Conception of Information Systems
Lecture 3: Integration of Heterogeneous Databases

22 March 2005
http://lsirwww.epfl.ch/courses/cis/2005ss/

Outline

- Introduction
- Important types of Integrated Database Systems
- Wrappers: Transparent Access to Data Sources
- Schema Integration
  - Schema Integration Method
  - – Identification of Correspondences
  - – Resolution of Conflicts
  - – Implementation of an Integrated View
  - – Processing of Queries and Updates
- Summary
- References

In this part of the lecture we will learn more about the technical details of integrating information systems at the data level, or in other words about data or database integration. We will see later approaches, which allow access to different databases from within the same application (e.g. JDBC, transaction monitors, or workflow management systems). Thus it is important to understand the difference between database integration and access to different databases. With database integration the integrated DBMS provides data processing functionality (e.g. querying, updating) for the integrated database as if it were managed by a standard DBMS. There exist many situations where data processing can be done much more efficiently on the integrated database (or is the only possiblitiy) than if it would be performed on each of the component databases, i.e. the databases to be integrated.

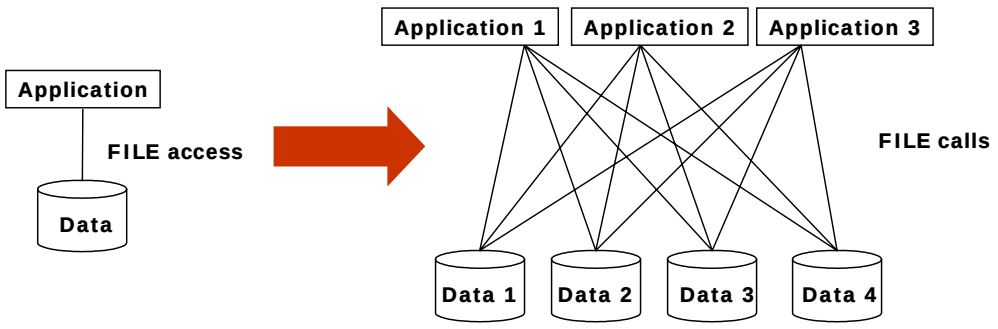
-e.g. computing a join from two relations in two different relational databases.

-Computing aggregations from transactions from many different local databases in a data warehouse.

A further advantage is that integrated DBMS support the application developer by hiding different data models, access methods, schemas etc. which he would have to take care otherwise within his application code. In other words integrated DBMS provide an intermediate layer that hides heterogeneity, distribution and autonomy.

Introduction to Data Management

- Applications use persistent data
- Problems
  - Reuse
  - Dependencies
  - Sharing



In the most general sense an information systems runs applications accessing (a large amount of) information that is persistently stored. The notion of ***persistence*** expresses that the lifetime of data is independent of the execution of a program. In that sense "Persistence bridges temporal distance among applications", similarly as distribution bridges spatial distance.

Typical examples of applications with persistent data are:

Product catalog - order application

Bank account data - bank transaction application

Gene databank - protein search application

Large databases are normally used by many different applications and the same applications are using many different databases. This leads to a number of problems

•All applications need to know how to manage the access to a large database. There is little reuse of the functions needed to efficiently access large datasets.

•Inconsistencies and redundancies occur in the data when different applications access the same databases and each application has its own assumptions on which constraints or dependencies hold.

