DATA INTEGRATION SERVICES

Christian Convey, Olga Karpenko, Nesime Tatbul and Jue Yan

1 Introduction

** subject to modifications

1.1 Definition of Data Integration

Data integration systems harmonize data from multiple sources into a single coherent representation. The goal is to provide an integrated view over all the data sources of interest and to provide a uniform interface to access all of these data. The access to the integrated data is usually in the form of querying rather than updating the data.

The data sources to be integrated may belong to the same enterprise or may be arbitrary sources on the web. Most of the time, each of the sources is independently designed for autonomous operation. Also, the sources are not necessarily databases; they may be legacy systems (old and obsolescent systems that are difficult to migrate to a modern technology) or structured/unstructured files with different interfaces. Data integration requires that the differences in modeling, semantics and capabilities of the sources together with the possible inconsistencies be resolved.

1.2 Motivation for Data Integration Systems

- Historical view: integration-by-hand
- Users can focus on specifying *what* they want, not on *how* to obtain what they want. Instead of finding relevant sources, interacting with every source and combining data from different sources, a user can ask queries in a unified way.
  Particular examples:
  - Desire for reports that describe all parts of a merged organization (bank mergers, car dealerships, etc.).
- Facilitates decision support applications (OLAP, Data mining)

  OLAP (On-Line Analytical Processing) is making financial, marketing or business analysis to be able to make business decisions on a collection of detailed data from one or more data sources. The analysis is done through asking large number of aggregate queries on the detailed data.

  Data Mining is discovering knowledge from a large volume of data. Statistical rules or patterns are automatically found from the raw collection of data.

1.3 Major Issues

- Heterogeneity of data sources
- Availability of data sources
- Dynamicity of individual data sources
- Autonomy of data sources

- Correctness of the integrated view of the data
- Query performance

1.4 Summary of State of the Art

2 Data Integration Architectures

2.1 Dimensions to categorize architectural models for integrating data sources

- Autonomy
  Autonomy refers to the degree which individual data sources can operate independently.
  - Design autonomy
    The source is independent in data models, naming of the data elements, semantic interpretation of the data, constraints etc.
  - Communication autonomy
    The source is independent in deciding what information it provides to the other components that are part of the integrated system and to which requests it responds.
  - Execution autonomy
    The source is independent in execution and scheduling of incoming requests.
- Heterogeneity
  Heterogeneity refers to the degree of dissimilarity between the component data sources that make up the data integration system. It occurs at different levels. On a technical level, heterogeneity comes from different hardware platforms, operating systems, networking protocols or similar lower-level concepts. On a conceptual level, heterogeneity comes from different programming and data models as well as different understanding and modeling of the same real-world concepts (ex: naming).
- Distribution
  Distribution refers to the physical distribution of data over multiple sites.
  - Client/Server
    Server does data management, client provides user interface.
  - Peer-to-Peer (fully distributed)
    Each machine has full functionality of data management.
- Transparency
  Transparency refers to the separation of higher-level semantics of a system from lower-level implementation issues. A transparent system hides the implementation details from users.

2.2 Major Approaches to Data Integration

Three common approaches that data sources can be made to work together:

- Virtual View Approach
  No data is stored in the integrated database. Data is accessed on-demand (lazy approach).
- Materialized View/Warehousing Approach
  Information is stored in a repository (warehouse) to be queried later (eager approach).
- Hybrid approach
  Data is selectively materialized.

2.3 Virtual View Approach

Problems common to all virtual view systems will be described first.

2.3.1 Federated Database Systems

The sources are semi-autonomous and the distribution is peer-to-peer. Each source asks all others in the federation to supply information when needed. One-to-one connections between all pairs of databases, where each database has interface to the others. The software on the top of federation allows user to access it in unified manner.

- Picture
- Schema integration
  ** We will briefly mention it here and refer to the following section about schema integration.

2.3.2 Mediated Systems

Definition from David Martin's paper (Information Broker Project):
"Mediation is a process that permits a user to get information from a wide variety of sources, without having to be aware of the identities, locations, schemas, access mechanisms, or contents of these sources. A component that performs mediation presents a single schema (called **mediated schema** - our comment) to its requesters, accepts queries expressed in that schema, and handles all the details of getting the appropriate data from relevant information sources (each of which is likely to operate with a different schema)."

