BRICKS
Building Resources for Integrated Cultural Knowledge Services

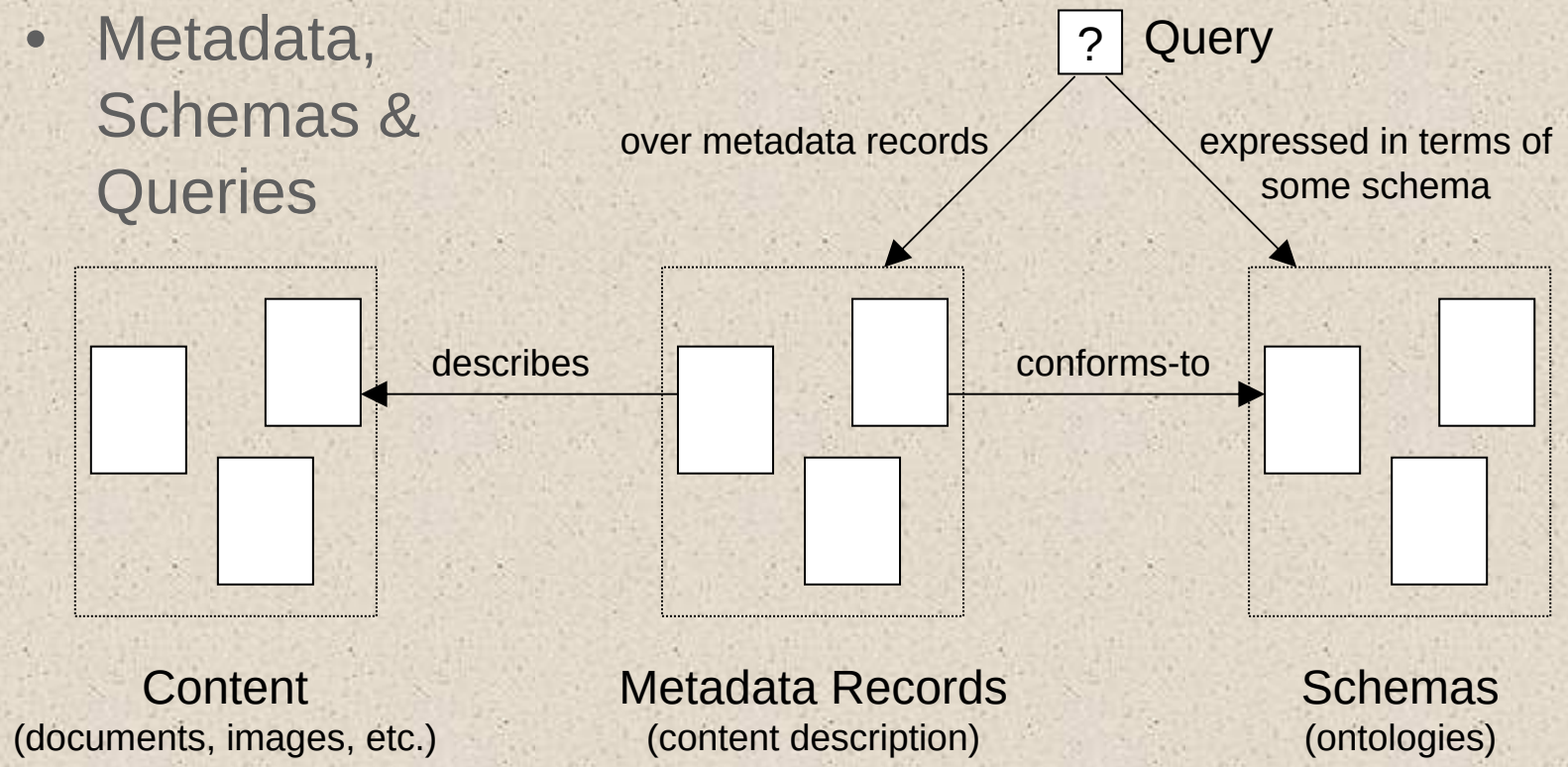
Ontology Mapping in BRICKS

Presentation Overview

- Background Architecture
  - Metadata, Schemas & Queries
- Outline of Ontology Mapping
  - Query Translation
- Implementation Details
  - Mapping Language
  - Use of Prolog

