Basic Concepts About Metadata Stores in D2K

Xavier Llorà

Automated Learning Group
National Center for Supercomputing Applications
University of Illinois at Urbana-Champaign

xllora@ncsa.uiuc.edu

Motivation

- Provide easy access to metadata stores in D2K
- Build a generic interface to
  - Manage metadata
  - Archive and retrieve metadata in D2K
  - Reason using the stored data
- How to achieve such goal?
  1. Create an abstraction layer that provides a simple view of a metadata store
  2. Build the management, archival, and the reasoning via the abstracted metadata store layer
  3. Implement D2K wrapping modules of the desired functionalities

Wait, wait, meta-what?

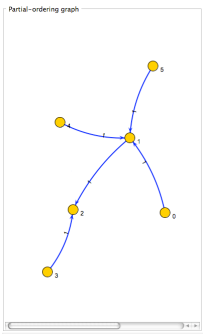
- Metadata usually refers to data that describe some data
- Examples:
  - The description of a table in a RDBMS is metadata
  - The associated information of a blog post (author, timestamp…)
  - The properties on a file system
  - …
- When properly expressed, metadata help to:
  1. Introduce structure in the data
  2. Provide a common framework to store and manipulate heterogeneous data
  3. Metadata allows modeling and reasoning on data

Do NOT Panic Yet! The Enlightening Example

- The data available for an active interactive GA:
  1. A population of solutions. You may regard it as a table where columns are decision variables and the rows are the different solutions explored.
  2. A partial order of the solutions. Given two solutions we collect from the user preference evaluations among pairs of solutions.
- Ugh?

Estimated ranking of promissing solutions

ID	ESF	RESF	X1	X2	X3	X4	X5	X6	X7	X8
1.0	3.0	2.99594	1.0	1.0	0.0	0.0	0.0	1.0	0.0	0.0
2.0	3.0	3.0007212	1.0	0.0	0.0	1.0	0.0	0.0	1.0	1.0
4.0	3.0	3.0033357	0.0	1.0	1.0	0.0	0.0	0.0	1.0	0.0
0.0	-3.0	-2.9992769	0.0	0.0	0.0	0.0	1.0	0.0	1.0	1.0
3.0	-3.0	-2.9966621	0.0	0.0	1.0	0.0	0.0	1.0	0.0	1.0
5.0	-3.0	-3.0040581	0.0	1.0	0.0	1.0	0.0	0.0	0.0	1.0



So What?

- The triple <s,p,o> is the basic basic unit for metadata expression
  - *Subject* (s): The element to describe
  - *Predicate* (p): The predicate that describe the metadata
  - *Object* (o): The metadata attached predicated about the element
- All the metadata is expressed using triples
- The *object* of a triple is the only one that may contain literals
- Some example:
  - <alg:john> <alg:has> <alg:car>
  - <alg:john> <alg:is> '30'
  - …
- Elements have the aspect of <name_space:element>
- Only the object can be a literal
- alg = http://alg.ncsa.uiuc.edu/
- Ontologies (standardization, RDF, …)

Question: How Do I Model My GA Data?

- Example data

Variable 0	Variable 1	Variable 2
1	0	1
0	1	0
0	0	0
1	1	1
1	1	0
0	1	1

- The relations
  - 101 > 010, 010 > 000, 000 > 101, 000 > 111, 111 > 110, 110 > 011, 011> 111

The triples

- A bunch of triples

  <aiga:101> <aiga:greater-than> <aiga:010>

  <aiga:010> <aiga:greater-than> <aiga:000>

  <aiga:000> <aiga:greater-than> <aiga:101>

  <aiga:000> <aiga:greater-than> <aiga:111>

  <aiga:111> <aiga:greater-than> <aiga:110>

  <aiga:110> <aiga:greater-than> <aiga:011>

  <aiga:011> <aiga:greater-than> <aiga:111>

- Chaining triples I can deduce that <aiga:101> is also greater than <aiga:000> because <aiga:101> <aiga:greater-than> <aiga:010> and <aiga:010> <aiga:greater-than> <aiga:000>.
- But is this enough? NO! A lot of information has been left out encoding the triples this way

Triples, Triples, and a Few More Triples

- The metadata available is the one that constrains what kind of questions I can ask
- For instance, the previous triples only allow to reason about the partial order
- No partial string matching or manipulation of entities is promoted. Hence, I can not extract the values of the variables for a given solution
- Let's review again what we know about the aiGA:
  - We have several solutions
  - All the solutions have variables with given values
  - There is a partial order among solutions