- Picture
- Steps in how the queries are answered
  - Receive a query formulated on the unified schema from the user
  - Query reformulation
  - Optimization and execution
  - Translation

  (** For detailed description of the steps above refer to the section on query processing)
- Wrapper as an important component - hides technical and data model heterogeneity.
- Additional problems raised by this specific architecture
- Known implementations
  - TSIMMIS
  - Information Manifold
  - SIMS
  - Carnot

  (** very briefly, may be just mention)

2.4 Materialized View Approach (Data Warehousing)

Filtered data from several sources is stored in a single repository (called "data warehouse"). This data is combined into a unified schema. A data warehouse often contains historical and summarized data which is used for decision support.

- Picture

- Motivation: Why would we want to materialize data?
  OLTP (On-Line Transaction Processing) vs OLAP
  - Terabytes of data
  - Workloads are query intensive; queries are complex
  - OLAP data is modeled multi-dimensionally
- What is involved in building a data warehouse?
  (** For each of this steps mention the main research issues)
  - Modeling and design
  - Maintenance (creation and refreshing)
  - Operation
- Known implementations
  - Squirrel
  - WHIPS

2.5 Hybrid approach

- What data to materialize?
- How data is maintained
- Known implementations

(** refer to section on Materialized View Management for a detailed discussion of selection of views to materialize)

2.6 Comparative strengths and weaknesses of each approach/architecture

3 Data Models and Schema Integration

3.1 Data Models

A **data model** is an abstract description of how information is encoded in a system. Typically, people use data models to perform certain reasoning about a system, especially when doing design work regarding the system.

Data models are helpful because they are simplified versions of the system that the model corresponds to. Data models can omit details about the modeled system that are irrelevant to the task at hand. This can make work easier for a person working with the modeled system.

A data model is valid (with respect to its author's intentions) if it accurately describes all facets of the modeled system that the data model's author intended the model to describe.

For example, an entity-relationship (ER) model can be a valid model of an associated relational database.

A data model might not represent all of the interesting information that exists in the system that the model is an abstraction of. For example, an ER model would not likely describe how many records appear in a given table of the relational database.

A data model (as conceived here) does not necessarily attempt to describe dynamic aspects of the information encoded a system. For example, a data model might describe that a system had four explicit states: "STARTING", "RUNNING", "QUIESCING", and "SHUTDOWN".

A data model probably would *not* describe the possible sequences of state transitions that the modeled system could undergo. That kind of description would be characteristic of a state model, not a data model.

Common types of data models used in software

Numerous types of data models have been studied and put into use. Some of the most common ones are described below.

- Entity-Relationship (ER) data models
  These models describe a system as a set of entity classes and a set of classes of potential relations between entities.
- Object-oriented data models
  These models describe classes of objects, relationships that will exist between instances of those classes, *and* services that will be provided by objects.

  Object-oriented models therefore tend to be more than just data models. Object-oriented models go one step further by describing to some extent what dynamic behavior the modeled system can undergo when executing.
- Hierarchical data models
  These models represent information as a tree-like hierarchy, composed of nodes and edges. Information may be encoded in such details as node values, edge labels, and the shape of the tree.

3.2 Schemas

A **schema**, as one would expect to find in a database management system (DBMS), is a data model of the information stored in the database.

What typically distinguishes schemas from more general data models is that schemas may provide storage information that is irrelevant to the data model itself.

Examples include index definition and where database files reside on the file system.
Common types of schemas used in software

- Relational schemas
  These are much like E-R models, except that they can be specific about storage structures.

  Example: SQL Data Manipulation Language
  It has structure, but is not convenient for software to parse the argument's structure.

- Object-oriented database schemas
  These are much like object-oriented data models.

  Additional implementation details may include the source code of the methods associated with classes or instances of objects.

  Example: CORBA
  In CORBA, bindings are generated between details specified in a generic description of objects' interfaces (IDL) and the actual program code that implements those objects. That binding code, as well as the bound-to code itself, could reasonably be considered part of the schema.

