INTEGRATION BY REASONING

Horticulture application | Jack Verhoosel & Barry Nouwt

See poster #17

CONTEXT AND PROBLEM

Increase in number of distributed data sources leads to need by organisations for:

- **Combination** of **data** from various **sources** to improve on efficiency of business processes and productivity
- The **secure sharing** of data to gain **better insight** into the entire chain providing there is a **mutual benefit**.
- The **alignment** of the **semantics** of the terms in the different data sources.
- **Analysis** of combined data to discover **trends** and **patterns** in the data to guide the organization.
- Possibilities to ask **complex questions** on the combined set of data.

These are still challenges to be tackled!

SOLUTION DIRECTION

- Commonly used approach:
  - **transform all the data** into linked data triples into a central graph database
  - this central graph database then contains the combination of available data
  - can be **queried as a single** data sources
- Disadvantages:
  - **updates** are needed every time new data becomes available to represent the latest status
  - feasibility of a **large, single central** database is questionable

- Our approach:
  - based on semantic technology solution with data **kept at distributed data sources**
  - data is **only retrieved when needed** to answer a user query

SEMANTIC AND TECHNICAL CHALLENGES

- Data is inherently **different in meaning** and made accessible via **various types** of technical interfaces.
- Our solution is based on:
  - A **common OWL ontology** that contains the concepts that represent the data used from the distributed sources, made available via the Apache Jena Fuseki triple store with a SPARQL engine in front.
  - End users can query the common ontology on all of its concepts providing maximum **querying flexibility**.
  - Retrieval of specific data from distributed sources is done using a set of **customized rules** defined in terms of the concepts and relations in the ontology as well as **built-in functions** that call the data sources APIs.
  - A reasoner is triggered to **execute these rules** when a user issues a SPARQL query on the ontology and necessary data needs to be retrieved.
  - We used Apache Jena's **generic rule reasoner** to execute this rule-based reasoning integration approach.

RELATED WORK

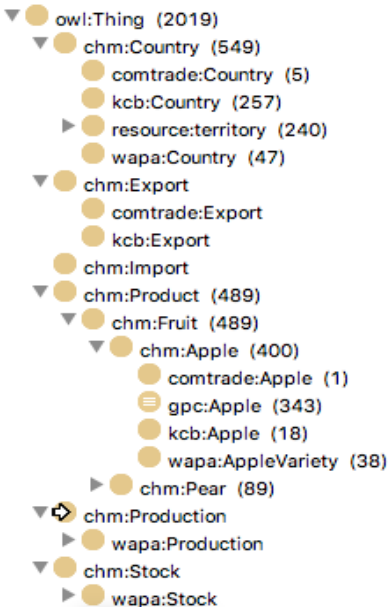
- Data Integration
  - Similar problem, **different technology**: SQL and RDBMS
- Ontology-based Integration:
  - Uses **ontology matching** to integrate multiple data sources
- Semantic Web Service Descriptions
  - Complementary technology
- Web Service Composition
  - Uses the same AI planning techniques
  - No on-the-fly **planning**
  - Starting point: Composite web service versus **SPARQL query**

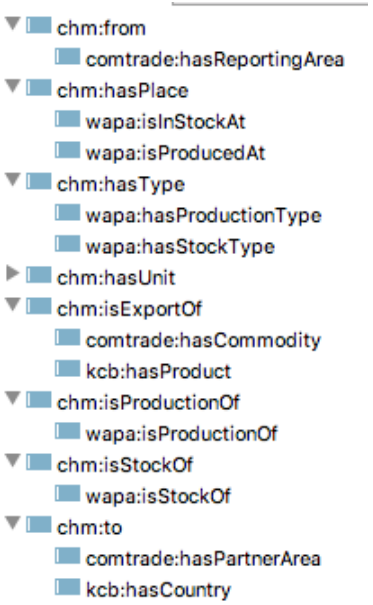
VALIDATION BY APPLICATION

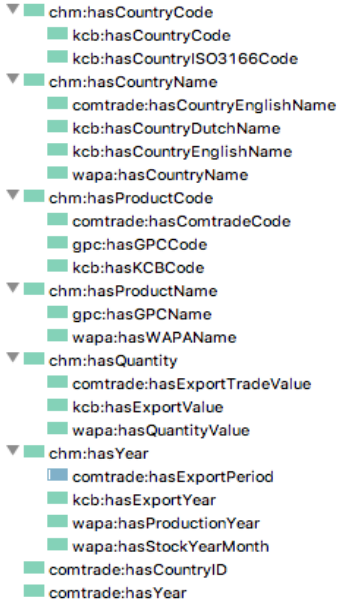
- Horticulture domain:
  - Various distributed data sources **along the food supply chain**
  - **Import/export transactions**, production (forecast) data, stock data, parcel crop data, etc.
- HortiCube based on our Knowledge Engine platform, 3 data sources:
  - Apple/pear yearly **production** forecast figures per variety per EU country. A WAPA (World Apple and Pear Association) source transformed from CSV format to RDF using the LODRefine tool and a WAPA ontology, made available as linked data in a triplestore at one of our servers.
  - Apple/pear monthly **stock** per variety per EU country. A WAPA source transformed to RDF made available similarly as the WAPA production dataset.
  - **Import/export** between countries in the EU of apples/pears from the UN Comtrade data source available via an external API provided by the UN (https://comtrade.un.org/data/).