The More, The Merrier (I/III)

- <aiga:sol-1> <aiga:gene> 'x0=1'
- <aiga:sol-1> <aiga:gene> 'x1=0'
- <aiga:sol-1> <aiga:gene> 'x2=1'
- <aiga:sol-2> <aiga:gene> 'x0=0'
- <aiga:sol-2> <aiga:gene> 'x1=1'
- <aiga:sol-2> <aiga:gene> 'x2=0'
- <aiga:sol-3> <aiga:gene> 'x0=0'
- <aiga:sol-3> <aiga:gene> 'x1=0'
- <aiga:sol-3> <aiga:gene> 'x2=0'
- <aiga:sol-4> <aiga:gene> 'x0=1'
- <aiga:sol-4> <aiga:gene> 'x1=1'
- <aiga:sol-4> <aiga:gene> 'x2=1'
- <aiga:sol-5> <aiga:gene> 'x0=1'
- <aiga:sol-5> <aiga:gene> 'x1=1'
- <aiga:sol-5> <aiga:gene> 'x2=0'
- <aiga:sol-6> <aiga:gene> 'x0=0'
- <aiga:sol-6> <aiga:gene> 'x1=1'
- <aiga:sol-6> <aiga:gene> 'x2=1'

The More, The Merrier (II/III)

- <aiga:sol-1> <aiga:is-a> <aiga:solution>
- <aiga:sol-2> <aiga:is-a> <aiga:solution>
- <aiga:sol-3> <aiga:is-a> <aiga:solution>
- <aiga:sol-4> <aiga:is-a> <aiga:solution>
- <aiga:sol-5> <aiga:is-a> <aiga:solution>
- <aiga:sol-6> <aiga:is-a> <aiga:solution>

- <aiga:sol-1> <aiga:is-greater-than> <aiga:sol-2>
- <aiga:sol-2> <aiga:is-greater-than> <aiga:sol-3>
- <aiga:sol-3> <aiga:is-greater-than> <aiga:sol-1>
- <aiga:sol-3> <aiga:is-greater-than> <aiga:sol-4>
- <aiga:sol-4> <aiga:is-greater-than> <aiga:sol-5>
- <aiga:sol-5> <aiga:is-greater-than> <aiga:sol-6>
- <aiga:sol-6> <aiga:is-greater-than> <aiga:sol-4>

The More, The Merrier (III/III)

- With this new approach to the triples we can:
  - Reason about the partial ordering (as before)
  - Look what solutions are there (new)
  - Reason over the particular values (new)
- Important lessons:
  - Thinking in terms of triples is important
  - The categorization of the triples define what things you can reason about
  - The questions you can ask are highly dependent on the metadata you have
- And there is lots of other metadata left out for the aiGA example
  - Date of creation of a solution
  - What users have evaluated a solution/pair
  - …

Another Example. The Time Problem

- How can we represent time using triples?
- We want to be able to reason about:
  - Time line (events that precede or succeed a given one)
  - Use the time line as a backbone of the events
- The bad news
  - iTQL does not support reasoning on date type
  - The triples are triples, no other semantics attached
- Do not run away. Two tricks can help:
  - Japanese date encoding (Year-Month-Day-Hour-Minute-Second) allow sorting date alphabetically. Kowari DOES support text sorting!
  - A simple way to generate the time back bone

Linking Timestamps To Events

- Dating triples

  <alg:fire> <alg:is-a> <alg:event>

  <alg:fire> <alg:datetime> '20060101130504'

  <alg:rain> <alg:is-a> <alg:event>

  <alg:rain> <alg:datetime> '20050101130504'

  <alg:sun> <alg:is-a> <alg:event>

  <alg:sun> <alg:datetime> '20060111130504'

- <alg:is-a> brings all the events together
- <alg:datetime> contains the dates
- We can sort now the event with a single iTQL query

Sorting It Out

- iTQL query

```
select $s <alg:datetime> $o
from <rmi://norma.ncsa.uiuc.edu/server1#timeTmp>
where $s <alg:is-a> <alg:event> and $s <alg:datetime> $o
order by $o desc ;
```

- Returns

  [alg:sun, alg:datetime, "20060111130504"]

