An Approach to Information Integration Based on the AMN Formalism

M. G. Manukyan

Yerevan State University, Armenia

First Workshop on Programming the Semantic Web

November 11, 2012, Boston, USA
Goals

- To use two or more databases (data sources) in the frame of a large database, possibly virtual, containing information from all sources, so the data can be queried as a unit.
Agenda

- Problems of Information Integration
- State of the Art
- Canonical Model and Principle of its Extension
- AMN Formalization of Data Model
- Mapping from RDM into Canonical Model
- Summary & Future Work
Problems of Information Integration

- **Data Type Differences** The same property is presented by different types;
- **Value Differences** The value of the same property have different encodings;
- **Semantic Differences** Terms may be given different interpretations;
- **Missing Values** A source might not record information of a type that all or most of the other sources provide.
Common Approaches

- **Federated Databases** Everybody talks directly to everyone else;

- **Warehousing** Sources are translated from their local schema to a global schema and copied to a central DB;

- **Mediation** Transforms a user query into a sequence of source queries.
Federated Database Systems

Diagram showing the concept of wrappers connecting different database systems.
Data Warehouses

Warehouse

Wrapper

Source 1

Wrapper

Source 2
Two Mediation Approaches

- **Global as View** Global schema is defined as a view over the data sources;
- **Local as View** Sources are defined in terms of global schema; mediator finds all ways to build query from views.
Example: Local View (1)

The mediator supports a \texttt{par(c,p)} relation:

- Source 1 provides some parent facts:
  \[ V_1(c,p) \leftarrow \texttt{par(c,p)} \]

- Source 2 supports only grandparent facts:
  \[ V_2(c,g) \leftarrow \texttt{par(c,p)} \text{ AND } \texttt{par(p,g)} \]
Example: Local View (2)

- **Query** (great-grandparents):
  
  \[
  \text{ggp}(c,x) \leftarrow \text{par}(c,u) \text{ AND par}(u,v) \text{ AND par}(v,x)
  \]
  
  \[
  \text{Sol1}(c,x) \leftarrow \text{V1}(c,u) \text{ AND V1}(u,v) \text{ AND V1}(v,x)
  \]
  
  \[
  \text{Sol2}(c,x) \leftarrow \text{V1}(c,u) \text{ AND V2}(u,x)
  \]
  
  \[
  \text{Sol3}(c,x) \leftarrow \text{V2}(c,u) \text{ AND V1}(u,x)
  \]
LAV and GAV Comparison

- **GAV** approach has better query processing capabilities (simpler to implement);
- **LAV** is more extensible (new data sources easier integrated).
GLAV

GLAV approach combines the GAV and LAV approaches:

- Provides direct access to data sources;
- Provides capability to define global schema as a view over the data sources.
Theoretical Basis (1)

Works of SYNTHESIS Group (http://synthesis.ipi.ac.ru):

- **AMN** is used to formalize data models;
- The canonical model should be extensible;
- A kernel of the canonical model is fixed. For each specific data model $M$ of the environment an appropriate kernel extension is defined axiomatically;
- The canonical model is synthesized as a union of extensions, constructed for models $M$ of the environment.
Theoretical Basis (2)

In the frame of the considered formalism is offered to construct:

- A mapping from $M_s$ into an extension $M_t$;
- AMN semantics of $M_s$;
- AMN semantics of the extended $M_t$.

After that the B - technology is applied to proof that $M_s$ is a refinement of the extension of $M_t$. We say that specification $S_1$ refines specification $S_2$, if it is possible to use $S_1$ instead of $S_2$ so that the user of $S_2$ does not notice this substitution.
XDM as Kernel of Canonical Model

The following kinds of XDM objects are considered:

- **Basic Objects** constants of atomic types, variables of different sorts, etc.
- **Compound Objects** are defined in terms of application and binding in $\lambda$ - calculus.
Type System of XDM (1)

- To define metadata;
- To formalize the signature of XDM symbols;

Formalism:
  - term calculus;
  - inference rules of judgments.
Type System of XDM (2)

- **Basis** consists of conventional atomic types, for example, `int`, `string`, `boolean`, etc.