ALIGNMENT OF DATASOURCES

- Common Horticultural Model
- Mapping with data sources:
  - owl:subClassOf
  - owl:subPropertyOf
- Countries and apple/pear instances are part of CHM
- Not the actual production, stock and export data!!
- Specific user query:
  “What was the trade value of apples that were exported from The Netherlands to Belgium in 2014?”







RULE-BASED REASONING

- Rule: `[mortality-rule: (?x rdf:type :Man) -> (?x rdf:type :Mortal)]`
- Data: `:socrates rdf:type :Man`

Forward or data-oriented reasoning:

- Triples after: `:socrates rdf:type :Man .` `:socrates rdf:type :Mortal .`

new!

Backward or goal-oriented reasoning:

- Start with goal/query: `?var rdf:type :Mortal .`
- Proof:
  - Rule `mortality-rule` concluded `:socrates rdf:type :Mortal .` from:
    - Fact `:socrates rdf:type :Man .`

new!

BUILT-INS

- Normal built-in usage
  - `[rule: (?n rdf:type :Man)(?n :cmLength ?o) -> CmInch(?o ?p)(?n :inchLength ?p)]`
- Our built-in usage:
  - `[rule: (?n rdf:type :Man)(?n :cmLength ?o) -> WebService(?o ?p)(?n :inchLength ?p)]`
- Why?
  - Benefit from existing **rule planning** mechanisms
  - (**On-the-fly**) external data source integration
  - Graph patterns allow **complex** mappings.

EXPERIMENTS

Specific user question:

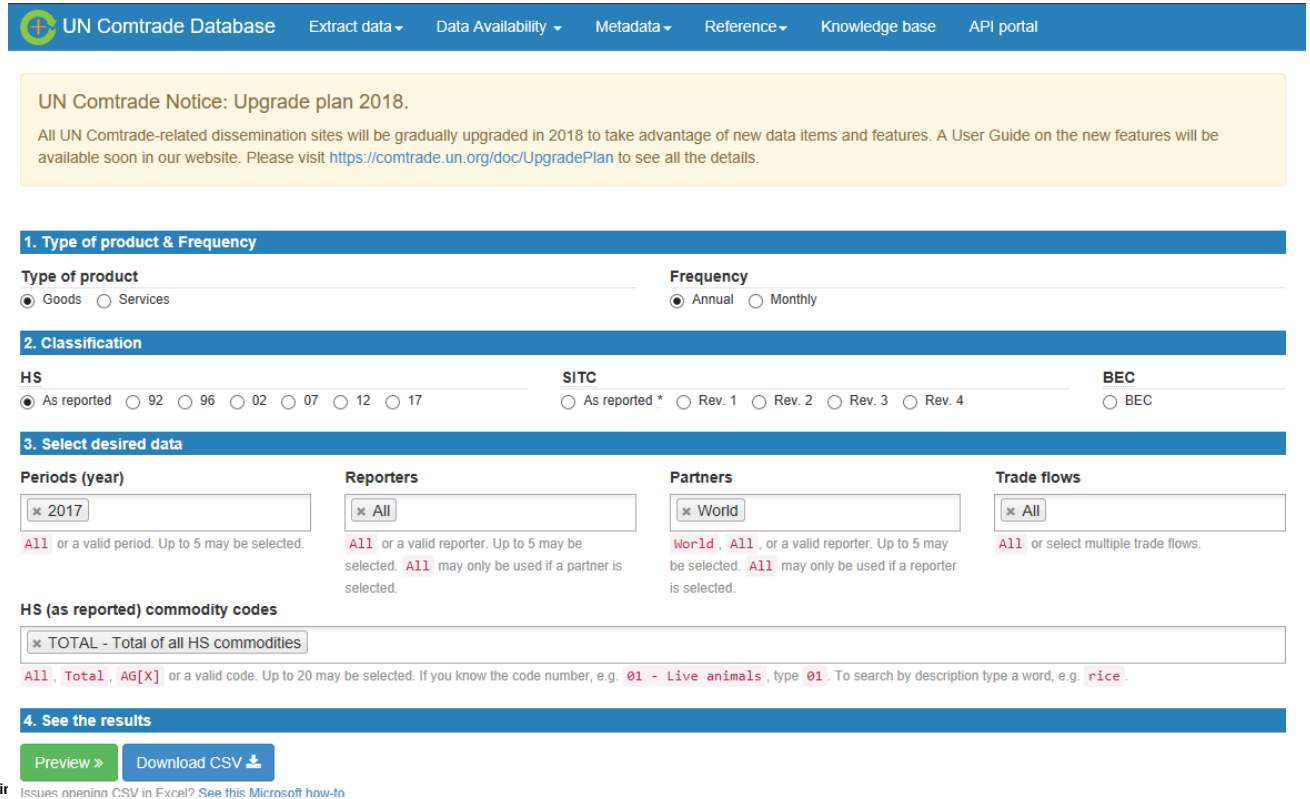
“What was the trade value of apples that were exported from The Netherlands to Belgium in 2014?”

