Realizing a Semantic Web Application

Emanuele Della Valle
Dario Cerizza
Irene Celino

Center of Excellence For Research, Innovation, Education and industrial Lab partnership - Politecnico di Milano

http://www.cefriel.it
http://swa.cefriel.it
emanuele.dellavalle@cefriel.it
http://emanueledellavalle.org
Goal

- We will “develop” together an application of the Semantic Web we named **Music Event Explorer** or simply **meex**
- We will challenge the Semantic Web technologies in realizing a new service for Web users
  - Using
  - Transforming and
  - Combining existing data

For more info please visit [http://swa.cefriel.it](http://swa.cefriel.it) and [http://emanueledellavalle.org](http://emanueledellavalle.org)!
Ingredients

- **RDF** as unified data model
- **OWL** as modelling language for the data sources
- **GRDDL** as a standard approach to translate in RDF the data stored in XML data sources
- **D2RQ** as tool to translate in RDF the data stored in relational data sources
- **SPARQL** as standard query language to access RDF data
- **Jena** as application framework to merge the various data in a single RDF model and manipulate it
- **Joseky** as tool to expose SPARQL endpoint
- **ARQ** as SPARQL client library
- A **RDF storage** to guarantee persistency
- A **OWL reasoner** to infer new knowledge
- **Exhibit** as user interface

For more info please visit [http://swa.cefriel.it](http://swa.cefriel.it) and [http://emanueledellavalle.org](http://emanueledellavalle.org)!
In order to realize meex

1. We start from the user need
2. We derive user requirements
3. We develop the ontologies and the software components

While presenting we will explain the use of Semantic Web technologies and tools. (green background slides)

A demonstrative installation of the application, together with the source code, is available at

http://swa.cefriel.it/meex
Towards a Semantic Web

- Ivan Herman in introducing the Semantic Web* explains: the current Web represents information using
  - natural language (English, Hungarian, Chinese,...)
  - graphics, multimedia, page layout

- Humans can process this easily
  - can deduce facts from partial information
  - can create mental associations
  - are used to various sensory information
    - (well, sort of... people with disabilities may have serious problems on the Web with rich media!)

* [http://www.w3.org/People/Ivan/CorePresentations/IntroThroughExample/](http://www.w3.org/People/Ivan/CorePresentations/IntroThroughExample/)

For more info please visit [http://swa.cefriel.it](http://swa.cefriel.it) and [http://emanueledellavalle.org](http://emanueledellavalle.org)!
Towards a Semantic Web

- Tasks often require to combine data on the Web:
  - hotel and travel information may come from different sites
  - searches in different digital libraries
  - etc.

- Again, humans combine these information easily
  - even if different terminology's are used!
Towards a Semantic Web

- However: machines are ignorant!
  - partial information is unusable
  - difficult to make sense from, e.g., an image
  - drawing analogies automatically is difficult
  - difficult to combine information automatically
    - is `<foo:creator>` same as `<bar:author>`?
    - how to combine different XML hierarchies?
  - ...

For more info please visit http://swa.cefriel.it and http://emanueledellavalle.org!
Imagine the users need to explore music events related to a given music style

- An event is a concert, a show or a workshop at which one or more artist participate.
- An artist is either a single musician or a band.

For instance, if a user is interest in Folk music meex
- finds the artists that play Folk music
- searches for events of those artists
- allows the users to explore the events related to each artist as a list, on a time line and on a map
A manual solution

1. I open musicmoz [1] and I look up artists that play Folk music

2. If the pages of the artists on musicmoz don’t satisfy me I navigate to musicbrainz [2]

3. I look up in EVDB [3] if some of those artists have organized an event close to my location in these days

4. I take note of the possible alternatives and I check how to get there using google maps [4]


For more info please visit http://swa.cefriel.it and http://emanueledellavalle.org!
A manual solution

1. I look up artists that play Folk music

For more info please visit http://swa.cefriel.it and http://emanueledellavalle.org!
2. I can learn more navigating to musicbrainz

For more info please visit http://swa.cefriel.it and http://emanueledellavalle.org!
3. I look up in EVDB if some of those artists have organized an event close to my location in these days.
4. I take note of the possible alternatives and I check how to get there using google maps.
Of course I can do it manually, but I need the time to do so. Can’t I write a mash-up?

For more info please visit http://swa.cefriel.it and http://emanueledellavalle.org!
Ivan Herman in introducing the Semantic Web* explains:

- (Some) data should be available for machines for further processing.
- Data should be possibly combined, merged on a Web scale.
- Sometimes, data may describe other data (like the library example, using metadata).
- ... but sometimes the data is to be exchanged by itself, like my calendar or my travel preferences.
- Machines may also need to reason about that data.

---

* [http://www.w3.org/People/Ivan/CorePresentations/IntroThroughExample/](http://www.w3.org/People/Ivan/CorePresentations/IntroThroughExample/)
The rough structure of data integration

1. Map the various data onto an abstract data representation
   - make the data independent of its internal representation...

2. Merge the resulting representations

3. Start making queries on the whole!
   - queries that could not have been done on the individual data sets

For more info please visit [http://swa.cefriel.it](http://swa.cefriel.it) and [http://emanueledellavalle.org](http://emanueledellavalle.org)!
The rough structure of data integration

Applications

Data represented in abstract format

Data in various formats

Query, Manipulate, etc.

Map, Expose, etc.

For more info please visit http://swa.cefriel.it and http://emanueledellavalle.org!
The Semantic Web provides technologies to make such integration possible! For example:

- an **abstract model** for the relational graphs: **RDF**
- **extract** RDF information from XML (eg, XHTML) pages: **GRDDL**
- **add** structured information to XHTML pages: **RDFa**
- a **query** language adapted for the relational graphs: **SPARQL**
- **characterize** the relationships, categorize resources: **RDFS, OWL, SKOS, Rules**
  - applications may choose among the different technologies
  - some of them may be relatively simple with simple tools (RDFS), whereas some require sophisticated systems (OWL, Rules)
- **reuse** of existing “ontologies” that others have produced

For more info please visit [http://swa.cefriel.it](http://swa.cefriel.it) and [http://emanueledellavalle.org](http://emanueledellavalle.org)!
So where is the Semantic Web?

Data in various formats

Data represented in RDF, possibly with extra knowledge (RDFS, OWL, SKOS, Rules, …)

Applications

SQL <=> RDF, GRDDL, RDFa etc.

SPARQL, OWL inferences, etc.

For more info please visit http://swa.cefriel.it and http://emanueledellavalle.org!
A Semantic Web application is still an application, thus we need to follow good practice from Software Engineering in developing it.

- We adopt a Spiral Model inspired by the famous Boehm spiral model.
- We extend it with Knowledge Engineering practices.

For more info please visit http://swa.cefriel.it and http://emanueledellavalle.org!
R.1 Users’ needs analysis

R.2 Risk analysis

R.3 Software requirements analysis

D.1 Model the application ontology

Reuse ▼ Merge ▼ Extend

D.2 Model the content ontology

Reuse ▼ Merge ▼ Extend

D.3 Model sample contents

D.4 Design Application

I.1 Implement the initial Knowledge Base

I.2 Implement the integrated model

I.3 Choose content annotation methods

I.4 Implement the application

T.1 Testing
R.1 Users’ needs analysis

R.2 Risk analysis

R.3 Software requirements analysis

R.4 Content requirements analysis

D.1 Model the application ontology

D.2 Model the content ontology

D.3 Model sample contents

D.4 Design Application

D.4 Design Application

I.1 Implement the initial Knowledge Base

I.2 Implement the integrated model

I.3 Choose content annotation methods

I.4 Implement the application

T.1 Testing
Content requirements analysis

- Given we are developing a Semantic Web application is crucial we reuse data already available on the Web
  - EVDB - http://eventuful.com
  - MusicBrainz - http://musicbrainz.org

For more info please visit http://swa.cefriel.it and http://emanueledellavalle.org!
EVDB is a Web 2.0 website that makes available information about events all around the world.