- Semistructured schemas
  These are much like hierarchical data models. The additional details provided might include a canonical name for the schema (to be referred to by individual database instances), or what character set encoding is used in a database conforming to the schema.

  Example: XML DTDs

- Unstructured
  Many documents have structure that's easily recognizable to humans, but it very unclear to software. From a software system's perspective, these documents are unstructured.

  Examples

  - Plato's Republic It has structure, but is not convenient for software to parse the argument's structure.
  - A bitmap image of a molecule's shape The image contains structure, but it's quite difficult for an application to appreciate that fact.

3.3 Data Model Semantics

The **semantics** of a data model describe how the details of the data model correspond to the details of the modeled system.

Perhaps the most familiar representation of data model semantics is the use of a **data dictionary** in some data models.

3.4 Mapping Between Schemas

Some of the most complicated work of data integration is the construction of techniques that can map the data from one schema into another.

In mediated database systems and in data warehouse systems, there is a single schema used when providing the integrated view of the data. In such systems that have n different data source schemas, at most n different schema mappings must be devised and implemented.

In data warehouse systems, the schema mapping can generally be one-way. The mapping just needs to figure out how to present information in the data source in a way that's consistent with the integrated view's schema.

In mediated systems, the schema mapping must be two-way. Like with a data warehouse, a mapping from the data source schema to the integrated schema must be defined. However, mediated systems receive queries from their users, which must then be translated into queries processed by the data sources. The user formulates the query using terms from the integrated schema. The query posed to the data source by the mediator must be in terms of the data source's schema.

3.5 Hard Problems in Schema Mapping

- **Onto mappings**
  Example: Numeric grades to letter grades [give credit to the H.P. paper]

- **Different categorizations**
  I.e., Data model A records (Unit color, Number sold), and data model B records (Unit weight, Number sold). Unless color =¿ weight or weight =¿ color, there's a serious limit on any unified data model that wants to represent Number units sold in a useful way.

- **Recognizing object identity**
  Problem: Each schema implies that is only has one data object per object in the modeled system. When producing a unified database, it's possible to lost this one-data-object-per-modeled-object quality unless care is taken. Even then, it can be a hard problem to solve (** Research this better - cjc).

- **Conflicting data**
  Schema semantics often make an implicit claim that data using the schema is absolutely correct. People usually know to take that claim with a grain of salt. When unifying data from two or more data sources, contradictions can be reached, leading to an internally inconsistent view of data.

- **Unequal expressive power of schemas**
  By way of example, object-oriented databases (OODBs) might include their method behaviors as part of the schema. However, SQL schemas have only limited support for expressing system behavior.

  In particular, the OODB methods might be written in computationally complete languages, and SQL is not computationally complete. The important lesson from this is that care must be taken in the selection of which schema type to use for presenting the integrated view of the data.

3.6 A Survey of Automatible Schema Mapping

Not all schema mapping is necessarily difficult to perform with computers. Some types of schemas are nicely translatable to other types of schemas, which opens the door for using software to produce the mappings.

What makes this feasible is that in these cases, mapping patterns have been identified that preserve data model semantics without requiring the map-writing software to care about the details of the data model semantics.

Examples appear below.

- **Mapping hierarchical schemas to relational schemas**
  Consider a hierarchical type schema like HTML, and a relational type schema like SQL.

  Generally, HTML has is a tree of nodes. Each node has a set of attributes name/value pairs and a set of children nodes.

  On naive mapping is possible: Create a SQL table where one row corresponds to each node from the HTML schema. The tree structure is preserved by having a column in the table that lists the primary key of the row that corresponds to each child node.

- **Mapping relational schemas to object-oriented schemas**
  One solution is to define one object class per relation. Create one instance of each class to model each row that appear in the corresponding relation.

  Relational view definitions and calculated values can be expressed with methods in the object-oriented system.

4 Querying the Integrated Data

The traditional way of query processing involves: getting a declarative query from the user; parsing it; passing it through a query optimizer which produces an efficient query execution plan that describes how to exactly evaluate the query (i.e., apply which operators, in what order using what algorithm); and finally executing the plan.