Translates into SPARQL query on CHM:

```
PREFIX chm: <http://www....

SELECT ?uri ?reportingArea ?tradeValue
WHERE {
      ?uri rdf:type chm:Export .
      ?uri chm:from ?reportingArea .
      ?reportingArea chm:hasCountryName "Netherlands" .
      ?uri chm:to ?partnerArea .
      ?partnerArea chm:hasCountryName "Belgium" .
      ?uri chm:hasYear ?year .
      ?year comtrade:hasYear "2014"^^xsd:gYear .
      ?uri chm:isExportOf ?product .
      ?product chm:hasProductCode "080810" .
      ?uri chm:hasQuantity ?tradeValue .
}
```

UNITED NATIONS COMTRADE API



RULE AND BUILT-IN DEVELOPMENT

```
[comtradeExportForwardRule:
        (?exportingcountry rdf:type comtrade:Country)
        (?exportingcountry comtrade:hasCountryEnglishName ?exportingcountryname)
        (?exportingcountry comtrade:hasCountryID ?exportingcountryvalue)
        (?importingcountry rdf:type comtrade:Country)
        (?importingcountry comtrade:hasCountryEnglishName ?importingcountryname)
        (?importingcountry comtrade:hasCountryID ?importingcountryvalue)
        notEqual(?exportingcountry,?importingcountry)
        (?product rdf:type comtrade:Apple)
        (?product comtrade:hasComtradeCode ?productvalue)
        (?year rdf:type comtrade:Year)
        (?year comtrade:hasYear ?yearvalue)
        uriConcat(comtrade,'#',"Export_",?exportingcountryvalue,'_',?importingcountryvalue,'_',?productvalue,'_',?yearvalue,?uri)
        comtradeExport(?exportingcountryvalue, ?importingcountryvalue, ?productvalue, ?yearvalue, ?export)
->
        (?uri rdf:type comtrade:Export) (?uri comtrade:hasExportTradeValue ?export)
        (?uri comtrade:hasReportingArea ?exportingcountry) (?uri comtrade:hasPartnerArea ?importingcountry)
        (?uri comtrade:hasCommodity ?product) (?uri comtrade:hasExportPeriod ?year)
]
```

RULE AND BUILT-IN DEVELOPMENT

- Reuse of **existing RDFS rules** to allow the reasoner to apply the defined mapping.

- subClassOf:
  - [rdfs9: (?x rdfs:subClassOf ?y), notEqual(?x,?y) -> [(?a rdf:type ?y) <- (?a rdf:type ?x)]]

- subPropertyOf:
  - [rdfs6: (?p rdfs:subPropertyOf ?q), notEqual(?p,?q) -> [(?a ?q ?b) <- (?a ?p ?b)]]

RULE AND BUILT-IN DEVELOPMENT

comtradeExport(?exportingcountryvalue, ?importingcountryvalue, ?productvalue, ?yearvalue, ?export)

```
public class ChmExport extends BaseBuiltin {
        …
        public String getName() {
                return "comtradeExport";
        }

        public int getArgLength() {
                return 5;
        }
        …
        public boolean bodyCall(Node[] args, int length, RuleContext context) {
                //next slide
        }
}
```

Built-in name

Number of arguments

API call

RULE AND BUILT-IN DEVELOPMENT

Get parameters

```
Node sourceVar = getArg(0, args, contex
Object sourcename = sourceVar.get iteralValue();
Node countryVar = getArg(1, a gs, context);
Object countryname = countryVar.getLiteralValue();
Node productVar = getArg(2, args, context);
Object productname = productVar.getLiteralValue();
Node yearVar = getArg(3, args, context);
Object yearname = yearVar.getLiteralValue();

Client client = ClientBuilder.newClient();
WebTarget target = client.target(String.format(comtradePath,
sourcename, yea ame, countryname, productname));
Response r = target.r st().get();
```

Call Comtrade API

Process API Response

```
JsonReader jsonReader = Json.createRea ((InputStream)
r.readEntity(InputStream.class));
JsonObject result = jsonReader.rea Object();
JsonArray dataset = result.getJsonArray("dataset");
JsonObject row = dataset.iterator().next().asJsonObject();
int exportValue = row.getInt(JSON_FIELD_NAME_TRADEQUANTITY);
jsonReader.close();

Node exportVar = getArg(4, args, context);
Node exportVal = NodeFactory.createLiteral(String.valueOf(exportValue),
XSDDatatype.XSDinteger);
BindingEnvironment be = context.getEnv();
return be.bind(exportVar, exportVal);
```

Bind export value to variable

EXPERIMENTS

Specific user question:

"What was the trade value of apples that were exported from The Netherlands to Belgium in 2014?"