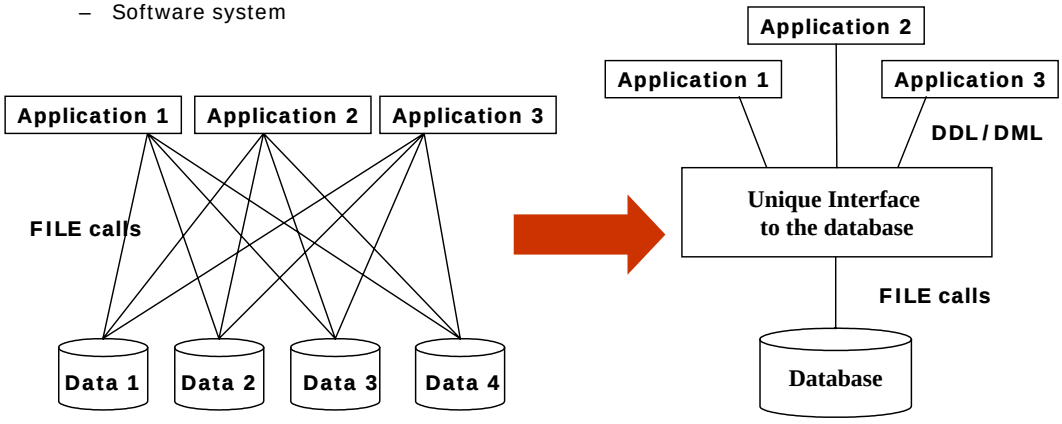
•Concurrent access to the same data leads to conflicts if multiple applications are using the same database at the same time.

•Dependencies exist: among databases (databases assume the existence of certain data in other databases), databases and applications (applications assume certain properties to hold in the database), among applications (applications assume that other applications are performing certain operations)

•Changes in the data and applications become difficult to manage. Since the "knowledge" on dependencies is distributed in many places and since the same functionality is implemented in many places each change becomes a complex task to perform.

Database Management Systems

- Provide a unique point of access to the data
  - DDL/DML
  - Transactions
  - Software system



A DBMS (database management systems) factors out the standard functions to access large databases. In fact we could see a DBMS as a way of integrating the access to otherwise unrelated data files.

A DBMS supports an abstract data model that consists of

DDL = language to describe the structure of data

DML = language to access the data (query, update)

A DBMS coordinates the access by multiple users (operational model) by supporting the concept of transactions

Transaction = operation that moves the database from one consistent state to another

The notion of database and database management system need to be differentiated

The **DBMS** is a software system that supports DDL/DML/Transactions for accessing a **database**

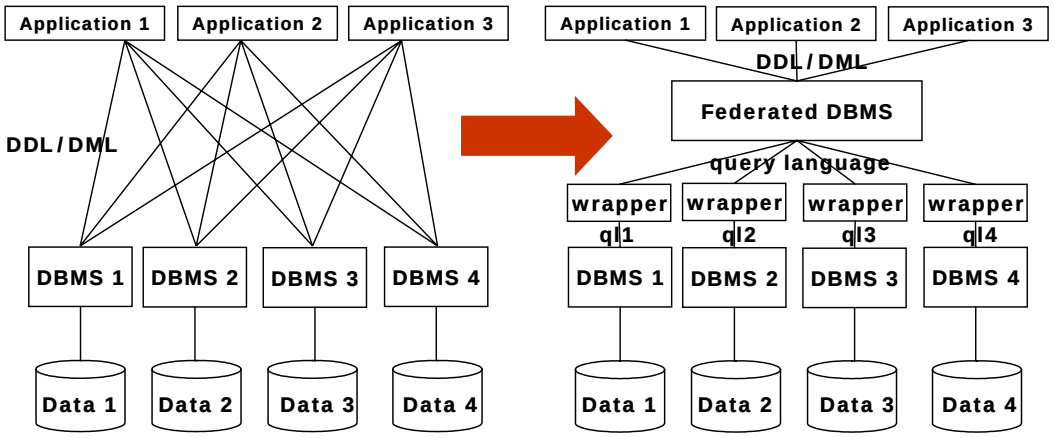
The main differences among different DBMS are

Data and operational model

Type of transaction support

Performance

Integrating Databases



 5

As soon as multiple database systems come into existence it is likely that a situation occurs where multiple applications would like to use data from multiple of the databases managed by those DBMS. This means that queries should be be enabled spanning multiple databases, updates should be directed to those databases where the data is originating from and integrity constraints spanning multiple databases should be supported.