Background Architecture

- Metadata, Schemas & Queries



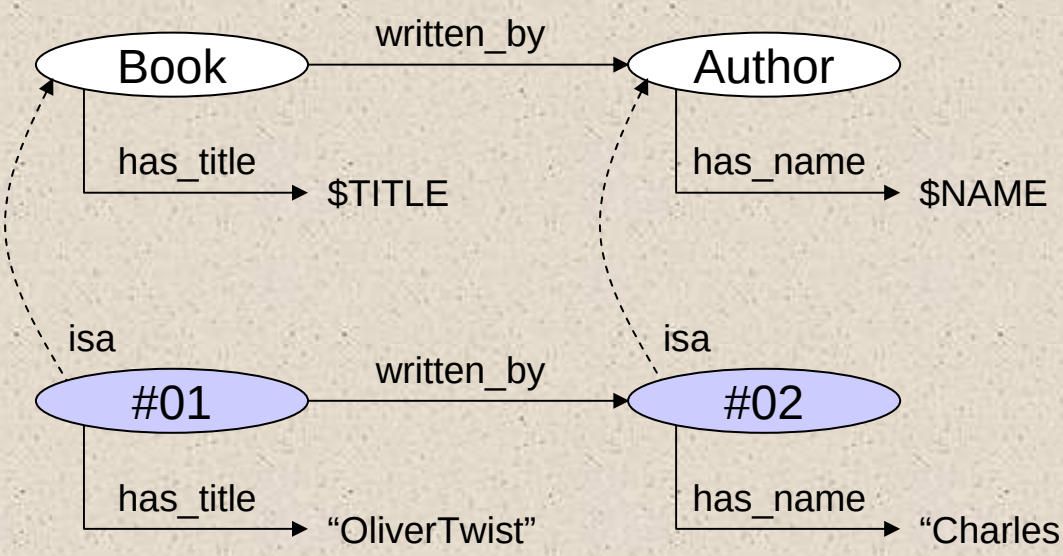
Background Architecture (2)

Query expressed in terms of the schema

?

```
select $Book
where { $Book , written_by , $Author }
and { $Author , has_name , “Charles Dickens” }
```

A Simple Schema ...



& Metadata Record ...