In data integration systems this is not so straight forward because the query coming from the user is not asked on a single source but rather on an integration of data sources, each having different query formulation and processing mechanisms defined on different data models.

4.1 Data Modeling

4.1.1 Languages

Alternatives will be discussed and Datalog will be briefly introduced as the choice of query and view language for the following subsections.

4.1.2 Modeling the Mediated Schema

Mediated schema is the single schema of the integrated system which is obtained by unifying the schemas of the data sources being integrated. Queries to the data integration system are formulated on this mediated schema.

4.1.3 Modeling the Data Sources

Data source descriptions which specify the relationship between the relations in the mediated schema and those in the local schemas at the sources should be provided. The description of a source specifies its

- contents
- attributes
- constraints on its contents
- completeness and reliability
- query processing capabilities

Besides, using the descriptions of the sources, we should be able to detect the following:

- overlapping and contradictory data among sources
- semantic mismatches among sources
- differing naming conventions for data values among sources

4.2 Query Reformulation/Rewriting

Using the source descriptions, user query written in terms of the mediated schema is reformulated into a query that refers directly to the schemas of the sources (but still in the data model of the mediated system). Goals to achieve:

- Semantic Correctness of the Reformulation: The answers obtained from the sources will be correct answers to the original query.
- Minimizing the Source Access: Sources that can not contribute any answer or partial answer to the query should not be accessed. In addition to avoiding access to redundant sources, we should reformulate the queries as specific as possible to each of the accessed sources to avoid redundant query evaluation.

4.2.1 Global As View (GAV) Approach

For each relation R in the mediated schema, a query over the source relations is written which specifies how to obtain R's tuples from the sources.

4.2.2 Local As View (LAV) Approach

For each data source S, a rule over the relations in the mediated schema is written that describes which tuples are found in S. This is an application of a much broader problem called "Answering Queries using Views". (** It also applies to view design and selection in Data Warehousing.)

Query reformulation in LAV is more complex. However, it has important advantages compared to GAV: adding new sources and specifying constraints in LAV are easier.

- equivalent rewritings vs maximally-contained rewritings

- The Bucket Algorithm
- The Inverse-Rules Algorithm
- The Minicon Algorithm
- The Shared-Variable-Bucket Algorithm
- The CoreCover Algorithm
- Comparison of the Algorithms

4.2.3 Completeness and Complexity of Finding Query Rewritings

- source incompleteness
- recursive rewritings

4.2.4 Using Probabilistic Information

- for source completeness
- for overlap between parts of the mediated schema
- for overlap between information sources

4.2.5 Alternative Query Reformulation for Dynamic Information Integration

4.3 Query Optimization and Execution

Query optimization refers to the process of translating a declarative query into a query execution plan, i.e., a specific sequence of steps that the query execution engine should follow to evaluate the query. Why is it difficult in data integration systems?

- sources are autonomous; optimizer may not have any statistics or reliability info about the sources, query planning is difficult
- sources are heterogeneous; they may have different query processing capabilities
- data transfer time is not predictable due to the existence of the network environment, it is difficult to make cost estimates
- some sources may become unavailable (?)

4.3.1 Query Plan Generation

4.3.2 Query Execution

4.3.3 Adaptive Query Execution

- interleaving optimization and execution of the query in the Tukwila System

4.3.4 Query Translation

Up to now, the queries are reformulated into a form that refers directly to the data sources rather than the mediated schema. However, they are still in the data model of the mediated system. They should be translated into the data models of the data sources. This is performed by source-specific wrappers which will be defined in the following section.

(** This is the responsibility of the source wrappers. We will be connecting Query Processing section to Data Extraction section briefly discussing the query translation stage inside query execution.)

(** We should also be discussing the flow of translation in the reverse direction, i.e., after the sources successfully perform the local queries, how the resuls are collected and combined into a single answer to present to the user)

5 Data Extraction

Combines techniques from DB and AI (NLP, Machine learning).