Example: Integration of publication data. An integrated database could draw from many different resources

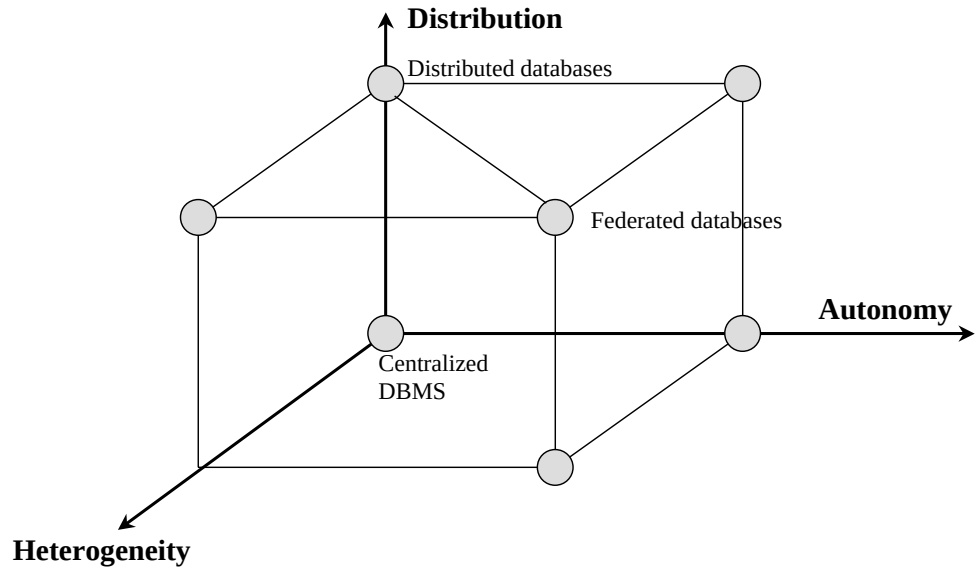
- Web resources: e.g. DB Server in Trier, CiteSeer citation index
- Local bibliography: e.g. MS Access application for lab library
- University library server: e.g. Oracle database application
- Private bibliography: e.g. BibTex file, MS Word document

The difficulty of this task originates in problems like

- Different data models at the participating DBMS
- Different data schemas and representation at the participating DBMS
- Different processing capabilities at the participating DBMS
- Distribution and communication

This means we encounter the typical problems of distribution, heterogeneity and autonomy when trying to integrate the access to multiple database that are a priori unrelated. Therefore a system is required that overcomes these problems, which is called a federated database management system. As we have learned in the introduction also an embedding of the participating database is required, which is provided by so-called wrappers. Wrappers overcome, for example differences in the data models (relational, OO, file, XML, etc.).

Database Integration



In order to position federated database systems let us look at our problem dimensions. Federated databases have to deal with all three problem dimensions. There exist however situations where autonomy and heterogeneity are less important. With distributed databases we have no heterogeneity and autonomy at all, as they deal only with the problem of distributing on logical database over the network in order to improve performance.

Data Integration Checklist

- Task: integrate distributed, heterogeneous (and probably autonomous) data sources
- Abstract Model: *which model to use ? Relational, object-oriented, XML ?*
- Embedding
- Architectures and Systems
- Methodology

For federated database management systems the situation is fundamentally different. Since the databases participating in a federation have been developed independently (design autonomy) we encounter normally a high degree of heterogeneity. In addition the participating DBMS continue to serve their original purposes even when participating in a database federation and therefore they have a high degree of autonomy.

Looking at the four key questions in information systems integration we want to look now at the first one, the question which is the abstract model that is used for the integrated system, in other words the data model that is supported by the federated DBMS and under which the integrated databases can be accesses. We can consider all of the existing kinds of data models that have been proposed, and in fact each of those has also been used as global model for federated DBMS:

Relational Model

Most frequently used data model for centralized and distributed DBMS

Difficult to represent data from more complex models (documents, OO)

Thus best suited for integrating relational data sources

Difficult to represent constraints occuring in integration (e.g. generalization/specialization)

Object-oriented Data Model

Expressive

Has been proven as successful approach and used in research and some systems

Did not prevail in practice because of a lack of commercial success of OODBMS

XML, XML Schema

Is becoming a ubiquitously used data model

XML Schema contains the main elements of an object-oriented data model

Can be used both for representation of data and documents

From these characterizations we can see that most likely the XML based models will become the prevailing global data model that is used for integration of databases. Therefore we will give an in-depth coverage of this model. But before we will repeat shortly the main aspects of the other data models.