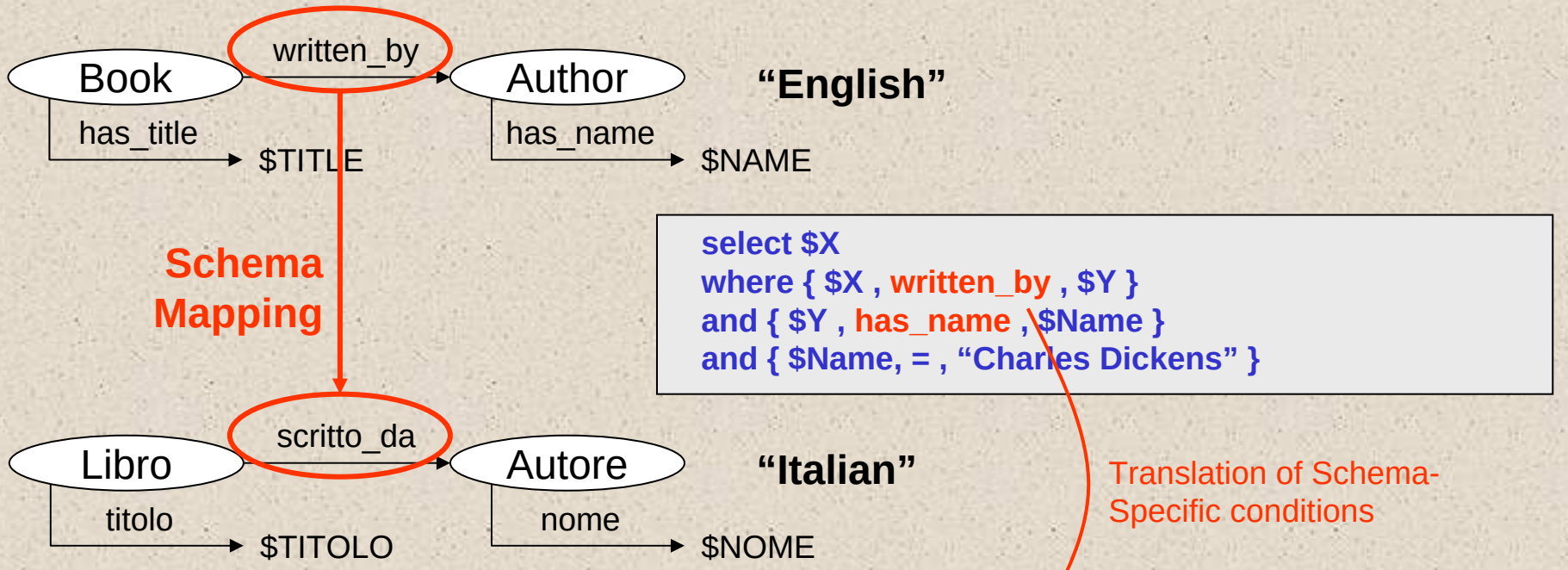
Structure of a Query

- a Query comprises (in brief) :
  - A set of 'conditions' (c.f. phrases in the 'where' clause):
    - **Relation-conditions**:
      { $Book , written_by , $Author }
    - **Instance-of conditions** :
      { $X , $Author } [the entity 'X' is an instanceof the class 'Author']
    - **Value-conditions**:
      { $Name , = , "Charles Dickens" } [operators: =, !=, <, >, >=, <= , ...]
  - A logical structure (boolean and/or hierarchy) over conditions:
    - A *and* (B *or* C) *and* (D *and* (E *or* F)) etc.
      - Where A,B,C,D,E,F are conditions

Outline of Query Translation

- Basic Idea:
  - To translate queries expressed in the terms of one schema, into “semantically equivalent” queries expressed in the terms of another …
  - In short:
    - Rewrite the ‘schema-specific’ parts of the query (the ‘conditions’) – based on user-defined ‘schema-mappings’
    - .. While preserving the logical structure of the query …

Example Query Translation ..



Schema Mapping

- Overarching goal = to translate queries
  - so: Schema Mappings are *query-oriented* …
- Basic form of a mapping assertion :

  IF the query contains the condition “P” ...
  THEN replace “P” with the new condition “Q”
- e.g. “English” to “Italian”:

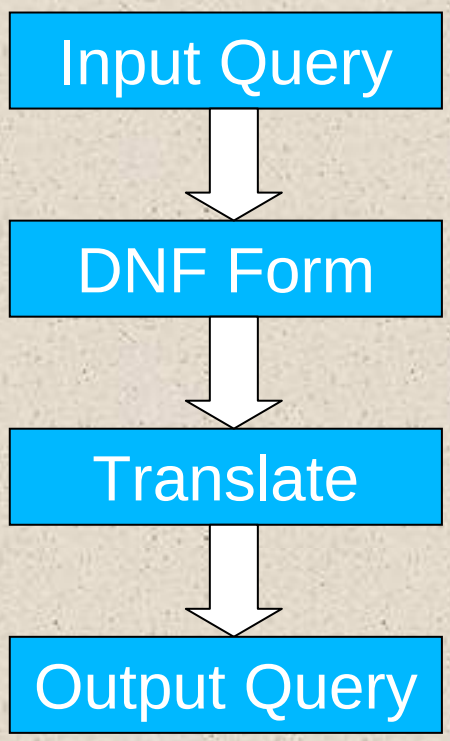
  IF the query contains { $X , written_by , $A } (P)