- **Attribution.** If \( v \) is a basic object variable and \( t \) is a typed object, then `attribution(v, type t)` is a typed object.
Type System of XDM (3)

Example:

```xml
<attrbution>
  <birthdate />
  <type cd="ecc" />
  <date cd="ecc" />
</attrbution>
```
Type System of XDM (4)

- **Abstraction.** If \( v \) is a basic object variable and \( t, A \) are typed objects, then \( \text{binding}(\lambda, \text{attribution}(v, \text{type } t), A) \) is typed object;

- **Application.** If \( F \) and \( A \) are typed objects, then \( \text{application}(F, A) \) is typed object;

- **Function Space.** If \( t \) and \( u \) are typed objects, and \( v \) is a basic variable object, then \( \text{binding}(\Pi \text{Type}, \text{attribution}(v, \text{type } t), u) \) is typed object.
Type System of XDM (5)

(Example)

- attribution
  - addressType
    - type
      - application
        - sequence
          - attribution
            - street
              - type
                - string
            - attribution
              - city
                - type
                  - string
Type System of XDM (6)

(Example)

PiType \text{attribution \ attribution} \text{integer}

\text{binding}

\text{x \ type \ integer} \quad \text{y \ type \ integer}

\text{integer} \times \text{integer} \rightarrow \text{integer}
Kernel Extension Principle (1)

- The extension of canonical model is formed by consideration of each new data model by adding new symbols to its DDL (symbols are used for data models concepts representation);

- For applying a concept on the canonical model level the following rule is proposed:

  Concept ← Symbol Context Definition
Kernel Extension Principle (2)

(Example)

- To support the concepts of referential integrity and key's of relational data model, we have expanded the kernel with the following symbols: key, unique, foreign key, constraint, on update, on delete, cascade, and set null.

- \( S = \{\text{Snum, Sname, Status, City}\} \)
- \( P = \{\text{Pnum, Pname, Color, Weight, City}\} \)
- \( SP = \{\text{Snum, Pnum, Qty}\} \)
Kernel Extension Principle (3)

(Example)

\[ S \leftarrow \text{attribute}(S, \text{type } \text{TypeContext}, \text{constraint } \text{ConstContext}) \]

\[ \text{TypeContext} \leftarrow \text{application}(\text{sequence}, \text{AppContext}) \]

\[ \text{AppContext} \leftarrow \text{attribute}(\text{Snum}, \text{type int}), \]
\[ \text{attribute}(\text{Sname}, \text{type string}), \]
\[ \text{attribute}(\text{Status}, \text{type int}), \]
\[ \text{attribute}(\text{City}, \text{type string}) \]

\[ \text{ConstContext} \leftarrow \text{attribute}(\text{ConstName}, \text{key } \text{Snum}) \]
Kernel Extension Principle (4)

(Example)

\[ SP \leftarrow \text{attribution}(SP, \text{type TypeContext}, \text{constraint KConstContext}, \text{constraint FKConstContext1}, \text{constraint FKConstContext2}) \]

...

\[ KConstContext \leftarrow \text{attribution}(ConstName, \text{key application(list, Snum, Pnum))} \]

\[ FKConstContext1 \leftarrow \text{attribution}(ConstName, \text{foreign key application(ref, ConstNameOFS), on update cascade, on delete cascade}) \]
AMN as Formal Language of Specification

AMN is a state-oriented formalism for software development and consists of the following notations:

- the logical notation;
- the basic set notation;
- the relational notation;
- the mathematical object notation;
- the generalized substitution notation.
Basic Constructions of AMN