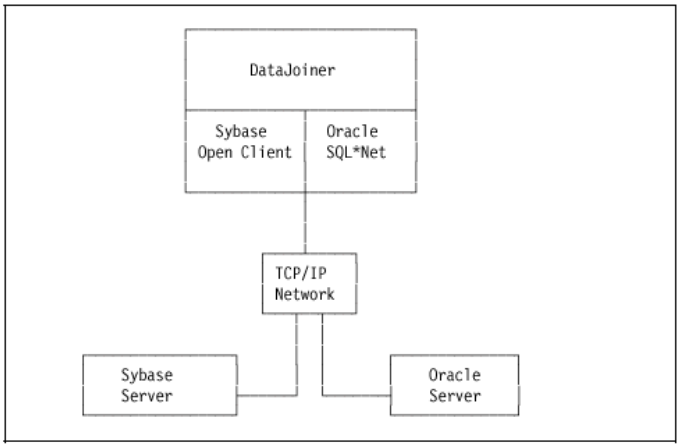
1. Important Types of Integrated Database Systems

- Multidatabases
  - Integrated access to different relational DBMS
- Federated Databases
  - Integrated access to different DBMS
- Mediator Systems
  - Integrated access to different data sources (on the web)
- Data Warehouses
  - Materialized integrated data sources

The different types of integrated database systems follow a natural evolution: originally the main problem was to integrate data from different relational databases and to create an intermediate layer that would let them appear like a single relational database. This led to the so-called multidatabase architectures. However, much of the data is not stored in relational databases, but either in legacy databases (hierarchical or network databases), non-standard databases (object-oriented databases) or in document-oriented formats (files, EDI data, Web data). Thus federated databases go one step further by integrating data from databases that use different data models. Furthermore, they typically also support the semantic integration of the database schemas. As a result they provide a database system layer that would provide *full DBMS functionality* (OLTP processing, online transaction processing) when accessing data from different database systems. On the Web data sources typically do not support full DBMS functionality. Therefore another form of federated DBMS evolved which focusses on *read-only* access to *online* resources on the Web, so-called mediator systems. Finally for the *online* analysis of large datasets from multiple data sources which are OLTP databases, which also requires only *read-only* access, data warehouses are another approach to allow the efficient processing of so-called OLAP queries (OLAP = online analytical processing).

Multidatabases

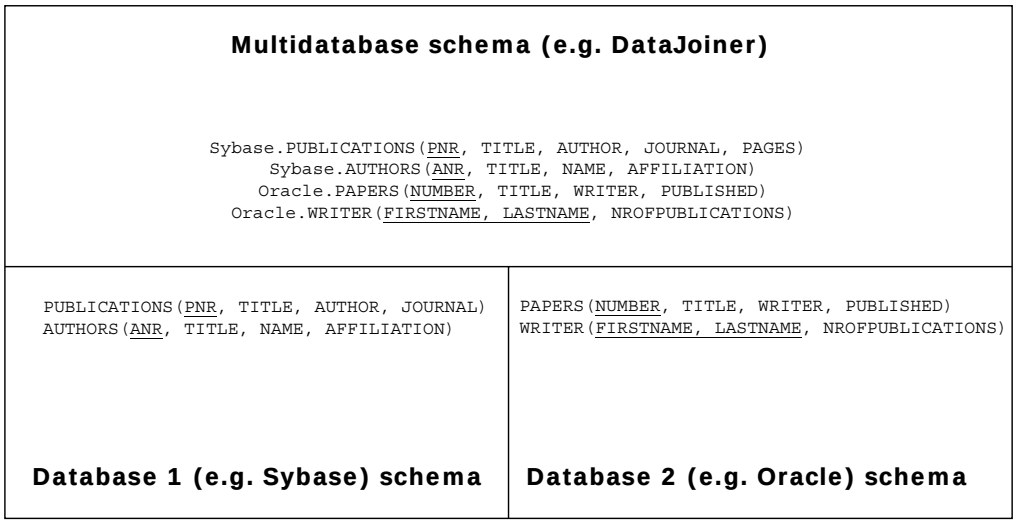
- Enable transparent access to multiple (relational) databases (relational database middleware)
  - Hide distribution, different database language variants
  - Process queries and updates against multiple databases (2-phase commit)
  - Do not hide the different database schemas