  THEN replace “P” with { $X , scritto_da , $A }

Mapping Table (English to Italian)

INPUT CONDITION	OUTPUT CONDITION
{ $X , written_by , $A }	{ $X , scritto_da , $A }
{ $X , has_title , $T }	{ $X , titolo , $T }
{ $A , has_name , $N }	{ $A , nome , $N }

Preserving the Logical Structure of Queries



(arbitrary logical structure)

(P_1 AND P_2 AND .. AND P_n) OR (Q_1 AND Q_2 AND .. AND Q_m) ..

Mapping Rules are applied
separately to each conjunction set ...
(P -> R , Q -> S)
preserving the DNF form ...

(R_1 AND R_2 AND .. AND R_n) OR (S_1 AND S_2 AND .. AND S_m) ..

Implementation Details

- Schema Mappings …

  A '*schema mapping*' is a set of rules (i.e. a mapping table) for translating queries expressed over one schema (the source) into queries expressed over another (the destination)

  – Schema mappings are :
    - defined by end-users,
    - encoded in XML
    - stored in the BRICKS distributed XML-storage system

  – A dedicated web-service (SchemaMappingManager) provides mapping management functions.

Example XML Mapping Table

```
<mappings-spec versionId="BRICKS_BML_v1.0">
```

Source & destination schema info

```
<meta>
  <src-info uri="http://www.eng.it/ENGLISH.owl#"/>
  <dst-info uri="http://www.eng.it/ITALIAN.owl#"/>
</meta>
```

List of mapping rules

```
<mapping-rule>
      …
</mapping-rule>
<mapping-rule>
      …
</mapping-rule>
```

```
</mappings-spec>
```

Example XML Mapping Table (2)

```
<mapping-rule>
  <src-expr>
    <rel
      svar=“Book"
      uri="http://www.eng.it/ENGLISH.owl#written_by"
      val="Author"/>
  </src-expr>
  <dst-expr>
    <rel
      svar=“Book"
      uri=" http://www.eng.it/ENGLISH.owl#scritto_da"
      val="Author"/>
  </dst-expr>
</mapping-rule>
```