  [alg:fire, alg:datetime, "20060101130504"]

  [alg:rain, alg:datetime, "20050101130504"]

- Problem: There is no way to query the time line backward or forward

The Solution: A Small Change

- A new set of triples

```
<alg:fire> <alg:is-a> <alg:event>
<alg:fire> <alg:datetime> <alg:date-20060101130504>
<alg:rain> <alg:is-a> <alg:event>
<alg:rain> <alg:datetime> <alg:date-20050101130504>
<alg:sun> <alg:is-a> <alg:event>
<alg:sun> <alg:datetime> <alg:date-20070111130504>
```

- We can also sort them as before

```
select $s <alg:datetime> $o
from <rmi://norma.ncsa.uiuc.edu/server1#timeTmp>
where $s <alg:is-a> <alg:event> and $s <alg:datetime> $o
order by $o desc;
```

- Similar result (but not literals now anymore)

```
[ alg:sun, alg:datetime, alg:date-20070111130504 ]
[ alg:fire, alg:datetime, alg:date-20060101130504 ]
[ alg:rain, alg:datetime, alg:date-20050101130504 ]
```


The Solution: A Small Change

- We can run the previous query and get the sorted list of dates
- Use it to construct triple that define temporal precedence
- Using the previous results we can assemble a new set of triples

```
<alg:date-20050101130504> <alg:pre-dates> <alg:date-20060101130504>
<alg:date-20060101130504> <alg:pre-dates> <alg:date-20070111130504>
```

- Now we can ask question like:
  - Did X happen before Y?
  - What events did happened after X involving Y?
  - What events X happened before Y?
  - …

An Example

- What events followed the ***rain*** of 20050101130504

```
select $o from  <rmi://norma.ncsa.uiuc.edu/server1#timeTmp>
where walk ( <alg:date-20050101130504> <alg:pre-dates> $t and
                $t <alg:pre-dates> $q)
        and $o <alg:datetime> $q
        and $o <alg:is-a> <alg:event>
        and <alg:rain> <alg:datetime> <alg:date-20050101130504>;
```

- Answer

  [alg:fire]

  [alg:sun]

Still Awake? Questions?

Implementing Metadata Stores in D2K

Xavier Llorà & David Clutter

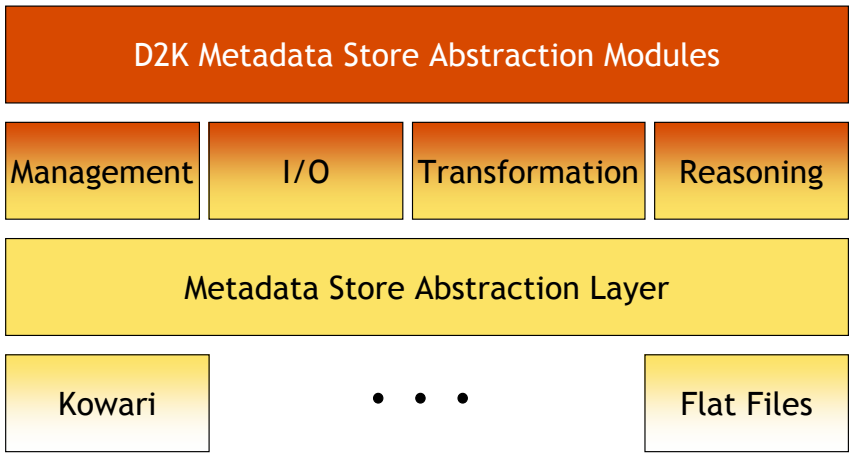
Automated Learning Group
National Center for Supercomputing Applications
University of Illinois at Urbana-Champaign

{xllora,dclutter}@ncsa.uiuc.edu

Where Are We Going?

- Three goals:
  1. Define an abstraction of a metadata store
     - Allow exchange the implementation of the metadata store
     - Provide a generic basic interface to build on top
  2. Implement functionalities for
     - Metadata store management
     - Data transformation and I/O
     - Reasoning
  3. Provide the modules for archiving, retrieving, and reasoning using metadata stores.