Source: IBM

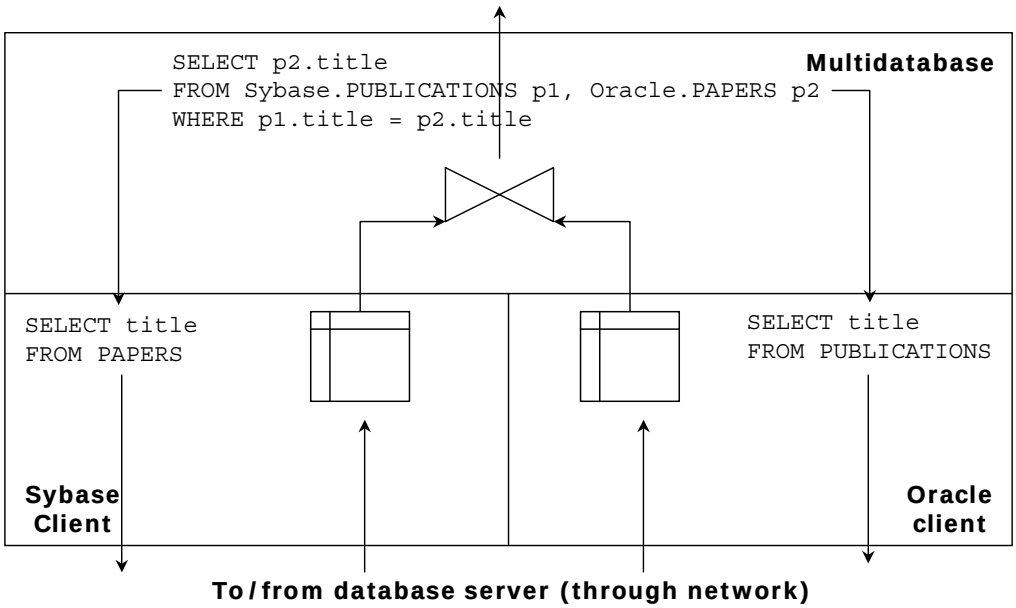
The task of multidatabases is to enable transparent access to multiple relational databases. In the illustration we see the basic architecture of a multidatabase (the example is taken from DataJoiner of IBM). Data Joiner uses the proprietary database clients (respectively their APIs) to access the different relational databases. It can overcome the differences among the different SQL dialects, and can decompose queries which are posed against the component databases (see example on next slides) as well as process updates (using 2-phase commit, we will introduce this concept later in the lecture). An important aspect is that multidatabases do not hide the different schemas from the component databases, the multidatabase schema is just the union of the schemas from the component databases (after renaming, in order to avoid name conflicts).

Multidatabase Schemas



This figure illustrates the usage of schemas in multidatabases. The components have each an own database schema, and the database can be accessed using this schema through the corresponding database client (e.g. one can submit queries against this schema to the client). The multidatabase schema is the schema that a user/developer who is using the multidatabase system sees. It is nothing else than the union of the two component schemas after renaming, by adding a prefix to the relational table names.