For each event, it knows:
- The start data
- The end data
- The place in terms of address and geographic coordinates

EVDB offers a Web API in the form of a REST service:
- See [http://api.evdb.com](http://api.evdb.com)

For more info please visit [http://swa.cefriel.it](http://swa.cefriel.it) and [http://emanuele.dellavalle.org](http://emanuele.dellavalle.org)!
MusicBrainz

- is a Web 2.0 website that gathered a large amount of information about music
- offers information about
  - artists and bands
  - songs, albums and tracks
  - relations among artists and bands

- The data of MusicBrainz are available as a PostgreSQL dump
  - see http://musicbrainz.org/doc/DatabaseDownload

For more info please visit http://swa.cefriel.it and http://emanueledellavalle.org!
MusicMoz

- is another Web 2.0 website dedicated to music
- offers information about
  - artists and bands including their nationality
  - music styles and their taxonomic relationships
  - the styles each artist or band plays
- reuses MusicBrainz identifier for artists and bands

The data of MusicMoz are available as large XML files
- see [http://musicmoz.org/xml/](http://musicmoz.org/xml/)

For more info please visit [http://swa.cefriel.it](http://swa.cefriel.it) and [http://emanueledellavalle.org](http://emanueledellavalle.org)!
meex needs to merge this data

- Meex in order to be able to manipulate all this data at the same time needs to merge the data of the three data sources.

- The artists and bands information from MusicBrainz should be linked to
  - The music styles they play from MusicMoz
  - The events related to them from EVDB

For more info please visit http://swa.cefriel.it and http://emanueledellavalle.org!
The data of all three data sources are freely usable, we just need to make sure that the logos of the three applications appears on each page of meex.

- EVDB requests also to include a link to the permalink of the event on EVDB website.
- MusicBrainz request also that derived data are made available in Creative Commons.

Read out more here:
- EVDB - http://api.eventful.com/terms
In this step (namely R.3) we should elicit
  - functional requirements of the application
    - as grouping and filtering data
  - non-functional requirements of the application
    - as performance and scalability w.r.t. number of users

However this is just a tutorial, therefore we concentrate on functional requirements, leaving non-functional requirements underspecified
Application requirements analysis (2)

- Meex
  - must enable a user to explore data in the form of
    - a list
    - a chronological graphic
    - a geographic map
  - for each event must show
    - name
    - begin and end date
    - place
  - for each artist must show
    - name
    - nationality
    - music styles he/she plays
    - related artists
  - must allow users to
    - filter and rank results

For more info please visit [http://swa.cefriel.it](http://swa.cefriel.it) and [http://emanueledellavalle.org](http://emanueledellavalle.org)!
Motivations for RDF

- Eric Prud'hommeaux explains*
  - Simple, consistent data model
  - Uses web architecture for web scalability
  - Glamorous use cases

* [http://www.w3.org/2008/Talks/1027-ISWC/HCLS](http://www.w3.org/2008/Talks/1027-ISWC/HCLS)

For more info please visit [http://swa.cefriel.it](http://swa.cefriel.it) and [http://emanueledellavalle.org](http://emanueledellavalle.org)!
What does RDF provide?

- Common (simple) model to for all data.

- Incentive and infrastructure to re-use terms when possible and invent terms when necessary.

- Simple and complex ontological languages (RDFS and OWL).

- Intuitive re-use of now-familiar web topology.

- Scalable — partial (monotonic) reasoning allowed.

- Apps need not be re-written for each extension to a data model.

For more info please visit [http://swa.cefriel.it](http://swa.cefriel.it) and [http://emanueledellavalle.org](http://emanueledellavalle.org)!
How do we write RDF?

- Name resources and relationships with URIs
  - e.g. http://emanueledellavalle.org/sw/foaf.rdf#me represents a person

- Express statements as subject, predicate, object

- Write the triples in
  - RDF/XML: Standard serialization in XML
    <Description about="subject">
    <property>value</property>
    </Description>
  
  - NTriples: Simple (verbose) reference serialization (for specifications only)
    <http://...subject> <http://...predicate> "value" .
  
  - N3 and Turtle: Developer-friendly serializations
    :subject :property "value" .

For more info please visit http://swa.cefriel.it and http://emanueledellavalle.org !
Turtle Introduction

- RDF triples analogous to one 3-place holds(s, p, o) predicate

  edv:me foaf:knows ic:me .
  holds(edv:me, foaf:knows, ic:me)

- Triples made from standard RDF terms:
  - IRIs: <http://emanueledellavalle.org/sw/foaf.rdf#me> or #me or edv:me
  - Literals: "Emanuele Della Valle"
    - Typed literals: "3.14"^^xsd:float
    - Literals with language tags: "日本語"@ja
  - Blank nodes: [] or _:bob

For more info please visit [http://swa.cefriel.it](http://swa.cefriel.it) and [http://emanueledellavalle.org](http://emanueledellavalle.org)!
Convenience Syntax

- URI terms can be abbreviated using namespaces
  ```
  @prefix edv: <http://emanueledellavalle.org/sw/foaf.rdf#> .
  @prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
  @prefix foaf: <http://xmlns.com/foaf/0.1/>
  edv:me rdf:type foaf:Person .
  ```

- `a` = `<http://www.w3.org/1999/02/22-rdf-syntax-ns#type>`
  ```
  edv:me a foaf:Person .
  ```

- In-line blank nodes
  ```
  edv:me foaf:knows [ foaf:name "Irene Celino" ] .
  ```
Abbreviating repeated subjects:

```rdfs
edv:me  rdf:type foaf:Person .
edv:me  foaf:knows ic:me .
```

... is the same as ...

```rdfs
edv:me  rdf:type foaf:Person ; foaf:knows ic:me .
```

Abbreviating repeated subject/predicate pairs:

```rdfs
edv:me  foaf:knows ic:me .
edv:me  foaf:knows dc:me .
```

... is the same as ...

```rdfs
edv:me  foaf:knows ic:me , dc:me .
```
Artist data in RDF

- **Original XML data as in MusicMoz**
  
  ```xml
  <category name="Bands_and_Artists/B/Beatles,_The"
type="band">
    <resource name="musicbrainz"
      link="http://musicbrainz.org/artist/
b10bbbfc-cf9e-42e0-be17-e2c3e1d2600d.html"/>
    <style number="1">British Invasion</style>
    <style number="2">Rock</style>
    <style number="3">Skiffle</style>
  </category>
  ```

- **The same data represented in RDF in meex**
  
  ```sparql
  mb:artist/b10bbbfc-cf9e-42e0-be17-e2c3e1d2600d.html
  a mb:Band ;
  rdfs:label "The Beatles" ;
  mm:hasStyle mm:BritishInvation, mm:Rock, mm:Skiffle .
  ```

For more info please visit [http://swa.cefriel.it](http://swa.cefriel.it) and [http://emanueledellavalle.org](http://emanueledellavalle.org)!
RDF Resources

- **RDF at the W3C** - primer and specifications
- **Semantic Web tools** - community maintained list; includes triple store, programming environments, tool sets, and more
- **302 Semantic Web Videos and Podcasts** - includes a section specifically on RDF videos

For more info please visit [http://swa.cefriel.it](http://swa.cefriel.it) and [http://emanueledellavalle.org](http://emanueledellavalle.org)!
RDFS/OWL in a nutshell: class and instance

Creating a class
- **RDFS:** `Artist rdf:type rdfs:Class .`
- **FOL:** `∃x Artist(x)`

Creating a subclass
- **RDFS:** `Painter rdfs:subClassOf Artist .`
- **RDFS:** `Sculptor rdfs:subClassOf Artist .`
- **FOL:** `∀x [Painter(x) ∨ Sculptor(x) → Artist(x)]`

Creating an instance
- **RDFS:** `Rodin rdf:type Sculptor .`
- **FOL:** `Sculptor(Rodin)`
RDFS/OWL in a nutshell: properties

- Creating a property
  - **RDFS:** creates rdf:type rdf:Property .
  - **FOL:** \( \exists x \exists y \text{Creates}(x, y) \)

- Using a property
  - **RDFS:** Rodin creates TheKiss .
  - **FOL:** Creates(Rodin, TheKiss)

- Creating subproperties
  - **RDFS:** paints rdfs:subPropertyOf creates .
  - **FOL:** \( \forall x \forall y [\text{Paints}(x, y) \rightarrow \text{Creates}(x, y)] \)
  - **RDFS:** sculpts rdfs:subPropertyOf creates .
  - **FOL:** \( \forall x \forall y [\text{Sculpts}(x, y) \rightarrow \text{Creates}(x, y)] \)

For more info please visit [http://swa.cefriel.it](http://swa.cefriel.it) and [http://emanueledellavalle.org](http://emanueledellavalle.org)!
RDFS/OWL in a nutshell: range & domain

- Checking which classes and properties can be used together

**RDFS:**
- creates rdfs:domain Artist.
- creates rdfs:range Piece.
- paints rdfs:domain Painter.
- paints rdfs:range Paint.
- sculpts rdfs:domain Sculptor.
- sculpts rdfs:range Sculpt.