- Abstract Machine;
- Refinement;
- Implementation.
Abstract Machine Notation

MACHINE MachineDeclaration
USES MachineList
SEES MachineList
INCLUDES InstantiatedMachineList
EXTENDS InstantiatedMachineList
SETS SetList
CONSTANTS ConstantList
PROPERTIES Predicate
VARIABLES VariableList
INVARIANTS Predicate
INITIALISATION Substitution
OPERATIONS OperationList
END
Refinement

REFINEMENT r
REFINES m
SEES sm
INCLUDES im
SETS s
CONSTANTS c
PROPERTIES P(s,c)
VARIABLES x
INVARIANTS I(x)
INITIALISATION S
OPERATIONS O_1; O_2;... ; O_n
END
REFINEMENT $M$
REFINES $K$
CONSTANTS $c_M$
PROPERTIES $P_M$
VARIABLES $v$
INVARIANTS $I_M$
INITIALISATION $Init_M$
OPERATIONS
  $y \leftarrow op(x) =$
  PRE $Pre_{op,M}$
  THEN $Def_{op,M}$
END

...
Formalization of the Concept Refinement in AMN (2)

**Definition** We say that $M$ refines $N$ if holds the following *proof obligations*:

- $P_M \land P_N \Rightarrow \exists(v, w) \bullet (I_M \land I_N)$
- $P_M \land P_N \Rightarrow [\text{Init}_N] \neg ([\text{Init}_M] \neg I_N)$
- $P_M \land P_N \land I_M \land I_N \land \text{Pre}_{op,M} \Rightarrow$
  
  \[ \text{Pre}_{op,N} \land [\text{Def}_{op,N}\{y \rightarrow y'\}] \neg ([\text{Def}_{op,M}] \neg (I_N \land y' = y)) \]
Our approach to information integration assumes to create:

- **AMN** semantics for XDM (mapping form XDM to AMN);
- **AMN** semantics for each source data model (mapping form source data model to AMN);
- The canonical model extension by concepts of the source data model which are not yet present in target model.

After that the B - technology is applied to proof that source data model is a refinement of the extension of target model.
First of all we introduce an abstract machine in which some common concepts of data models are included.

MACHINE
  DatamodelContext
SETS
  OBJ, Types
ABSTRACT_CONSTANTS
  atomicTypes, extent, typeOf
PROPERTIES
  atomicTypes ∈ P(OBJ) ∧ extent ∈ Types → P(OBJ) ∧
  typeOf ∈ OBJ → Types
END
AMN Formalization of Data Model (3)
(AMN semantics for XDM)

For each data model an abstract machines hierarchy is created which represents AMN semantics for that model. We use the following definition of XDM schema to create the AMN semantics for XDM:

**Definition** We say that $S$ is a schema, if

$S = \langle \text{name}, \text{atomictype}, f \rangle$ or

$S = \langle \text{name}, \text{typeOp}(S_1, S_2, \ldots, S_n), f \rangle$, and $S_i$ is a schema, where $\text{typeOp} \in \{\text{sequence, choice, all}\}$, $f \in \{?, *, +, \perp\}$, $1 \leq i \leq n$. 
MACHINE XDM_Schema
(p_name, p_operationType, p_argOfOperation, p_frequency)

CONSTANTS Frequency, OperationTypes, FunctionSpace

PROPERTIES
Frequency = {?, *, +, ⊥} ∧
OperationTypes = {sequence, choice, all, empty} ∧
FunctionSpace = seq(Types) x Types

VARIABLES name, operationType, argOfOperation, frequency

INVARIANT
name ∈ STRING ∧ operationType ∈ OperationTypes ∧
argOfOperation ∈ P(extent(XDM_Schema)) ∧
frequency ∈ Frequency