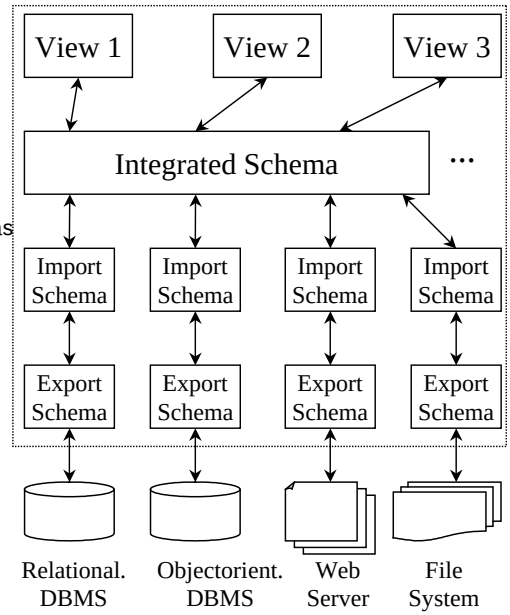
Multidatabase DML Processing



When a DML statement, such as a query, is submitted, the multiDBMS has to decompose it into queries against the two component databases. This is a non-trivial task, because it has to determine first which tables are from which database, which predicates apply to only a single database and which predicates apply to tuples from both databases. The latter ones can only be evaluated within the multidatabase, whereas the former ones can be evaluated within the component databases. In the example this is illustrated: the queries against the component databases do not contain the join predicate. The join itself needs to be evaluated at the multidatabase level. One of the main challenges in developing multidatabases is thus to find good strategies of how to decompose and evaluate queries against multiple databases in an efficient manner.

Federated Databases Schema Architecture

- Export Schema
  - Unifies data model
  - Defines access functions
- Import Schema
  - view on export schema
- Integrated schema
  - Homogenizes and unions import schemas
- Views
  - as in centralized DBMS



Federated databases go one step further than multidatabases by transforming the databases to be integrated both with respect to the data model as well as with respect to the database schema. Thus data stored according to different data models can be integrated and the database schema of the integrated database can be very different from the schemas of the component databases (which we will call in the following also component schemas). In order to structure the process of mapping the data from the component databases to the integrated database, a 5-layer abstract schema architecture has been introduced. It separates the different concerns in the mapping process.

In a first step differences in the data models (relational, XML etc.) are overcome by providing export schemas, which are expressed in the same (canonical) data model. Each component database can decide autonomously which data and which access to the data to provide in the export schema. These export schemas can be used by different integrated databases.

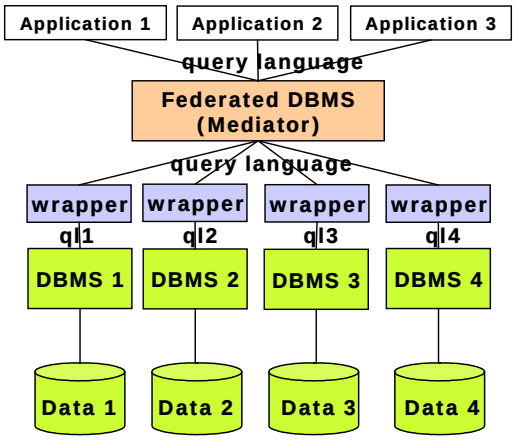
In a second step the integrated databases decide which data to use from the component databases by specifying an import schema. The import schema can of course only use what is provided by the export schema, thus it is a view on the export schema.

In a third step the integrated DBMS maps the data obtained from different databases as defined in the import schemas into one common view, the integrated schema. This can be a complex mapping, both at the schema level and the data level (we will extensively discuss this step later in the section on schema integration). The integrated schema is what the integrated database provides to the applications as logical database schema.

As in standard databases from the integrated schema application-specific views can be derived.