**FOL:**
- \( \forall x \forall y \ [\text{Crea}(x,y) \rightarrow \text{Artista}(x) \land \text{Opera}(y)] \)
- \( \forall x \forall y \ [\text{Dipinge}(x,y) \rightarrow \text{Pittore}(x) \land \text{Pittura}(y)] \)
- \( \forall x \forall y \ [\text{Scolpisce}(x,y) \rightarrow \text{Scultore}(x) \land \text{Scultura}(y)] \)

For more info please visit [http://swa.cefriel.it](http://swa.cefriel.it) and [http://emanueledellavalle.org](http://emanueledellavalle.org)!
The resulting ontology

Artist

 Painter

 paints

Sculptor

Sculptor

creates

Piece

 Paint

sculpts

Sculpt

For more info please visit http://swa.cefriel.it and http://emanueledellavalle.org!
## Some Inference Rules

<table>
<thead>
<tr>
<th>if</th>
<th>then</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x \text{ rdfs:subClassOf } y$ .</td>
<td>$a \text{ rdf:type } y$ .</td>
</tr>
<tr>
<td>$a \text{ rdf:type } x$ .</td>
<td></td>
</tr>
<tr>
<td>$x \text{ rdfs:subClassOf } y$ .</td>
<td></td>
</tr>
<tr>
<td>$a \text{ rdf:type } x$ .</td>
<td></td>
</tr>
<tr>
<td>$a \text{ rdf:type } y$ .</td>
<td></td>
</tr>
<tr>
<td>$x \text{ rdfs:subClassOf } y$ .</td>
<td>$x \text{ rdfs:subClassOf } z$ .</td>
</tr>
<tr>
<td>$y \text{ rdfs:subClassOf } z$ .</td>
<td></td>
</tr>
<tr>
<td>$x \text{ a } y$ .</td>
<td>$x \text{ b } y$ .</td>
</tr>
<tr>
<td>$a \text{ rdfs:subPropertyOf } b$ .</td>
<td>$x \text{ b } y$ .</td>
</tr>
<tr>
<td>$a \text{ rdfs:subPropertyOf } b$ .</td>
<td></td>
</tr>
<tr>
<td>$a \text{ rdfs:subPropertyOf } b$ .</td>
<td></td>
</tr>
<tr>
<td>$b \text{ rdfs:subPropertyOf } c$ .</td>
<td>$a \text{ rdfs:subPropertyOf } c$ .</td>
</tr>
<tr>
<td>$b \text{ rdfs:subPropertyOf } c$ .</td>
<td></td>
</tr>
<tr>
<td>$x \text{ a } y$ .</td>
<td></td>
</tr>
<tr>
<td>$x \text{ rdf:type } z$ .</td>
<td>$x \text{ rdf:type } z$ .</td>
</tr>
<tr>
<td>$a \text{ rdfs:domain } z$ .</td>
<td></td>
</tr>
<tr>
<td>$x \text{ a } u$ .</td>
<td></td>
</tr>
<tr>
<td>$a \text{ rdfs:range } z$ .</td>
<td>$u \text{ rdf:type } z$ .</td>
</tr>
<tr>
<td>$a \text{ rdfs:range } z$ .</td>
<td></td>
</tr>
</tbody>
</table>

For more info please visit [http://swa.crefiel.it](http://swa.crefiel.it) and [http://emanueledellavalle.org](http://emanueledellavalle.org)!
Shared the ontology ...

- Sculptor rdf:subClassOf Artist .
- Painter rdf:subClassOf Artist .
- Painter owl:disjointWith Sculptor .
- Sculpt rdf:subClassOf Piece.
- Painting rdf:subClassOf Piece .
- Painting owl:disjointWith Sculpt.
- creates rdfs:domain Artist .
- creates rdfs:range Piece.
- sculpts rdfs:subPropertyOf creates .
- sculpts rdfs:domain Sculptor .
- sculpts rdfs:range Sculpt .

... when transmitting the following triple ...

- Rodin sculpts TheKiss .

... the recipient can ...
... the recipient can answer the syntactic query
- Sculpts(Rodin, TheKiss)? yes

but it can also answer queries such as
- Sculptor(Rodin)? yes
- Artist(Rodin)? yes
- Painter(Rodin)? no
- Sculpt(TheKiss)? yes
- Piece(TheKiss)? yes
- Painting(TheKiss)? no
- Creates(Rodin, TheKiss)? yes
- Painting(Rodin, TheKiss)? no

**NOTE:** The recipient cannot give such answers without sharing the ontology (i.e., if the triple was an XML fragment valid w.r.t. a shared DTD)
As first design step (namely D.1) we model the **application ontology**

meex must manage information related to
- artists
- events at which the artists participate and
- music styles the artists play

For more info please visit [http://swa.cefriel.it](http://swa.cefriel.it) and [http://emanuele.dellavalle.org](http://emanuele.dellavalle.org)!
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix owl: <http://www.w3.org/2002/07/owl#> .
@prefix meex: <http://swa.cefriel.it/meex#> .

meex:Performer a owl:Class ;
    rdfs:label "Performer" .

meex:fromCountry a owl:DatatypeProperty ;
    rdfs:domain meex:Performer ;
    rdfs:range <http://www.w3.org/2001/XMLSchema#string> .

meex:relatedPerformer a owl:ObjectProperty ;
    rdfs:domain meex:Performer ;
    rdfs:range meex:Performer .

[more to follow]

Meex.n3
Modeling Style in OWL

[follows]

meex:Style a owl:Class .
  rdfs:label "Music Style" .
meex:performsStyle a owl:ObjectProperty ;
  rdfs:domain meex:Performer ;
  rdfs:range meex:Style .

[more to follow]

Meex.n3
For each event we should model begin and end date together with the place, but an XML schema defined by Google exists; thus we decide to reuse it by merging it.

For more info please visit http://swa.cefriel.it and http://emanueledellavalle.org!
Modeling \textit{When} in OWL

[namespace declaration]

\texttt{gd:When a owl:Class;}
\hspace{1em}
\texttt{rdfs:label "Time" .}

\texttt{gd:startTime a owl:DatatypeProperty ;}
\hspace{1em}
\texttt{rdfs:domain gd:When ;}
\hspace{1em}
\texttt{rdfs:range}
\hspace{2em}
\texttt{<http://www.w3.org/2001/XMLSchema#string> .}

\texttt{gd:endTime a owl:DatatypeProperty ;}
\hspace{1em}
\texttt{rdfs:domain gd:When ;}
\hspace{1em}
\texttt{rdfs:range}
\hspace{2em}
\texttt{<http://www.w3.org/2001/XMLSchema#string> .}

[more to follow]
Modeling *Where* in OWL

```n3
gd:Where a owl:Class; rdfs:label "Location" .
gd:postalAddress a owl:DatatypeProperty ;
    rdfs:domain gd:Where ;
    rdfs:range
      <http://www.w3.org/2001/XMLSchema#string>.
gd:hasGeoPt a owl:ObjectProperty ;
    rdfs:domain gd:Where ;
    rdfs:range gd:GeoPt .
gd:GeoPt a owl:Class ; rdfs:label "Geo-referenced Point" .
gd:lat a owl:DatatypeProperty ;
    rdfs:domain gd:GeoPt ;
    rdfs:range <http://www.w3.org/2001/XMLSchema#string>.
gd:lon a owl:DatatypeProperty ;
    rdfs:domain gd:GeoPt ;
    rdfs:range <http://www.w3.org/2001/XMLSchema#string>.
gd:label rdfs:subPropertyOf rdfs:label .
```

For more info please visit [http://swa.cefriel.it](http://swa.cefriel.it) and [http://emanueledellavalle.org](http://emanueledellavalle.org)!
We keep following our approach and we model the content ontology (step D.2)

The content ontology models in OWL the data of the three data sources used by meex

In the mean time we also model the sample contents (step D.3) that we will use to test meex during its implementation (see test-first method from Agile manifesto)
Modeling MusicBrainz schema in OWL

```n3
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix owl: <http://www.w3.org/2002/07/owl#> .
@prefix mb: <http://musicbrainz.org/> .

mb:Artist a owl:Class ;
    rdfs:label "MusicBrainz Artist and Band" .

mb:artist_relation a owl:ObjectProperty ;
    rdfs:domain mb:Artist ;
    rdfs:range mb:Artist .
```

MusicBrainz.n3

For more info please visit [http://swa.cefriel.it](http://swa.cefriel.it) and [http://emanueleedellavalle.org](http://emanueleedellavalle.org)!
Sample data for MusicBrainz in OWL

```owl
mb:artist/b10bbbfccf9e-42e0-be17-e2c3e1d2600d.html
   a mb:Artist ;
   rdfs:label "The Beatles" ;
   mb:related_artist
      mb:artist/ebfc1398-8d96-47e3-82c3-f782abcdb13d.html ,
      mb:artist/618b6900-0618-4fle-b835-bccb17f84294.html .

mb:artist/ebfc1398-8d96-47e3-82c3-f782abcdb13d.html
   a mb:Artist ;
   rdfs:label "The Beach Boys" .

mb:artist/618b6900-0618-4fle-b835-bccb17f84294.html
   a mb:Artist ;
   rdfs:label "Eric Clapton" .
```

SampleInstance-MusicBrainz.n3

- Please note that we choose to build the URI using the ID that MusicBrainz uses to identify the artists. This allows for easier reuse of meex data in other applications.