set of input conditions

set of output conditions

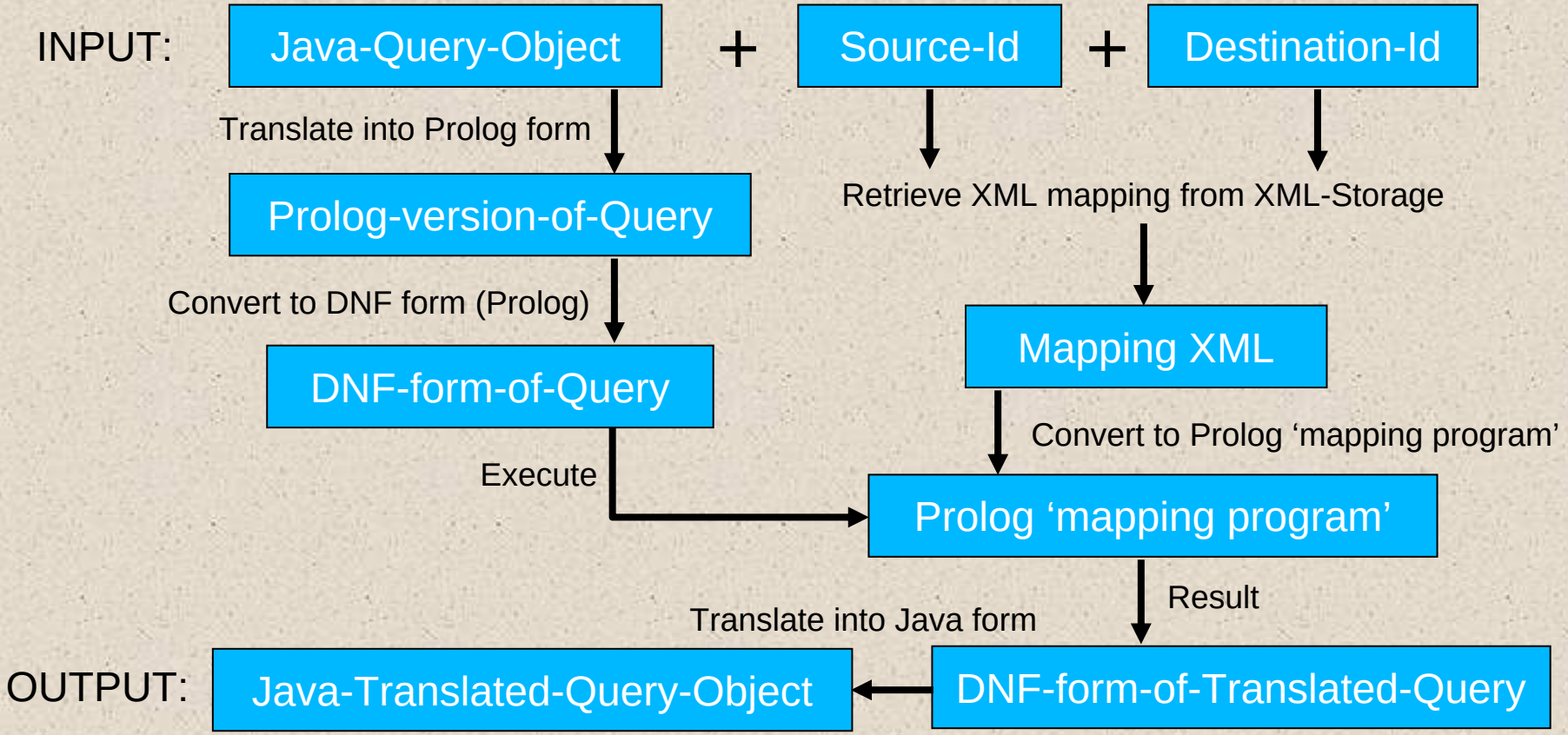
Translation Implemented in Prolog

- The XML ‘Schema Mapping’ defines a set of mapping *rules*,
- These rules are automatically translated into a Prolog ‘mapping program’
  - e.g. each rule takes the Prolog form:

```
mapping(List_of_translated_conditions) :-
        Input_Condition1,
        Input_Condition2,
        ...
```

- The original query is converted to Prolog form, and ‘executed’ against the ‘mapping program’ …

Overview of Translation Process



Complex Mappings

- Built-in Prolog predicates can be included to handle more complex mappings:
  - E.g. String concatenation:

    { $Author , first_name , $First }
    and { $Author , last_name, $Last }

    mapped-to:

    { $Author , full_name , $Name }
    atom_concat($First , $Last , $Name)

Complex Mappings (2)