5.1 Techniques for Extracting Data (Wrappers)

To access information from different heterogeneous data sources, we have to translate queries and data from one data model to another. This function is provided by wrappers around each individual data source. Wrapper converts queries into one or more queries understandable by the underlying data source and transforms results into the format understood by application (mediator).

5.1.1 Wrapper generation

- Issues
  - refer to the section on Data Models and Schema Integration
  - Mediator systems usually require more complex wrappers than do most warehouse systems

- Ways of creating wrappers
  - **Manual**
    Why is it impractical for some sources?
    In case of Web sources:
    - big number of sources
    - new sources are added frequently
    - format of sources change
    So, high maintenance costs.
  - **Semi-automatic (interactive)**
    Noted that only small part of the code deals with the specific access details of the source. The rest is common among wrappers or data transformation can be expressed in a declarative fashion (high-level). Graphical interface, programming by demonstration.
  - **Automatic**
    * site-specific or generic
    * usually needs training often supervised learning

- Tools for semi-automatic/automatic wrapper construction for structured/semistructured data
  - Template-based wrappers
  - Inductive learning techniques for automatically learning a wrapper (using labeled data)
    Inductive learning - task of computing some generalization from a set of examples
    Methods:
    * zero-order (decision tree learners)
    * first-order (inductive logic programming)
      - bottom-up/top-down approaches

- Tools for data extraction from unstructured documents
  - Using ontologies and conceptual models to extract and structure information from data-rich, unstructured documents.
  - Using heuristic approaches to find record boundaries in web documents.

5.1.2 Filters

If a wrapper returns a superset of what query wants, we can filter the results of the query.

5.2 Interfaces of Sources to the World

(** This section has the purpose of showing what kind of interfaces the sources provide so that wrappers can communicate with them)

5.2.1 General issues with interfaces

- **Standardized interfaces**
  Many interfaces are intended for reuse by many applications. Therefore, they separate their functionality into two groups.
  - Information regarding the reusable aspects of the interface (i.e., connections, sessions, supported units of transfer (files, whole records, etc.)).
  - Application-specific information (i.e., record layouts of tabular text files, the set of files, specific objects instances).
- **Data resolution**
  Division of application data encoding in up to two levels
  - Addressable by the interface (i.e., files via FTP, record column values via ODBC)
  - Not addressable by the interface (i.e., the FTP interface to retrieve an XML file, but FTP doesn't provide specific element extraction from that XML file.)

  Applications may have some freedom about where to encode information. For example, a FTP server for image files could serve one file per image (with a helpful filename), or could serve just one .zip file which must be opened by the client to choose the desired image file.
- **Data types**
  - Interface basic data types may or may not correspond well to the basic data types used to describe data at the data source.
  - Example In HTTP, all data is represented as a string. This works great for web pages. But if HTTP is being used to transfer a sound file, the string data type isn't particularly useful.
- **Interface semantics**
  - Data sources and data integrators are capable of encoding whatever information they choose in terms of an interface.

  - Some interface designs help provide hints to the semantics of application data moved across the interface. (I.e., ODBC allows its users to refer to specific table name, and those names might provide helpful hints as to the application semantics of the table contents.)
  - Why different interfaces suit different application data models Some interfaces are designed in a way that allows for *obvious* mapping between application data models and the interface. (I.e., ODBC with a relational database, or CORBA with an object-oriented system.)
  - Dealing with semantics of the application-specific data is covered in a another section (needs an xref)

5.2.2 Survey of Common Interfaces

It's instructive to consider some different kinds of non-application-specific interfaces.

- Non-network interfaces
  - **Difficulties**
    - Data integration may involve two or more computers that must communicate.
    - A data source that only has local interfaces may therefore require some integration software to be installed on the data source's computer
  - **Examples**
    - Writing to files on a local disk drive.
    - Displaying information on a local GUI only (i.e., MS Windows, not X).
- Network-capable interfaces

  Interfaces that allow an application to readily communicate with other applications that are either local or on a different computer.
  - **Difficulties**
    - Potentially independent failures of communication applications
    - Potentially significant communications costs / performance issues
  - **Examples**
    - FTP
    - CORBA
    - ODBC