For more info please visit [http://swa.cefriel.it](http://swa.cefriel.it) and [http://emanueledellavalle.org](http://emanueledellavalle.org)!
MusicMoz schema

```
category

  type

<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>*</td>
<td>resource</td>
</tr>
<tr>
<td></td>
<td>name</td>
</tr>
<tr>
<td></td>
<td>link</td>
</tr>
<tr>
<td></td>
<td>name</td>
</tr>
</tbody>
</table>

| 1              | from string     |
|                | style           |
|                | name            |
```

For more info please visit [http://swa.cefriel.it](http://swa.cefriel.it) and [http://emanueledellavalle.org](http://emanueledellavalle.org)!
Modeling MusicMoz schema in OWL

```owl
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix owl: <http://www.w3.org/2002/07/owl#> .
@prefix mm: <http://musicmoz.org/> .
@prefix mb: <http://musicbrainz.org/> .

mm:from a owl:DatatypeProperty ;
   rdfs:domain mb:Artist ;
   rdfs:range <http://www.w3.org/2001/XMLSchema#string>.

mm:Style a owl:Class ;
   rdfs:label "MusicMoz Music Style" .

mm:hasStyle a owl:ObjectProperty ;
   rdfs:domain mb:Artist ;
   rdfs:range mm:Style .
```

MusicMoz.n3
Sample data for MusicMoz in OWL

```
mb:artist/b10bbbfcc-f9e-42e0-be17-e2c3e1d2600d.html
    mm:from "England" ;
    mm:hasStyle mm:style/British-Invasion ,
        mm:style/Rock ,
        mm:style/Skiffle .

mm:style/British-Invasion a mm:Style ;
rdfs:label "British Invasion" .
```

Please note that also in this case we use the ID derived from MusicBrainz
Modeling EVDB schema in OWL

```owl
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix owl: <http://www.w3.org/2002/07/owl#> .
@prefix evdb: <http://eventful.com/> .
@prefix gd: <http://schemas.google.com/g/2005> .

evdb:Event a owl:Class ;
   rdfs:label "Eventful Event" .

evdb:hasWhen a owl:ObjectProperty ;
   rdfs:domain evdb:Event ;
   rdfs:range gd:When .

evdb:hasWhere a owl:ObjectProperty ;
   rdfs:domain evdb:Event ;
   rdfs:range gd:Where .
```

Please note that we reuse the concepts When and Where we model in the application ontology by merging Google schema (see GoogleSchema.n3).

For more info please visit http://swa.cefriel.it and http://emanueleedellavalle.org!
evdb:events/E0-001-008121669-0@2008022719 a evdb:Event;
  gd:label "Tell Me Why: A Beatles Commentary".
  evdb:hasWhen evdb:events/E0-001-008121669-0@2008022719_When;
  evdb:hasWhere evdb:events/E0-001-008121669-0@2008022719_Where.

evdb:events/E0-001-008121669-0@2008022719_When
  gd:startTime "2008-02-28";
  gd:endTime "2008-02-28".

evdb:events/E0-001-008121669-0@2008022719_Where
  gd:hasGeoPt evdb:events/E0-001-008121669-0@2008022719_GeoPt;
  gd:label "The Wilmington Memorial Library";
  gd:postalAddress "175 Middlesex Avenue, Wilmington, USA".

evdb:events/E0-001-008121669-0@2008022719_GeoPt
  gd:lat "42.556943";
  gd:lon "-71.165576".
“Application Connected by Concepts”

For more info please visit [http://swa.cefriel.it](http://swa.cefriel.it) and [http://emanueledellavalle.org](http://emanueledellavalle.org)!
Eric Prud'hommeaux explains* that SPARQL is the query language of the Semantic Web.

It lets us:

- Pull values from structured and semi-structured data
- Explore data by querying unknown relationships
- Perform complex joins of disparate databases in a single, simple query
- Transform RDF data from one vocabulary to another

* [http://www.w3.org/2008/Talks/1027-ISWC/HCLS](http://www.w3.org/2008/Talks/1027-ISWC/HCLS)
SELECTing variables

- SPARQL variables bind to RDF terms
  - Ex. ?artist, ?album, ?times_platinum

- Like SQL, we pick the variables we want from a query with a SELECT clause
  - Ex. SELECT ?artist ?album ?times_platinum

- A SELECT query results in a table of values:

```
?artist           ?album                ?times_platinum
Michael Jackson   Thriller             27
Led Zeppelin      Led Zeppelin IV     22
Pink Floyd        The Wall             22
```
A triple pattern is an RDF triple that can have variables in any of the subject, predicate, or object positions.

Examples:

- Find countries and their capital cities:

- Given a FOAF URI, find the person's name:
  - edv:me foaf:name ?name .

- What direct relationships exist between two people?
  - edv:me ?relationship ic:me .
Simple query pattern

- We can combine more than one triple pattern to retrieve multiple values and easily traverse an RDF graph:

  - Find countries, their capital cities, and their populations:
    ```
    ?country geo:capital ?capital ;
    geo:population ?population .
    ```

  - Given a FOAF URI, find the person's name and friends' names:
    ```
    edv:me foaf:name ?name ;
    foaf:knows ?friend .
    ```

  - Retrieve all third-line managers in the company:
    ```
    ?emp hr:managedBy ?first_line .
    ?first_line hr:managedBy ?second_line .
    ?second_line hr:managedBy ?third_line .
    ```

For more info please visit [http://swa.cefriel.it](http://swa.cefriel.it) and [http://emanueledellavalle.org](http://emanueledellavalle.org)!
Besides selecting tables of values, SPARQL allows three other types of queries:

- **ASK** - returns a boolean answering, does the query have any results?
- **CONSTRUCT** - uses variable bindings to return new RDF triples
- **DESCRIBE** - returns server-determined RDF about the queried resources

- **SELECT** and **ASK** results can be returned as XML or JSON.
- **CONSTRUCT** and **DESCRIBE** results can be returned via any RDF serialization (e.g. RDF/XML or Turtle).
Protocol Mechanics

The SPARQL Protocol is a simple method for asking and answering SPARQL queries over HTTP. A SPARQL URL is built from three parts:

1. The URL of a SPARQL endpoint
e.g. http://dbpedia.org/sparql

2. (Optional, as part of the query string) The graphs to be queried against
e.g. default-graph-uri=http://dbpedia.org

3. (As part of the query string) The query itself
e.g. Query=SELECT distinct ?x
WHERE {?x a <http://umbel.org/umbel/sc/Artist> }

This is the resulting URL

http://dbpedia.org/sparql?default-graph-uri=http%3A%2F%2Fdbpedia.org&query=SELECT+distinct+%3Fx+WHERE+{+%3Fx+a+%3Chttp%3A%2F%2Fumbel.org%2Fumbel%2Fsc%2FArtist%3E+}
SPARQL Resources

- **SPARQL Frequently Asked Questions**
- **SPARQL implementations** - community maintained list of open-source and commercial SPARQL engines
- **Public SPARQL endpoints** - community maintained list
- **SPARQL extensions** - collection of SPARQL extensions implemented in various SPARQL engines

For more info please visit [http://swa.cefriel.it](http://swa.cefriel.it) and [http://emanueledellavalle.org](http://emanueledellavalle.org)!
D.1 Model the application ontology
    - Reuse
    - Merge
    - Extend

D.2 Model the content ontology
    - Reuse
    - Merge
    - Extend

D.3 Model sample contents

D.4 Design Application

I.1 Implement the initial Knowledge Base

I.2 Implement the integrated model

I.3 Choose content annotation methods

I.4 Implement the application

T.1 Testing

V.1 Validation

R.1 Users’ needs analysis

R.2 Risk analysis

R.3 Software requirements analysis

R.4 Content requirements analysis
We are done with the modeling of ontologies and sample contents

We can now design meex (step D.4 of our approach)

In order to design meex architecture

- We first **design** its **interfaces** in terms of
  - both graphic user interface
  - and connection to the three data sources

- Secondly we **design** how it works **inside** in terms of
  - components and
  - execution semantics