CONSTRAINTS p_name ∈ STRING ∧ p_operationType ∈ operationTypes ∧ …

INITIALISATION name := p_name ∧ operationType := p_operationType ∧ …

END
Extension of the Canonical Model

MACHINE $M_{ch_T}$
SETS $S_T$
CONSTANTS $C_T$
PROPERTIES $P_T$
VARIABLES $V_T$
INVARIANT $I_T$
INITIALISATION $S$
END

MACHINE $Concepts_{ST}$
SETS $S_{ST}$
CONSTANTS $C_{ST}$
PROPERTIES $P_{ST}$
END

MACHINE $M_{ch_{ST}}$
EXTENDS $M_{ch_T}$
SEES $Concepts_{ST}$
END

REFINEMENT $M_{ch_S}$
REFINES $M_{ch_{ST}}$
SEES $Concepts_{ST}$
SETS $S_s$
CONSTANTS $C_s$
PROPERTIES $P_s$
VARIABLES $V_s$
INVARIANT $I_s \land Cond_s$
INITIALISATION $S$
END
MACHINE
   REL_Schema
VARIABLES
   schemaName, state
INARIANT
   schemaName ∈ STRING ∧
   state ∈ P(STRING × atomicTypes)
END
AMN Semantics of Relational Data Model (2)

MACHINE

   REL_DB_Schema

VARIABLES

   dbSchemaName, schemas

INARIANT

   dbSchemaName ∈ STRING ∧
   schemas ∈ P(extent(REL_Shema))

END
Mapping from Relational Data Model into Canonical Model (1)

MACHINE
    REL_Canonical_Mapping
USES
    REL_Schema, REL_DB_Schema, XDM_Schema
OPERATIONS
xdm_Schema ← REL_attr_to_XDM(attr) =
PRE
    attr ∈ string × atomicTypes ∧ xdmSchema ∈ extent(XDM_Schema)
THEN
    xdmSchema = XDM_Schema(dom(attr), ran(attr), ∅, ⊥)
END
Mapping from Relational Data Model into Canonical Model (2)

\[ \text{xdm\_Schema} \leftarrow \text{REL\_Schema\_to\_XDM}(\text{relSchema}) = \]

**PRE**

\[
\begin{align*}
\text{relSchema} & \in \text{extent}(\text{REL\_Schema}) \land \\
\text{xdmSchema} & \in \text{extent}(\text{XDM\_Schema})
\end{align*}
\]

**THEN**

\[
\text{xdmSchema} = \text{XDM\_Schema}(\text{name}, \text{sequence}, \text{recSequence}, \perp)
\]

**END**

\[ \text{xdm\_Schema} \leftarrow \text{REL\_DB\_Schema\_to\_XDM}(\text{relDbSchema}) = \]

**PRE**

\[
\begin{align*}
\text{relDbSchema} & \in \text{extent}(\text{REL\_DB\_Schema}) \land \\
\text{xdmSchema} & \in \text{extent}(\text{XDM\_Schema})
\end{align*}
\]

**THEN**

\[
\text{xdmSchema} = \text{XDM\_Schema}(\text{name}, \text{sequence}, \text{recSequence}, \perp)
\]

**END**

**END**
DML Formalization Principles in AMN

MACHINE
   RelationalAlgebra
USES
   Relation
OPERATIONS
s ← union(r, q) =
PRE
   r ∈ extent(Relation) ∧ q ∈ extent(Relation) ∧
   s ∈ extent(Relation) ∧ isUnionCompatibility(r, q)
THEN
   s.schema := r.schema || s.tuples := r.tuples ∪ q.tuples
END
Summary & Future Work

**Canonical Model**
- The XDM has been formalized in the frame of the AMN,
- Principle of extension of canonical model has been proposed and formalized in the frame of AMN.

**Example of Mappings**
- The RDM has been formalized in the frame of AMN,
- Mapping from AMN representation of RDM to canonical model has been proposed.

**In the future we plan to develop**
- Algorithms for generation of mappings from the source models into canonical model,
- Temporal extension of canonical model.
THANK YOU FOR YOUR ATTENTION!