D2K Metadata Store Architecture



Management

- Modules for managing a metadata store
  - Connect to a metadata store
  - List the models stored in a metadata store
  - Set the store name
- Modules for managing models in a metadata store
  - Create a model
  - Drop a model
- Auxiliary management modules
  - Supply a String

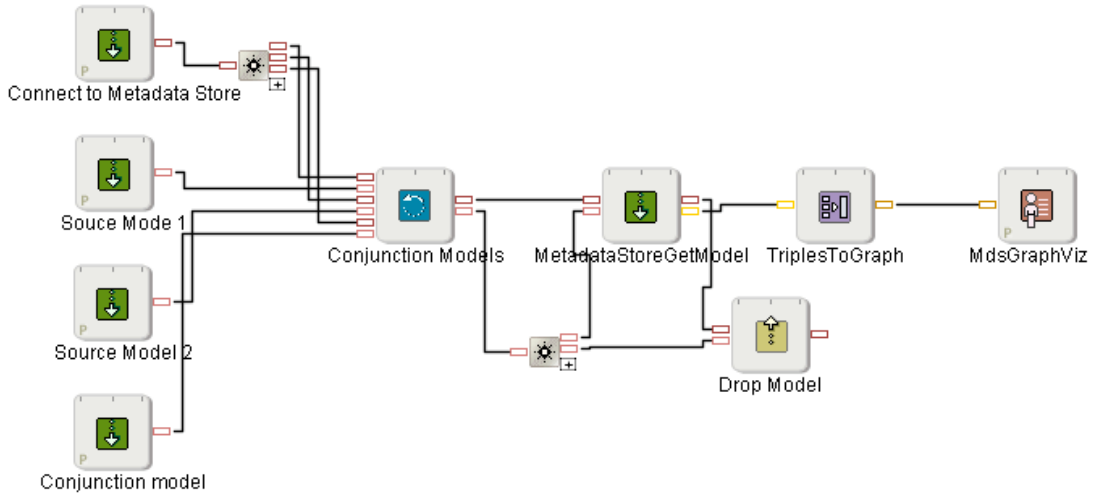
I/O And Transformations

- Input module for triples
  - Get the triples for a metadata store
- Output module for triples
  - Push a set of triples into a model archived in a metadata store
- Transformations and visualization for integration in D2KT
  - Transform a table into triples
  - Transform triples into a graph
  - Visualize the graph of converted triples
- The graph visualization uses jung currently (eventually Prefuse)

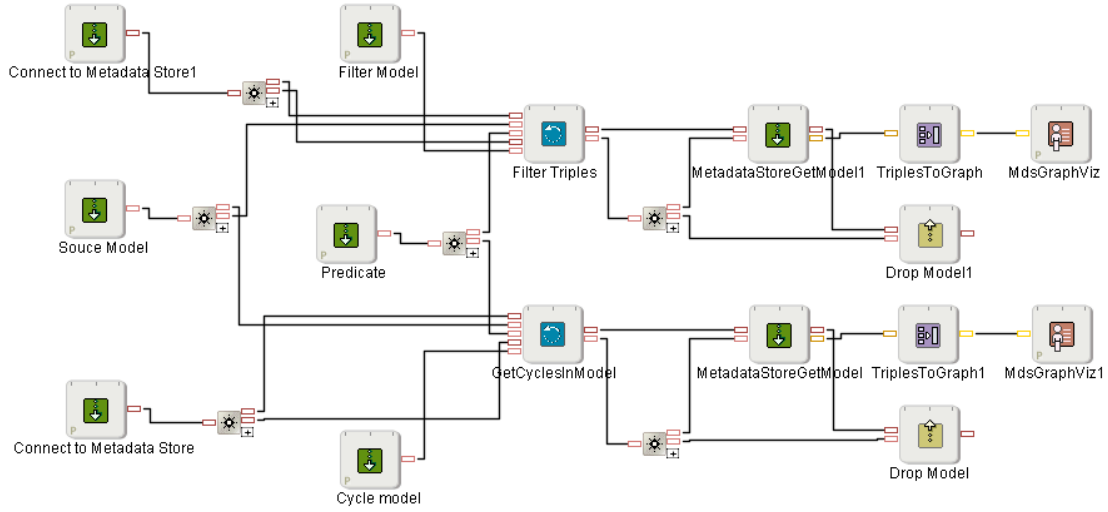
Reasoning

- Provide basic reasoning functions:
  - Filter triples out of a model
  - Compute the union of two models
  - Compute the conjunction of two models
  - Extract cycles out of a model
- Allow assembling of new reasoning patterns

Conjunction of Two Models



Extract Cycles Out Of a Model



A Live Demo