Z“, etc.)
  - Object model is incompatible with OWL semantics.

Our Reqs for the Ontology/Application Interface

- Ontology – application Interface consists of
  - (i) a formal contract between the app and the ontology
  - (ii) an object model that represents this contract
  - (iii) a platform-specific control logic that ensures transactional ontology access,
- … with the following requirements:
  - **contract stability** (be static comparing to the ontology)
  - **contract maintainability** (easy to establish and maintain)
  - **non-restrictive** (full entailment checking and query answering)
  - **validation** (modification of the ontology by the application does not violate the contract)

Our approach = explicit contract

- definition of an explicit formal contract between the application and the ontology based on ***OWL integrity constraints***.
- As the ontology evolves, the contract might be violated at some point:
  - the contract must be adjusted, the object model regenerated, and the application recompiled, or
  - the ontology changes are rolled back.
- Also the contract fixes data format modified by the application.

Overview of SROIQ

- For the sake of compactness, do not consider data properties and use SROIQ instead of OWL2-DL.
- classes, properties, individuals
  - *(∀publishedBy · Institution)* - “all objects published only by institutions”,
  - *(= 1 publishedBy)* - “all objects published by exactly one publisher”.
- axioms, semantics, consistency

SROIQ example

- O1 = {Journal(SMCC),

  Journal ⊑(∀publishedBy · Institution)},
- O2 = O1 ∪ {Journal ⊑ (= 1 publishedBy)},
- O3 = O2 ∪ {publishedBy(SMCC, IEEE)},
- O4 = O3 ∪ {publishedBy(SMCC, IEEE2)}.

===========================

- O3 ⊨ Institution(IEEE)
- O4 ⊨ IEEE = IEEE2

Integrity Constraints

- Closed world semantics to SROIQ defined by DCQ^{NOT} – distinguished conjunctive queries with negation:

- An integrity constraint α is valid w.r.t. ontology O if and only if there is no solution for the DCQ^{NOT} query T (α).

Integrity Constraints Semantics

	α_i	$\mathcal{T}(\alpha_i)$
α_1	$A_1 \sqsubseteq \forall S \cdot A_2$	$A_1(?x) \wedge S(?x, ?y) \wedge not(A_2(?y))$
α_2	$A \sqsubseteq (\leq 1\, S)$	$A(?x) \wedge S(?x, ?y_1) \wedge S(?x, ?y_2) \wedge not(?y_1 = ?y_2)$
α_3	$A \sqsubseteq (\leq n\, S)$	$A(?x) \bigwedge_{1 \leq i \leq (n+1)} S(?x, ?y_i) \bigwedge_{i < j \leq (n+1)} not(?y_i = ?y_j)$
α_4	$A \sqsubseteq (\geq n\, S)$	$A(?x) \wedge not\left(\bigwedge_{1 \leq i \leq n} S(?x, ?y_i) \bigwedge_{i < j \leq n} not(?y_i = ?y_j)\right)$

Integrity Constraints Example

- O1={Journal(SMCC),**$Journal \sqsubseteq (\forall publishedBy \cdot Institution)$**},
- O2=O1 ∪ {**$Journal \sqsubseteq (= 1\ publishedBy)$**},
- O3=O2 ∪ {publishedBy(SMCC, IEEE)},
- O4=O3 ∪ {publishedBy(SMCC, IEEE2)}.