For more info please visit [http://swa.cefriel.it](http://swa.cefriel.it) and [http://emanueledellavalle.org](http://emanueledellavalle.org)!
For more info please visit http://swa.cefriel.it and http://emanueledellavalle.org!
In order to get RDF data out from the three external data source we can use different techniques

- For **MusicBrainz** database we can use tools that enable to query non-RDF databases as virtual RDF graphs using a standard SPARQL endpoint
- For **MusicMoz** XML files we can use a GRDDL processor using the XSLT `MusicMoz->RDF`
- For **EVDB** we can use a GRDDL processor applying the XSLT `EVDB->RDF` to the XML file obtained using the EVDB REST service

For more info please visit [http://swa.cefriel.it](http://swa.cefriel.it) and [http://emanueledellavalle.org](http://emanueledellavalle.org)!
User Interface

- In order to collect users’ input and to present results back to the users, we can use Web 2.0 technologies and develop an AJAX interface

- Such AJAX interface must allow for
  - Inserting the music style, the resulting events will refer to
  - Exploring the events found by meex
  - Filtering the events based on
    - Artists
    - Their nationality
    - The music style they play

For more info please visit http://swa.cefriel.it and http://emanueledellavalle.org!
Designing how meex works inside

For each Artist

- **Music style**
- **Set of artist in RDF**

**Ajax Web Framework**

**GRDDL Processor**

**MusicMoz → RDF**

**HTTP REST Client**

**EVDB**

**HTTP REST service**

**GRDDL Processor**

**EVDB → RDF**

**HTTP Query**

**Events in XML**

**Events in RDF**

**GRDDL Processor**

**Linking Artists to Events**

**RDF Merge**

**Artists and events in RDF**

**MusicBrainz**

**SPARQL Endpoint**

**SPARQL Query**

**Dati RDF**

**Artists and events in RDF**

**MusicMoz → RDF**

**Ajax Web Framework**

**For more info please visit [http://swa.cefriel.it](http://swa.cefriel.it) and [http://emanueledellavalle.org](http://emanueledellavalle.org) !**
1. The user requests a music style

2. meex access the local copy of MusicMoz and using the GRDDL processors obtains a set of artist that plays the given music style

[more to follow]
3. For each artist meex:
   a) uses the SPARQL client to query the MusicBrainz SPARQL endpoint and it obtains the artist name and his/her relationships with other artist
   b) invokes the EVDB REST service, it obtains the events that refer to the artist in XML and uses the GRDDL processor to obtain this data in RDF
   c) links the data about each artist to the data about the events that refers to him/her

[more to follow]
4. When all the pieces of information about artists and events are available in the RDF storage, meex extracts them and serializes them in the format of the Ajax Web framework.

5. The Ajax Web framework allows the user for exploring the events found by meex.

6. When the user decides to start a new exploration, meex starts over from the beginning.
Two important internal components

- **The RDF storage**
  - must be initialized with both the application and the content ontology
  - is filled in with the data meex loads from the three data source given the music style requested by the user

- **The reasoner**
  - allows all query in meex to be express in terms of the application ontology even if data are loaded from the data sources using the content ontology

**NOTE:** the reasoner support the semantic integration of the data loaded from the external data sources. The meex’s programmer can ignore that multiple and heterogeneous data sources were used to load data.

For more info please visit [http://swa.cefriel.it](http://swa.cefriel.it) and [http://emanueledellavalle.org](http://emanueledellavalle.org)!
R.1 Users’ needs analysis

R.2 Risk analysis

R.3 Software requirements analysis

R.4 Content requirements analysis

D.1 Model the application ontology

Reuse
Merge
Extend

D.2 Model the content ontology

Reuse
Merge
Extend

D.3 Model sample contents

D.4 Design Application

I.1 Implement the initial Knowledge Base

I.2 Implement the integrated model

T.1 Testing

I.3 Choose content annotation methods

I.4 Implement the application

V.1 Validation
Implement the initial Knowledge Base (1)

- We start implementing meex by setting up the initial knowledge base (step I.1)

- We need to select tools
  - to read and write RDF in the RDF/XML and RDF/N3 syntax
  - to manipulate programmatically RDF
  - to store RDF
  - to reason on OWL
  - to interpret SPARQL

For more info please visit [http://swa.cefriel.it](http://swa.cefriel.it) and [http://emanueledellavalle.org](http://emanueledellavalle.org)!
We choose Jena because
- offers API
  - to read and write different RDF syntax
  - provides a programmatic environment for RDF, RDFS and OWL, SPARQL a
- guarantees RDF model persistence through several relational database adapters
- includes a rule-based inference engine which implement OWL semantics
- includes ARQ, a query engine that supports SPARQL

In order to use the RDF storage and the OWL reasoner from Jena we need to configure them as shown in the following slides

For more info please visit http://swa.cefriel.it and http://emanueledellavalle.org!
We choose to use Derby (from Apache) as relational database underneath the RDF storage.

1. `Class.forName("org.apache.derby.jdbc.EmbeddedDriver");`
2. `DBConnection con = new DBConnection("jdbc:derby:C:/Meex/RDFStorage;create=true","sa","", "Derby");`
3. `Model model = ModelFactory.createModelRDBMaker(con).createDefaultModel();`

- With row 1 we tell Jena where to find the JDBC driver
- With row 2 we define the JDBC connection
- With row 3 we instantiate the object `model` of Jena we will use to access and manipulate the RDF model in the storage

For more info please visit [http://swa.cefriel.it](http://swa.cefriel.it) and [http://emanueledellavalle.org](http://emanueledellavalle.org)!
Configuring the OWL reasoner

1. Reasoner reasoner = ReasonerRegistry.getOWL_microReasoner();
2. model = ModelFactory.createInfModel(reasoner, model);

- Jena offers numerous options to configure the internal rule-based inference engine with different expressivity-performance tradeoffs
- We need simple reasoning features (i.e., subClassOf and subPropertyOf transitive closure), the OWL Micro configuration is, therefore, the most appropriate one
- With row 1 we instantiate a OWL micro reasoner
- With row 2 we instantiate a model with inference support using the model previously created and the OWL micro reasoner

For more info please visit [http://swa.cefriel.it](http://swa.cefriel.it) and [http://emanueledellavalle.org](http://emanueledellavalle.org)!
R.1 Users' needs analysis

R.2 Risk analysis

R.3 Software requirements analysis

R.4 Content requirements analysis

D.1 Model the application ontology
  - Reuse
  - Merge
  - Extend

D.2 Model the content ontology
  - Reuse
  - Merge
  - Extend

D.3 Model sample contents

D.4 Design Application

I.1 Implement the initial Knowledge Base

I.2 Implement the integrated model

I.3 Choose content annotation methods

I.4 Implement the application

T.1 Testing
We move on with the implementation of meex realizing the integrated model (step I.2)

In the integrated model we merge application and content ontology

- Our intent is to integrate semantically the heterogeneous data coming from the external data sources

In order to realize the integrated model we need to define a **bridge ontology** using the properties

- `rdfs:subclassOf`
- `rdfs:subpropertyOf`

To connect classes and properties in the application ontology to those in the content ontology

For more info please visit [http://swa.cefriel.it](http://swa.cefriel.it) and [http://emanueledellavalle.org](http://emanueledellavalle.org)!
Implement the integrated model (2)

1. \text{mb:Artist rdfs:subClassOf meex:Performer}.
2. \text{mb:related_artist rdfs:subPropertyOf meex:relatedPerformer}.
3. \text{mm:Style rdfs:subClassOf meex:Style}.
4. \text{mm:hasStyle rdfs:subPropertyOf meex:performsStyle}.
5. \text{mm:from rdfs:subPropertyOf meex:fromCountry}.
6. \text{evdb:Event rdfs:subClassOf meex:Event}.
7. \text{evdb:hasWhen rdfs:subPropertyOf meex:hasWhen}.
8. \text{evdb:hasWhere rdfs:subPropertyOf meex:hasWhere}.

- In rows 1 and 2 we connect the ontology of MusicBrainz to the application ontology, i.e.
  - the classes \text{mb:Artist} and \text{meex:Performer}
  - the properties \text{mb:related_artist} and \text{meex:relatedPerformer}.
- Likewise, in rows 3, 4 and 5, we connect the ontology of MusicMoz to the application ontology and
- in rows 6, 7 and 8 we connect the ontology of EVDB to the application ontology.

For more info please visit \url{http://swa.cefriel.it} and \url{http://emanueledellavalle.org}!
Thanks to this bridge ontology, when data loaded from the external data sources are inserted in the RDF storage (using the data source specific ontologies), the OWL micro reasoner infers the triples that represent the same data in the application ontology.

meex can, therefore, query the RDF storage homogeneously in the terms of application ontology without caring of the heterogeneous formats of the three data sources.