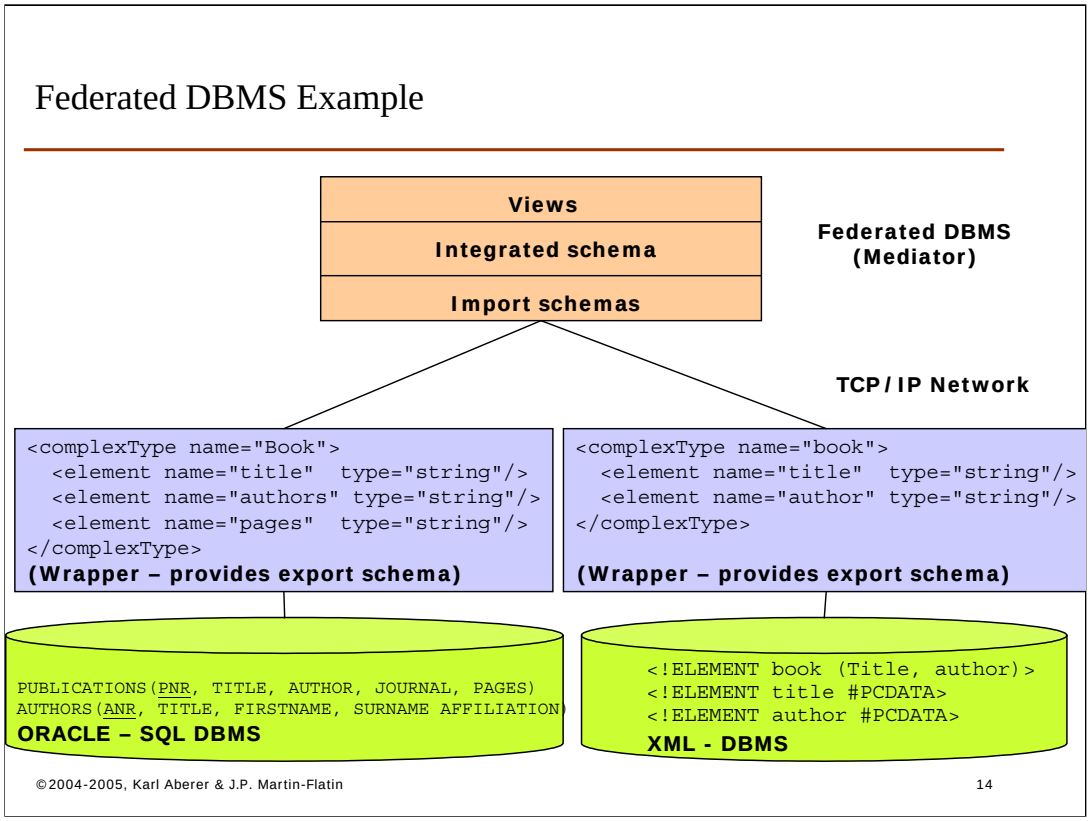
In practice the systems not always follow strictly this architecture and different layers are combined (e.g. often there is no clear distinction among the export and import schemas). Still the architecture is a very useful tool in order to distinguish the different aspects that need to be addressed in database integration.

Generic Federated DBMS Architecture



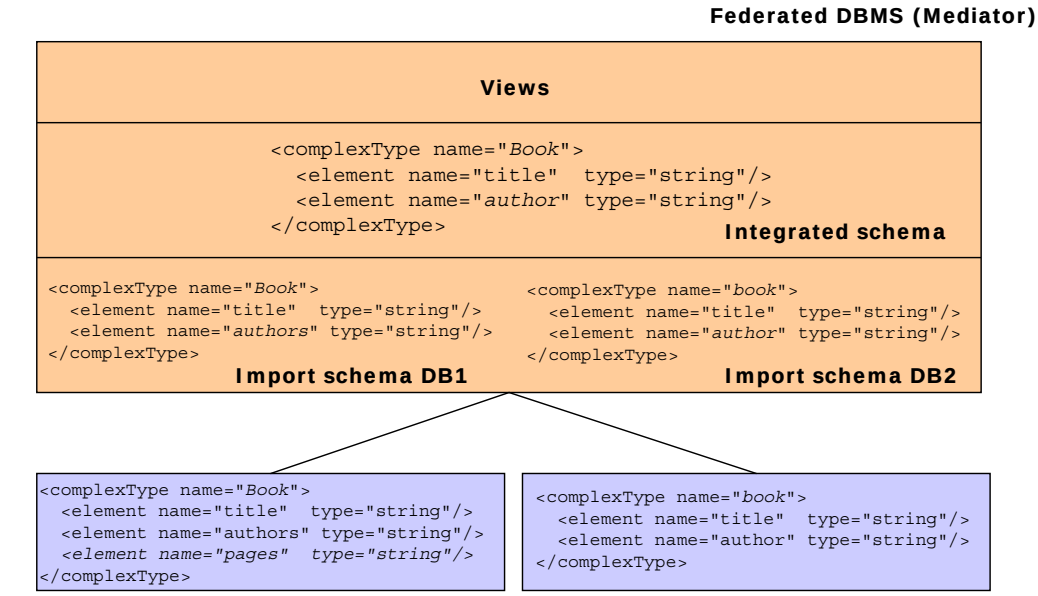
The standard approach to implementing the federated schema architecture is given by the generic federated DBMS architecture. The component DBMS (DB1-4) are accessed by so-called wrappers. The task of the wrappers is to map the component data models into the canonical data model and thus to support the export schemas for each of the component DBMS. The federated DBMS is the system that implements the integrated schema, based on the import schemas and provides the different views to the applications.