6 Materialized View Management

We are going to discuss two major subtopics under this heading with giving higher emphasis to the second one:

- Design and selection of views to materialize
- Maintenance of the materialized views

6.1 Design and Selection of Views to Materialize

Major Goals:

- to minimize total query response time
- to minimize the cost of maintaining the selected views

6.2 The Problem of View Maintenance

6.2.1 Definition

- traditional view update problem and why this one is different and more difficult
- As the database changes because of updates applied to the base relations, the materialized views may also require change. A materialized view can always be brought up to date by re-evaluating the view definition. However, complete re-evaluation is wasteful.
  "heuristic of inertia"- only a part of the view changes in response to changes in the base relations
- steps of view maintenance:
  - propagation (computing changes to the view)
  - refreshing (applying changes to the mat. view)

6.2.2 Dimensions

- Available Information
  materialized view, base relations, other views, integrity constraints, etc.
- Allowable Modifications
  insertions, deletions, updates, sets of each, group updates, change view definition, etc.
- Expressiveness of the View Definition Language
  conjunctive queries, duplicates, aggregation, union, recursion, negation, select, project, join, spj, etc.
- Database Instance
- Complexity
  - Complexity of the View Maintenance Language
  - Complexity of the View Maintenance Algorithm
  - Complexity of Auxiliary Information (in terms of space)

6.2.3 View Maintenance Policies

when to apply maintenance procedures on the materialized views

- Immediate View Maintenance
  Refreshing is done within the transaction that changes the base data. slow transactions; faster queries and always up to date results.

- Deferred View Maintenance

  - lazily, at query time
    fast transactions; slow query speed.
  - forced, after a certain amount of change to the base data
    non up to date results; good transaction and query time.
  - periodically, in certain time intervals
    non up to date results; good transaction and query time.

comparison of all in general; how the decision when to use which one is made.

6.3 Incremental View Maintenance

- pre-update vs post-update algorithms

- various algorithms will be summarized

6.4 View Maintenance Anomalies(Consistency Issues)

- caused by decoupling btw view definition and base data

- definition of correctness and levels of correctness

- solutions:

  - recomputing the views
  - storing copies of base relations
  - ECA (Eager Compensating Algorithm)

6.5 Update Filtering

Detection of base data updates that are irrelevant to the view (i.e., have no effect on the state of the view) wil be discussed. For such updates, we do not need to perform any maintenance. Thus, update filtering makes maintenance more efficient by preventing redundant work.

6.6 View Self-Maintenance

In general, Self-Maintenance refers to views being maintained without using all the base data. There exists different notions of its exact meaning depending on how much information is avaiable. At minimum, the view update is performed at the integrated system by only knowing the particular base data update that has occurred, the view definitions and the materialied data. We will be describing the alternative notions in detail here.

- Questions

  - Given a view, is it self-maintainable?
  - If it is self-maintainable, how?

- Single-View Self-Maintenance
  does not consider the materialized views

- Multiple-View Self-Maintenance
  makes use of contents of the other materialized views to minimize base data access

- Updates as a whole
  i.e., batch updates; may make maintenance easier and more efficient

- Making views self-maintainable
  When the answer to the first question listed above is No, we can define and materialize a minimal set of auxiliary views to make the original non-maintainable view maintainable. Here, basically we are increasing the amount of information available at the integrated system level.

6.7 Dynamic View Management

- problems of static view selection and maintenance

- dynamic view selection and maintenance

- performance parameters
  - space (taken up by the materialized views)
  - workload (changing query workload)
  - maintenance window (how often we would like to perform maintainance and how long is the system tolerated to be unfunctional)

- the solution in DynaMat

7 Systems

- TSIMMIS (Stanford)
- Ariadne (USC)
- Araneus (Web-Bases, University of Rome, Italy)
- Infomaster (Stanford)
- Information Manifold (AT & T)
- Tukwila (University of Washington)
- SIMS (USC)
- Carnot (MCC)
- WHIPS (Stanford)
- Squirrel

** subject to modifications

8 Open Questions and Research Issues

** will be formed later from everybody's list of research issues from the above sections

9 Concluding Remarks

** will be written later