To give an idea of the differences, in the next slide we compare the data expressed:

- in MusicBrainz ontology and
- in the application ontology.
Implement the integrated model (4)

```
mb:artist/b10bbbfc-cf9e-42e0-be17-e2c3e1d2600d.html
    a mb:Artist ;
    rdfs:label "The Beatles" ;
    mb:related_artist
        mb:artist/ebfc1398-8d96-47e3-82c3-f782abcdb13d.html,
        mb:artist/618b6900-0618-4f1e-b835-bcccb17f84294.html.
```

```
mb:artist/b10bbbfc-cf9e-42e0-be17-e2c3e1d2600d.html
    a meex:Performer ;
    rdfs:label "The Beatles" ;
    meex:relatedPerformer
        mb:artist/ebfc1398-8d96-47e3-82c3-f782abcdb13d.html,
        mb:artist/618b6900-0618-4f1e-b835-bcccb17f84294.html.
```

For more info please visit http://swa.cefriel.it and http://emanueledellavalle.org!
Now that we have configure both the RDF storage and the reasoner we can load all ontologies

```java
model.read("Meex.n3", ",", "N3");
model.read("Google.n3", ",", "N3");
model.read("MeexBindings.n3", ",", "N3");
model.read("MusicBrainz.n3", ",", "N3");
model.read("MusicMoz.n3", ",", "N3");
model.read("EVDB.n3", ",", "N3");
```

Note that the `read` method of `model` requires:
- The name of the file to load,
- The base URI (in our case all URI are absolute) and
- The RDF syntax in which data are serialized

For more info please visit http://swa.cefriel.it and http://emanuele.dellavalle.org!
**R.1 Users’ needs analysis**

**R.2 Risk analysis**

**R.3 Software requirements analysis**

**R.4 Content requirements analysis**

**V.1 Validation**

**D.1 Model the application ontology**
- Reuse
- Merge
- Extend

**D.2 Model the content ontology**
- Reuse
- Merge
- Extend

**D.3 Model sample contents**

**D.4 Design Application**

**I.1 Implement the initial Knowledge Base**

**I.2 Implement the integrated model**

**I.3 Choose content annotation methods**

**I.4 Implement the application**

**T.1 Testing**
Testing the integrated model

- A simple test, which we can perform to verify the semantic soundness of all the ontologies we modelled, consists in loading in the model the example we produced (in step D.3) and extracting the entire content of the RDF storage in a single file using the `write` method:

  ```java
  model.write("Dump.n3","N3");
  ```

- If we open the file `Dump.n3` we can verify the presence of all the inferred triple we presented in slide 89.
R.1 Users’ needs analysis

R.2 Risk analysis

R.3 Software requirements analysis

R.4 Content requirements analysis

D.1 Model the application ontology

Reuse

Merge

Extend

D.2 Model the content ontology

Reuse

Merge

Extend

D.3 Model sample contents

D.4 Design Application

I.1 Implement the initial Knowledge Base

I.2 Implement the integrated model

I.3 Choose content annotation methods

I.4 Implement the application

T.1 Testing
Choose content annotation methods

- Following the proposed approach, next step (i.e. I.3) suggests to choose content annotation methods.
- The contents we choose for meex are already annotated at data source level, we (only) need to lift the data from XML or relational database as instances of the content ontology.
- In the following slide we show how to implement and configure all the component necessary to allow meex to load data from the external data sources.

For more info please visit http://swa.cefriel.it and http://emanueledellavalle.org!
meex interfaces (1)

1) Music style

2) RDF

3) HTML and RDF

User

Browser Web

SPARQL Server

Adapter Database \(\rightarrow\) RDF

MusicBrainz database

GRDDL processor

EVDB \(\rightarrow\) RDF

MusicMoz \(\rightarrow\) RDF

EVDB REST service

MusicMoz File XML

For more info please visit http://swa.cefriel.it and http://emanueledellavalle.org!
Importing annotations from MusicBrainz

- The annotations of MusicBrainz are stored as dump of PostgreSQL database
- So, first of all we install the relational database PostgreSQL
  - necessary documentation is available on PostgreSQL and MusicBrainz official websites
- When the database is available we need to install and configure
  1. a translator from relational database to RDF
  2. a SPARQL endpoint
- We choose D2RQ as translator and Joseki as SPARQL server

For more info please visit http://swa.cefriel.it and http://emanueledellavalle.org!
Configuring D2RQ for MusicBrainz (1)

@prefix map: <http://swa.cefriel.it/meex/D2RQ-MusicBrainz.n3#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix owl: <http://www.w3.org/2002/07/owl#> .
@prefix d2rq: <http://www.wiwiss.fu-berlin.de/suhl/bizer/D2RQ/0.1#>. 
@prefix mb: <http://musicbrainz.org/> .

map:database a d2rq:Database;
  d2rq:jdbcDriver "org.postgresql.Driver";
  d2rq:jdbcDSN "jdbc:postgresql://localhost:5432/MusicBrainzDB";
  d2rq:username "postgres";
  d2rq:password "sw-book".

[more to follow]
Configuring D2RQ for MusicBrainz (1)

(map:artist a d2rq:ClassMap;
  d2rq:dataStorage map:database;
  d2rq:class mb:Artist;
  d2rq:uriPattern "http://musicbrainz.org/artist/@@artist.gid@@.html";

map:artist_name a d2rq:PropertyBridge;
  d2rq:belongsToClassMap map:artist;
  d2rq:property rdfs:label;
  d2rq:column "artist.name".

map:artist_relation a d2rq:PropertyBridge;
  d2rq:belongsToClassMap map:artist;
  d2rq:property mb:artist_relation;
  d2rq:join "artist.id = artist_relation.artist";
  d2rq:join "artist_relation.ref = artist2.id";
  d2rq:uriPattern "http://musicbrainz.org/artist/@@artist2.gid@@.html".

NOTE
due to a limitation of D2RQ we need to create a view of the Artist table
create view Artist2
select * from Artist

D2RQ-MusicBrainzDB.n3
Configuring Joseky for MusicBrainz

1. [] rdf:type joseki:Service ;
   rdfs:label "SPARQL for MusicBrainzDB" ;
   joseki:serviceRef "MusicBrainz" ;
   joseki:dataset _:MusicBrainzDS ;
   joseki:processor joseki:ProcessorSPARQL_FixedDS .

2. _:MusicBrainzDS rdf:type ja:RDFDataset ;
   ja:defaultGraph _:MusicBrainzModel ;
   rdfs:label "MusicBrainz Dataset" .

3. _:MusicBrainzModel rdf:type d2rq:D2RQModel ;
   rdfs:label "MusicBrainz D2RQ Model" ;
   d2rq:mappingFile <file:D2RQ-MusicBrainzDB.n3> ;

joseki-config.ttl

- With row 1 we expose a SPARQL endpoint giving the name of the service and the URL at which it will become accessible http://localhost:2020/MusicBrainz

- With row 2 and 3 we configure the SPARQL endpoint to expose MusicBrainz via D2RQ using the configuration file D2RQ-MusicBrainzDB.n3 (see previous slide)
Testing the SPARQL endpoint

1. String sparqlQueryString = "PREFIX mb: <http://musicbrainz.org/>\n    + "DESCRIBE <" + artist + ">");

2. Query query = QueryFactory.create(sparqlQueryString);

3. QueryExecution qexec = QueryExecutionFactory.sparqlService
    ("http://localhost:2020/MusicBrainz", query);

4. Model resultModel = qexec.execDescribe()

- We choose ARQ to test the MusicBrainz SPARQL endpoint submitting a DESCRIBE SPARQL query to obtain the description of an artist

- With row 1 we define the SPARQL query in which the variable artist contains the URI of the artist we want to be described

- With row 2 and 3 we instantiate a query model and we configure the QueryExecution to send the query to the endpoint at the URL http://localhost:2020/MusicBrainz

- With row 4 we execute the query and we obtain a Jena model as a result

For more info please visit http://swa.cefriel.it and http://emanueledellavalle.org!
meex interfaces (2)

For more info please visit http://swa.cefriel.it and http://emanueledellavalle.org!
The MasicBrainz SPARQL endpoint is ready, let’s importing annotations from MusicMoz and EVDB. They both exchange data in XML.

In the design steps we chose to use a GRDDL processor to convert from XML in RDF (in the RDF/XML syntax)

The GRDDL recommendation requires the XML documents to directly refer to the XSLT that performs the translation.

