Fusionplex: Resolution of Data Inconsistencies in the Integration of Heterogeneous information Sources

Amihai Motro
Philipp Anokhin

Information Fusion
Vol. 7, No. 2, June 2006
Pages 176-196
Information Integration and Conflicts

✓ **Information Integration**: Given a collection of *heterogeneous* and *autonomous* information sources, provide a system that allows its users to perceive the entire collection as a single source, query it transparently, and receive a *single, unambiguous answer*.

✓ **Information conflicts**: The sources may have conflicts of two types:

  - *Intensional (semantic) inconsistencies*: Differences in
    - The data models,
    - The schemas within the same data model,
    - The data is in different languages, or in different measurement systems.
  - *Extensional (data) inconsistencies*: Factual discrepancies among sources that describe the same data objects.
    - Can be observed only after intensional inconsistencies have been resolved.
    - Most information integration systems deal with semantic inconsistencies only.
Fusionplex

- Resolve extensional inconsistencies by means of *data fusion*.
- Assumes the Multiplex information integration system.
- Basic principles of Multiplex:
  - Global schema in the relational model $R_1, \ldots, R_n$.
  - The global schema is *mapped* to a collection of local schemas, by means of *view pairs*.
  - Each pair associates a view of the global schema $V_i$ with a view of the a local schema $U_i$.
  - The view $V_i$ is materialized by the view $U_i$.
  - A query $Q$ on the global relations $R_1, \ldots, R_n$ must be translated to a query over the materializable views $V_1, \ldots, V_m$. 
Multiplex Advantages

✓ Mapping of global schema to local schemas is not total:
  • Framework for situations where some local databases to become unavailable.

✓ Mapping of global schema to local schemas is not onto:
  • Framework for partial integration of local databases.

✓ Mapping of global schema to local schemas is not single-valued:
  • Framework for extensional inconsistencies: multiple materializations of the same global view.

✓ Local databases need not be relational: Heterogeneity.
  • The relational model is only a global description, and a communication protocol.
  • The global database requests and receives information in tables.
How Multiplex Handles Inconsistencies

✓ **Inconsistency**: A global query is answered by multiple *sets of records* (tuples).

✓ Multiplex provides an approximation for the *true* answer by “sandwiching” it between two answers:
  - The *lower bound* set of records that indicates a *sound* answer: Each tuple in this answer is in the true answer.
  - An *upper bound* set of records that indicates a *complete* answer: All tuples of the true answer are in this answer.

✓ These two estimates are obtained in a process similar to *voting*:
  - All tuples in the *union* of answers are sent to the complete upper bound.
  - Tuples showing in *intersection* of answers are sent to the sound lower bound.
Limitations of the Multiplex Approach

✓ Inconsistencies are identified at the record (tuple) level:
✓ When the same real-world object is described slightly differently in two tuples, they are assumed to describe two different objects.
✓ Hence, they are both relegated to the upper bound estimate.
✓ No attempt at object identification.
The Fusionplex Approach

 ✓ Not all data are created equal: The data environment is not egalitarian, where all information sources have the same qualifications.

 ✓ Individual sources have their individual advantages and disadvantages.
  • Some data are more recent.
  • Some data come from more authoritative sources.
  • Some data are more costly to obtain.
  • Etc.

 ✓ To resolve conflicts, Fusionplex considers the qualifications of the providers.
  • It uses meta-data to resolve data conflicts.
Where Does the Meta-data Come from?

✓ Some may be provided by the sources themselves.
  • Date of update, cost, …
✓ Some may be obtained from Web sites dedicated to the evaluation of the quality of other sites.
  • Usually by summarizing the evaluations of the community of users.
✓ In a restricted environment, A multidatabase administrator could assign and maintain the meta-data scores of the local sources.
Information Features

✓ Our term for meta-data is features.
✓ Examples include
  • Timestamp: The time when information was validated.
  • Cost: The time to transmit the information, or the money paid, or both.
  • Accuracy: Probabilistic information denoting the accuracy.
  • Availability: Probability that at a random moment the source is available.
  • Clearance: Security clearance needed to access the information.
Users Control Inconsistency Resolution

✓ Inconsistency resolution requires the following input:

• User provides (in each query)
  o A pruning predicate to remove from the answer non-performing data, and
  o A vector of feature weights expressing their utility to the user.

• System provides a policy for resolving individual attributes.
  o Policies is part of the global database design.
  o Could be modified by users.