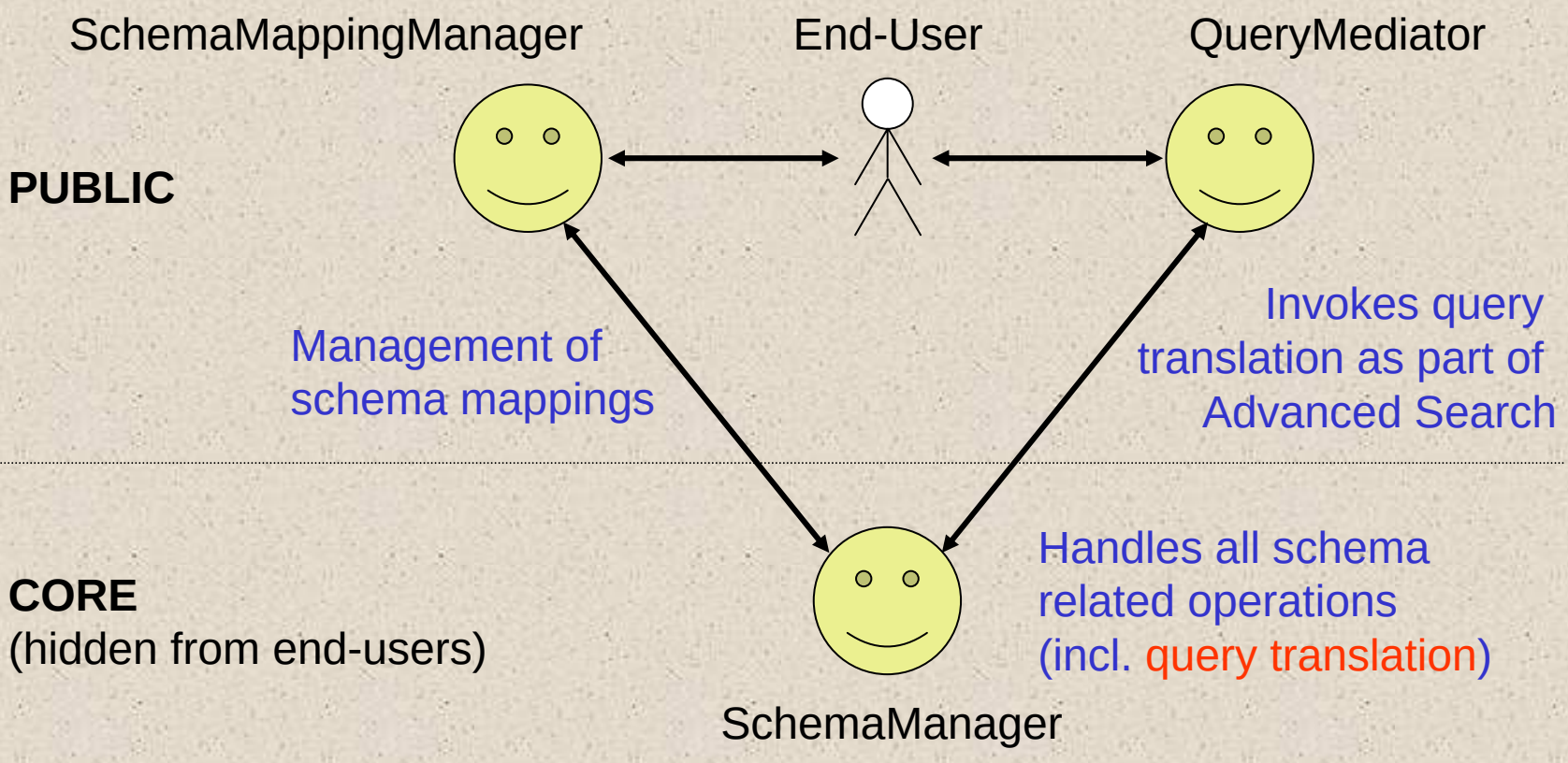
- Mappings can be defined as *operator-specific*:

- { $Document , year_of_publication , $Year }
  { $Year , =, “2006” }
  mapped-to:
  { $Document , publication_date , $Date }
  { $Date, >= , 01-01-2006 } and { $Date, <= , 31-12-2006 }

  but:

  { $Document , year_of_publication , $Year }
  { $Year , >, “2005” }
  mapped-to:
  { $Document , publication_date , $Date }
  { $Date, >= , 01-01-2006 }

Use of Query Translation in BRICKS



Implementation Status

- "Beta" version
  - All the query translation functions described have been fully implemented & tested in simulation
    - Still waiting on feedback from "real world" trials
  - SchemaMappingManager is a working BNode service
  - Integration with the Query Mediator is in progress

- Future work
  - The mapping system is limited by the Query Language
    - e.g. at present the query language is non-functional (does not allow operations over variable values)
  - Functional extensions to the Query Language will permit more complex schema mappings
    - Which the Prolog back-end can already accommodate