- Neither MusicMoz nor EVDB XML files originally include the reference request by GRDDL
- We can programmatically add it
  - In the following slide we show an excerpt of the modified XML files for MusicMoz

We can proceed likewise for EVDB

For more info please visit http://swa.cefriel.it and http://emanueledellavalle.org!
<musicmoz
    xmlns:grddl='http://www.w3.org/2003/g/data-view#'
    grddl:transformation="file:///[..]/musicmoz-to-rdf.xsl">

    <category name="Bands_and_Artists/B/Beatles,_The"
        type="band">

        <resource name="musicbrainz"
            link="http://musicbrainz.org/artist/
            b10bbbfccf9e-42e0-be17-e2c3e1d2600d.html"/>

        <from>England</from>

        <style number="1">British Invasion</style>
        <style number="2">Rock</style>
        <style number="3">Skiffle</style>
    </category>

</musicmoz>

Excerpts from the files musicmoz.bandsandartists.xml and musicmoz.lists.styles.xml

For more info please visit http://swa.cefriel.it and http://emanueleedellavalle.org!
<xsl:template match="musicmoz/category[(@type='band' or @type='artist') and resource/@name='musicbrainz']">  
    <xsl:variable name="artist_uri" select="resource[@name='musicbrainz']/@link"/>
    <xsl:for-each select="style">
        <xsl:variable name="style_reformatted" select="concat('http://musicmoz.org/style/',text())"/>
        <rdf:Description rdf:about="{${artist_uri}">
            <mm:hasStyle rdf:resource="{${style_reformatted}}"/>
        </rdf:Description>
    </xsl:for-each>
    <rdf:Description rdf:about="{${artist_uri}}">
        <mm:from><xsl:value-of select="from"/></mm:from>
    </rdf:Description>
</xsl:template>

<xsl:template match="musicmoz/style">
    <xsl:variable name="style_reformatted" select="concat('http://musicmoz.org/style/', name)"/>
    <mm:Style rdf:about="{${style_reformatted}}">
        <rdfs:label><xsl:value-of select="name"/></rdfs:label>
    </mm:Style>
</xsl:template>

Excerpts from the file musicmoz-to-rdf.xsl

For more info please visit http://swa.cefriel.it and http://emanueledellavalle.org!
As GRDDL processor we choose **GRDDL Reader**, the GRDDL processor for Jena.

1. Model mmModel = ModelFactory.createDefaultModel();
2. RDFReader reader = mmModel.getReader("GRDDL");
3. reader.read(mmModel, "file:///.../musicmoz.bandsandartists.xml");
4. reader.read(mmModel, "file:///.../musicmoz.lists.styles.xml");
5. model.add(mmModel);

With row 1 we instantiate a Jena model that will momentarily contain the RDF data produced by the GRDDL processor.

With row 2 we instantiate a **RDFReader** that uses a GRDDL processor to load RDF data.

With row 3 and 4 we load in the RDF model instantiate in row 1 the data contained in the XML files of MusicMoz using the RDF reader configured for GRDDL.

With row 5 we merge the loaded RDF data with those already present in the RDF storage.

For more info please visit [http://swa.cefriel.it](http://swa.cefriel.it) and [http://emanueledellavalle.org](http://emanueledellavalle.org)!
So far so good! (1)

1) Music style

2) RDF

3) HTML and RDF

User

Browser

Web

SPARQL

Server

Adapter

Database → RDF

MusicBrainz database

1) Music style

GRDDL processor

XML

EVDB REST service

MusicMoz File XML

EVDB → RDF

MusicMoz → RDF

EVDB

meex

For more info please visit http://swa.cefriel.it and http://emanueledellavalle.org!
So far so good! (2)

For each Artist:
- **MusicBrainz SPARQL Endpoint**
  - Artist
  - **MusicMoz** → RDF
  - **GRDDL Processor**
    - **EVDB** HTTP REST service
      - Events in XML
      - **EVDB** → RDF

For Artist data in RDF:
- **HTTP REST Client**
  - **EVDB** HTTP REST service
  - **GRDDL Processor**
    - Events in RDF
    - **Linking Artists to events**
      - **RDF Merge**
        - Artists and events in RDF

For Set of artist in RDF:
- **GRDDL Processor**
- **Ajax Web Framework**
  - **Music style**

For Artists and events in RDF:
- **Estrazione e trasformazione**
  - **Dati RDF**
  - **Ajax Web Framework**

For more info please visit [http://swa.cefriel.it](http://swa.cefriel.it) and [http://emanueledellavalle.org](http://emanueledellavalle.org)!
R.1 Users’ needs analysis

R.2 Risk analysis

R.3 Software requirements analysis

R.4 Content requirements analysis

D.1 Model the application ontology
  - Reuse
  - Merge
  - Extend

D.2 Model the content ontology
  - Reuse
  - Merge
  - Extend

D.3 Model sample contents

D.4 Design Application

I.1 Implement the initial Knowledge Base

I.2 Implement the integrated model

I.3 Choose content annotation methods

I.4 Implement the application

T.1 Testing
What’s left?

- All the business logic that coordinates the interaction among the internal component is still to be implemented.

- NOTE: Implementing the business logic requires:
  - both writing many lines of pure Java code
  - and work with several Semantic Web technologies

we will focus our attention to the Semantic Web technologies.

- The complete Java code is available on meex the website for downloading.
  - See [http://swa.cefriel.it/meex](http://swa.cefriel.it/meex)

For more info please visit [http://swa.cefriel.it](http://swa.cefriel.it) and [http://emanueledellavalle.org](http://emanueledellavalle.org)!
What’s left?

1. **Ajax Web Framework**
2. **GRDDL Processor**
   - **MusicMoz** → RDF
3. **For each Artist**
   - **SPARQL Client**
   - **HTTP REST Client**
   - **EVDB**
     - **HTTP REST service**
4. **MusicBrainz SPARQL Endpoint**
   - **MusicMoz** → RDF
5. **GRDDL Processor**
   - **EVDB** → RDF
6. **Estrazione e trasformazione**
   - **Dati RDF**
7. **Linking Artists to events**
   - **Artists and events in RDF**
8. **Ajax Web Framework**

For more info please visit [http://swa.cefriel.it](http://swa.cefriel.it) and [http://emanueledellavalle.org](http://emanueledellavalle.org)!
MEMO: Execution Semantics (1)

1. The user requests a music style
2. meex access the local copy of MusicMoz and using the GRDDL processors obtains a set of artist that plays the given music style

[more to follow]
Step 2: from the music style to the artists

The step 2. of meex execution semantics requires to query MusicMoz for the artist that plays the music style requested by the users.

The following Java code shows how to encode the SPARQL query in terms of the application ontology:

```java
String sparqlQueryString = 
    "PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>\n"
    + "PREFIX meex: <http://swa.cefriel.it/meex#>\n"
    + "SELECT DISTINCT ?performer \n"
    + "WHERE { ?performer meex:performsStyle ?style.\n"
    + " ?style rdfs:label \"" + style + "\".}";
```

For more info please visit http://swa.cefriel.it and http://emanueledellavalle.org!
MEMO: Execution Semantics (2)

[follows]

3. For each artist meex:
   a) uses the SPARQL client to query the MusicBrainz SPARQL endpoint and it obtains the artist name and his/her relationships with other artist
   b) invokes the EVDB REST service, it obtains the events that refer to the artist in XML and uses the GRDDL processor to obtain this data in RDF
   c) links the data about each artist to the data about the events that refers to him/her

[more to follow]
Step 3.a: querying MusicBrainz

The step 3.a of meex execution semantics requires to query MusicBrainz for the data that describe an artist including the related artists.

```java
String sparqlQueryString = 
    "PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>\n" + 
    "PREFIX mb: <http://musicbrainz.org/>\n" + 
    "DESCRIBE <" + artist + ">";

SPARQLClient sparqlClient = new SPARQLClient(null);
try {
    return sparqlClient.executeQuery(sparqlQueryString, Config.MusicBrainzSPARQLEndpoint);
} finally {
    sparqlClient.closeQuery();
}
```

Excerpts from the file MusicBrainz.java

For more info please visit [http://swa.cefriel.it](http://swa.cefriel.it) and [http://emanueledellavalle.org](http://emanueledellavalle.org)!
The step 3.b of meex execution semantics requires to invoke the EVDB REST service, obtain the list of events in XML and use the GRDDL processor to obtain the RDF

```java
invokeHttpEndpoint(performerLabel, eventsFilename);
prepareForGRDDL(eventsFilename);
Model m = GRDDLProcessor.ApplyGRDDLTransformation(eventsFilename);

private static void invokeHttpEndpoint(String keywords,
   String outputFilename) throws IOException {
   URL url = new URL(
      "http://api.evdb.com/rest/events/atom?sort_order=relevance&"
   + "keywords=" + URLEncoder.encode(keywords, "UTF-8")
   + "&category=music&app_key="+Config.EVDBKey);
   URLConnection conn = url.openConnection();
   conn.setDoOutput(true);
   BufferedReader in = new BufferedReader(new InputStreamReader(
       conn.getInputStream()));
   [...]  
   while ((inLine = in.readLine()) != null)
      writer.write(inLine + "\n");
}
```

Excerpts from the file EVDB.java
Step 3.c: linking artists to events

- The step 3.c of meex execution semantics requires to link the artist information retrieved from MusicMoz and MusicBrainz to the event information retrieved from EVDB.