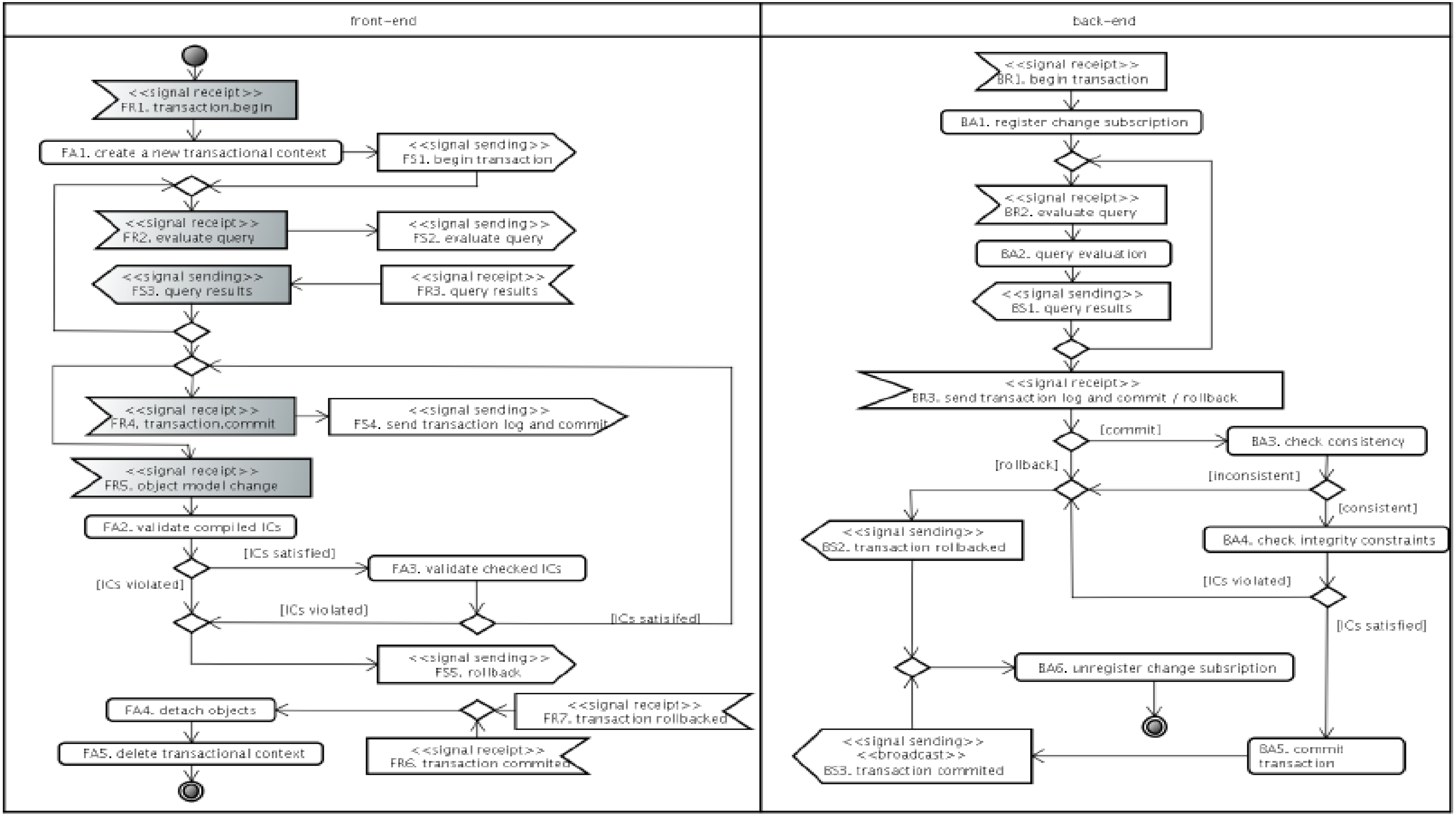
  ==============================

- Violation of ICs:
  - In O2, as no individual is known publisher of SMCC
  - In O3, as IEEE is not known to be an institution
  - In O4, as two institutions being reported as publishers of SMCC, although there must be exactly one.

Integrity constraint types in an IS

- **compile-time** – compiled into the object model
  - A1 ⊑ (∀S · A2) is compiled to a field Set<A2> S; in class A1
  - A ⊑ (≤ 1 S) compiles to a field Object S; in class A
  - easy validation (during compile time)
- **run-time** – optimized in run-time by cheap procedural pre-checks within the object model
  - Whenever A ⊑ (≤ n S) is present, the number of fillers of field O(S) of an instance O(A1 , i) is smaller than n.
- **reasoning-time** – all other
  - passed to the into the DCQ^{NOT} query engine.

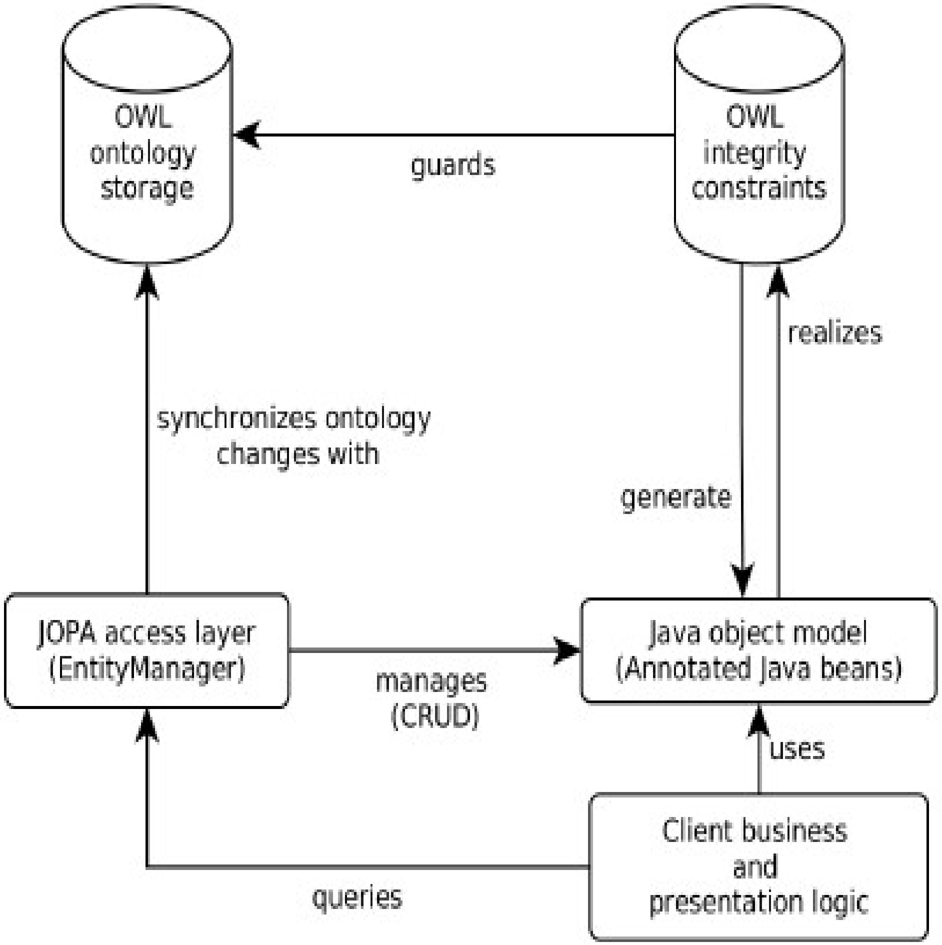
Transaction support



JOPA

- Java OWL Persistence API
- Inspired by JPA 2.0
  - Object model generator based on integrity constraints
  - JPA-like entitymanager API
- Implementation of the proposed system
- http://krizik.felk.cvut.cz/km/jopa

DEMO



OWL2Query

- SPARQL-DL^{NOT}
- Generic SPARQL-DL engine on top of arbitrary OWLAPI reasoner.
- http://krizik.felk.cvut.cz/km/owl2query

DEMO