Translates into SPARQL query on CHM:

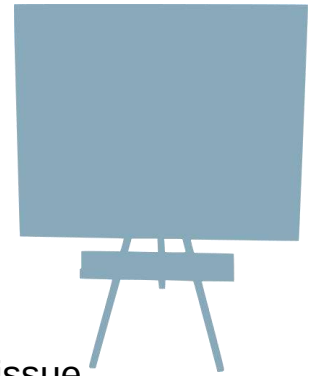
```
PREFIX xsd: <http://www....

SELECT ?uri ?reportingArea ?tradeValue
WHERE {
      ?uri rdf:type chm:Export .
      ?uri chm:from ?reportingArea .
      ?reportingArea chm:hasCountryName "Netherlands" .
      ?uri chm:to ?partnerArea .
      ?partnerArea chm:hasCountryName "Belgium" .
      ?uri chm:hasYear ?year .
      ?year comtrade:hasYear "2014"^^xsd:gYear .
      ?uri chm:isExportOf ?product .
      ?product chm:hasProductCode "080810" .
      ?uri chm:hasQuantity ?tradeValue .
}
```

	uri	reportingArea	tradeValue
1	<comtrade#528560808102014>	<http://bigtu-comtrade-instances#_Netherlands>	"17998814"^^xsd:integer

LESSONS LEARNED

- Performance of backward vs forward reasoning
  - Goal-driven **backward reasoning** is preferable
  - Tests with instances with 5 countries, 1 apple variety and 1 year only
  - Response time = **3 seconds** backward instead of **75 seconds** forward.
  - Less rule executions and external API only called **1 instead of 20 times**



- Ordering and variable bindings
  - **Ordering** of clauses in **SPARQL query** determines rule execution! => end-user issue
  - **Ordering** of clauses in the **rules** determines correctness of reasoning result! => designer issue

- Design effort
  - Multiple ontologies, mappings, different rules and custom built-ins
  - Requires effort, but enables generic, flexible access to distributed data sources

FUTURE CHALLENGES/IDEAS

- Cascading of various reasoners,
  e.g. OWL ontology reasoner on top of the rule-based reasoner to infer OWL axioms
- Handy features for reasoners, such as:
  - **aggregated** built-in calls that combines multiple built-in calls into a single API call
  - support of **multiple heads** in backward reasoning would make rules more efficient.
  - reasoners could provide information about **missing facts** to reach a particular goal.
- **Generic custom built-in** that uses existing **web services discovery** techniques to select appropriate web services, based on their semantic descriptions.
- Include **parallelism** where a reasoning process can be **interrupted for a while** due to an unresponsive external data source so other derivations can continue.

BACKWARD RULE LIMITATION

Only a single triple allowed!

```
[comtradeExportBackwardRuleSplit6:
        (?uri comtrade:hasExportTradeValue ?export)
<-
        (?exportingcountry rdf:type comtrade:Country)
        (?exportingcountry comtrade:hasCountryEnglishName ?exportingcountryname)
        (?exportingcountry comtrade:hasCountryID ?exportingcountryvalue)
        (?importingcountry rdf:type comtrade:Country)
        (?importingcountry comtrade:hasCountryEnglishName ?importingcountryname)
        (?importingcountry comtrade:hasCountryID ?importingcountryvalue)
        notEqual(?exportingcountry,?importingcountry)
        (?product rdf:type comtrade:Apple)
        (?product comtrade:hasComtradeCode ?productvalue)
        (?year rdf:type comtrade:Year)
        (?year comtrade:hasYear ?yearvalue)
        uriConcat(comtrade,'#',"Export_",?exportingcountryvalue,'_',?importingcountryvalue,'_',?productvalue,'_',?yearvalue,?uri)
        comtradeExport(?exportingcountryvalue, ?importingcountryvalue, ?productvalue, ?yearvalue, ?export)
]
```

CONCLUSION

- Problem:
  - query answering over multiple heterogeneous data sources
- Solution:
  - using rules, built-ins and a reasoner to map external data sources onto the ontology
- Application:
  - answer query about the horticultural domain using the Comtrade API

- Future:
  - more complex use case
  - find/build better suited reasoner

See poster #17