- We can use the following SPARQL `CONSTRUCT` query to create the links:

```sql
String sparqlQueryString = 
  "PREFIX meex: <http://swa.cefriel.it/meex#>\n"
+ "CONSTRUCT {" + performer + "} meex:performsEvent ?event.}
+ "WHERE {?event a meex:Event.}";
```

For more info please visit [http://swa.cefriel.it](http://swa.cefriel.it) and [http://emanueledellavalle.org](http://emanueledellavalle.org)!
4. When all the pieces of information about artists and events are available in the RDF storage, meex extracts them and serializes them in the format of the Ajax Web framework.

5. The ajax Web framework allows the user for exploring the events found by meex.

6. When the user decides to start a new exploration, meex starts over from the beginning.
Step 4: preparing the data for the GUI

- We choose **Exhibit** as Ajax Web framework because
  - allows facet browsing
  - allows grouping and filtering events by
    - artist name
    - artist nationality
    - the style the artist plays
    - the related artists
  - includes different views
    - an ordered list
    - a chronological graph
    - a geographic map

For more info please visit [http://swa.cefriel.it](http://swa.cefriel.it) and [http://emanueledellavalle.org](http://emanueledellavalle.org)!
Step 4: configuring Exhibit

- We can configure Exhibit by the means of two files:
  - an HTML page that controls the look and feel and
  - a JSON file that contains the data to be explored by the user

- In this tutorial we focus on the preparation of the JSON file. We refer to Exhibit documentation and the website of our Semantic Web book for the preparation of the HTML page of Exhibit for meex

- A JSON file is a simple text file that contains data organized in set of recors. In the following slide we show the information of The Beatles expressed in JSON.

For more info please visit http://swaCEFRIEL.it and http://emanueledellavalle.org!
Step 4: a sample JSON file

1. type: "Event",
2. label: "1964 The Tribute Tribute to Beatles",
4. when_startTime: "2008-01-25",
5. when_endTime: "2008-01-26",
6. where_label: "Paramount Theater",
7. where_address: "17 South Street, New York 10940, United States",
8. where_latlng: "41.4544,-74.471",
9. performer_label: "The Beatles",
10. fromCountry: "England",
11. styles: ["Skiffle", "British Invasion", "Rock"],
12. relatedPerformers: ["The Beach Boys", "Eric Clapton"]

For more info please visit [http://swa.cefriel.it](http://swa.cefriel.it) and [http://emanueledellavalle.org](http://emanueledellavalle.org)!
Step 4: serializing RDF in JSON

- In order to serialize RDF in JSON
  - we extract the information we loaded in the RDF storage using the SPARQL query shown in the following slide
  - we serialize the result in JSON

- NOTE: as we’ve already said several times, the query can be expressed in terms of the application ontology even if the data were loaded in other heterogeneous formats.
Step 4: extracting the data

PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX meex: <http://swa.cefriel.it/meex#>
PREFIX gd:    <http://schemas.google.com/g/2005>
WHERE {
  ?event rdfs:label ?event_label;
    meex:hasWhen ?when;
    meex:hasWhere ?where.
?when gd:startTime ?when_startTime;
    gd:endTime ?when_endTime.
?where gd:label ?where_label;
    gd:postalAddress ?where_address;
    gd:hasGeoPt ?geoPt.
?geoPt gd:lat ?where_lat;
    gd:lon ?where_lon.
?performer meex:performsEvent ?event;
    rdfs:label ?performer_label;
    meex:fromCountry ?fromCountry.}
20th Anniversary Celebration with Pete Seeger and Friends at Memorial United Methodist Church
At Memorial United Methodist Church from 2009-01-10 to 2009-01-10
Pete Seeger, from United States, performs Folk

A Salute to Paul Robeson at Wilde Lake High School - James Rouse Theater
At Wilde Lake High School - James Rouse Theater from 2009-05-03 to 2009-05-03
Paul Robeson, from United States, performs Opera, Folk, and Gospel

A Seasonal Celebration with John McDermott and special guest Eric Bogle at Centrepoin
At Centrepoin Theatre from 2008-12-22 to 2009-12-22
Eric Bogle, from Australia, performs Folk

Alys Stephens Center Season Opening Celebration featuring BeauSoleil avec Michael Doucet at Saint Stephens, Alabama, United States
At Saint Stephens, Alabama, United States from 2008-09-27 to 2008-09-27
BeauSoleil, from United States, performs Cajun and Folk

An Evening with Hal Ketchum - Full Band at Workplay
At Workplay from 2000-03-22 to 2000-03-22
Hal Ketchum, from United States, performs Country and Folk

Performers
6 BeauSoleil
2 Bob Dylan
2 Boys of the Lough
8 Cara Dillon
20 Donovan
10 Eddi Reader

Performer's Nationality
3
7 Australia
28 Canada
8 Ireland
32 Scotland
23 United Kingdom

Music Styles
191 Folk
24 Celtic
20 British Invasion
20 Folk-Pop
15 Rock
12 Country

Events
3
3 The Aladdin Theater
3 The Birchmore
3 The Quarry Amphitheatre
2 Berklee Performance Center
2 Junction
2 Kimmel Center

For more info please visit http://swa.cefriel.it and http://emanueledellavalle.org!
Tools employed (1)

- **Jena**
  - Application Framework
  - [http://jena.sourceforge.net](http://jena.sourceforge.net)

- **Derby**
  - Relational database for the RDF storage
  - [http://db.apache.org/derby](http://db.apache.org/derby)

- **PostgreSQL**
  - Relational database for MusicBrainz
  - [http://www.postgresql.org](http://www.postgresql.org)

- **D2RQ**
  - Translator from relational database to RDF
  - [http://sites.wiwiss.fu-berlin.de/suhl/bizer/d2rq](http://sites.wiwiss.fu-berlin.de/suhl/bizer/d2rq)

For more info please visit [http://swa.cefriel.it](http://swa.cefriel.it) and [http://emanueledellavalle.org](http://emanueledellavalle.org)!
Tools employed (2)

- Joseki
  - SPARQL Endpoint Server
    - http://www.joseki.org
- ARQ
  - SPARQL query engine for Jena
    - http://jena.sourceforge.net/ARQ
- GRDDL Reader
  - GRDDL processor
    - http://jena.sourceforge.net/grddl
- Exhibit
  - Ajax Web Framework
    - http://static.simile.mit.edu/exhibit

For more info please visit http://swa.cefriel.it and http://emanueledellavalle.org!
Thank you for paying attention

Any Question?

For more info please visit [http://swa.cefriel.it](http://swa.cefriel.it) and [http://emanueledellavalle.org](http://emanueledellavalle.org)!
Realizing a Semantic Web Application

Emanuele Della Valle
Dario Cerizza
Irene Celino

Center of Excellence For Research, Innovation, Education and industrial Lab partnership - Politecnico di Milano

http://www.cefriel.it
http://swa.cefriel.it
emanuele.dellavalle@cefriel.it
http://emanueledellavalle.org
**CREDITS**

- Dario Cerizza [dario.cerizza@cefriel.it]
  - who help in conceiving, designed and developed meex
- Irene Celino [irene.celino@cefriel.it]
  - who help in conceiving and support the design and development of meex
- All the people involved in CEFRIEL’s Semantic Web Activities

**Links**

- Visit [http://swa.cefriel.it](http://swa.cefriel.it)
- Try [http://swa.cefriel.it/Squiggle](http://swa.cefriel.it/Squiggle)
- Try [http://swa.cefriel.it/SOIP-F](http://swa.cefriel.it/SOIP-F)

**Acknowledgement**

- The research behind this tutorial is partially funded by the Italian research project NeP4B

For more info please visit [http://swa.cefriel.it](http://swa.cefriel.it) and [http://emanueleedellavalle.org](http://emanueleedellavalle.org)!
Advertisement: if you speak Italian ...

For more info please visit http://swa.cefriel.it and http://emanueledellavalle.org!