• SQL has been extended to allow users to specify such input.
Formal Framework

✓ A global set of features $F$ is defined.
✓ Each feature is associated with a domain of values. For example,
  • Availability: the interval $[0,1]$.
  • Clearance: \{top-secret, secret, confidential, unclassified\}.
✓ Top facilitate comparisons, all features are \textit{normalized} by mapping them to the interval $[0,1]$.
  • Worst feature value: 0.
  • Best feature value: 1.
  • Source with timestamp $= 0.7$ is more recent than source with timestamp $= 0.4$.
  • Source with cost $= 0.7$ is cheaper than source with cost $= 0.4$.
✓ Each source is associated with a set of values for the entire feature set $F$.
  • Null values are used, if the source does not have a particular feature.
Homogeneity Assumption

✓ Sources are *homogeneous* with respect to every feature.
  • All features values are associated with the *entire* source.
  • Each feature value is *inherited* by each relation, tuple and field.
    o For example, the same *timestamp* is associated with every bit of data.

✓ This limitation can be avoided by *splitting* a source into several parts, each with its own features.

✓ Small change to Multiplex:
  • Mapping pairs are extended to triplets:
  • The third element of each mapping is the set of source features.
  • Every source provides its meta-data along with the data.
Extended Relational Model and Algebra

- Each relation scheme is extended with all the features of \( F \), as new attributes.
- Each tuple is extended correspondingly, with values for these features.
- The relational algebra operators selection, projection, and Cartesian product are extended to handle these new attributes.
- The relational algebra operators union and difference remain unchanged.
Selection, Projection, Cartesian Product

✓ Selection
  • Uses an additional predicate to select tuples by their feature values.

✓ Cartesian product
  • Concatenates data values, but fuses feature values.
  • Fusion method depends on the feature. For example,
    o Cost is the sum of the costs of the two components (must be normalized to the interval [0,1]).
    o Availability is the product of the availabilities of the two components.
    o Timestamp is the minimum of the timestamps of the two components (reflects worst-case approach).
  • Feature value fusion must consider possible null values. Consider timestamps.
    o Null value indicates any value in the range [0,1].
    o When null is merged with specific value $x$, their minimum is the interval [0,$x$].
    o While there’s more information, it’s still a null.

✓ Projection
  • Feature attributes cannot be removed.
  • After removing attributes (as usual), it resolves inconsistencies in the result (next).
Data Collection and Assembly

- Amalgamate the triplets of different mappings in a single set: \( \{(V,URL,f)\} \)
- Each triplet is a *contribution*.
- Determine the contributions *relevant* to the query (discover overlaps).
- Convert a relevant contribution to a *query fragment*:
  - Remove tuples and attributes not requested in the query.
  - Add null values for attributes missing from the contribution.
- The union of all query fragments is called *polyinstance*.
  - Intuitively, all information culled from the data sources in response to the query.
- *Inconsistency resolution*: Convert the polyinstance to a regular instance.
Inconsistency Detection and Resolution

✓ Two phases:
  • Phase one: Inconsistency detection.
  • Phase two: Inconsistency resolution.

✓ Inconsistency detection:
  • Scan the polyinstance and find the tuples that are versions of each other.
  • Resulting clusters are called polytuples.
  • Different identification methods can be used.
  • We use the simplest: We assume identification by keys.

✓ Inconsistency resolution requires 4 inputs:
  • A tuple of feature weights \( w \) (specified in the query).
  • A feature-based selection predicate (specified in the query).
  • A utility threshold (pre-defined).
  • A resolution policy for every global schema attribute (pre-defined).
Polytupple Resolution: Steps 1 and 2

✓ Step 1: Selection
  • Remove members that do not satisfy the feature-based selection predicate.

✓ Step 2: Discard non-performers:
  • The user query provides a vector of weights for the different features: \( w_i \).
  • Each polytupple member has a vector of values for the different features: \( f_i \).
  • Each tuple in a polytupple receives a utility value, by calculating \( \sum_i w_i \cdot f_i \).
  • These values are used to rank the members.
  • Using the utility threshold, members with insufficient utility are discarded.
Polytuple Resolution: Steps 3 and 4:

✓ **Step 3**: Resolve a polytuple.
  - Within each polytuple, data inconsistency is resolved in each attribute separately:
    - Each column of values is fused in a single value.
    - The resolution policy for that attribute is applied.

✓ **Step 4**: Calculate the feature values of each resulting tuple.
  - Similar to how it has been done in the Cartesian product.
Resolution Policy

✓ Each resolution policy is a sequence of
  • *Elimination functions*, followed by
  • *Fusion function*.

✓ Examples of elimination functions:
  • $\text{max}(\text{timestamp})$ (feature-based)
  • $\text{max}(\text{availability})$ (feature-based)
  • $\text{min}(\text{cost})$ (feature-based)
  • $\text{max}(\text{salary})$ (content-based)

✓ Examples of fusion functions (all content-based):
  • *average*
  • *any*
  • *average-without-extreme-values*
Fusionplex Architecture