The integration function that is performed by the federated DBMS is also termed frequently as taking the role of a "mediator".

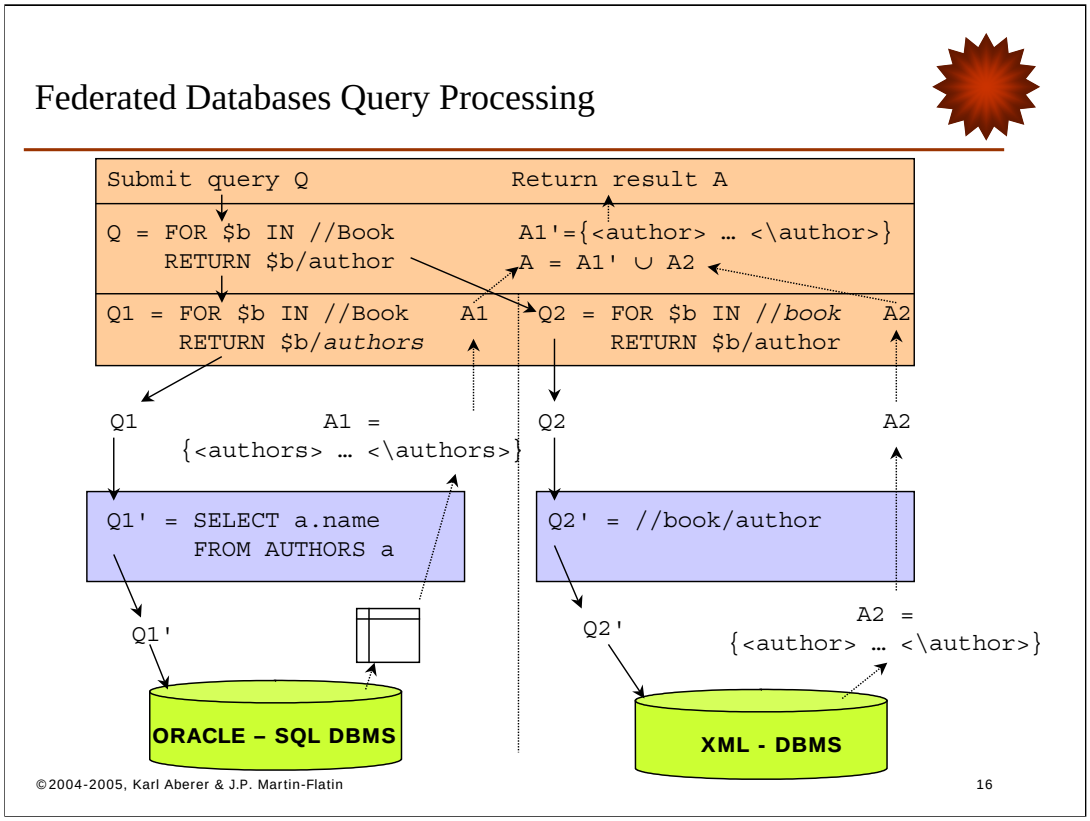


We illustrate now the relationship between the federated schema architecture and system architecture by an example. Here the component DBMS use different data models (relational and XML DTD) and the canonical data model is again different, i.e. XML Schema. Thus the wrappers provide the necessary functionality to access data in the component databases according to the canonical data model. For the relational database, for example, a complex type "Book" is defined in the export schema that contains data from both tables PUBLICATIONS and AUTHORS (assuming the element "authors" contains the author names). For populating the export schema (or for transforming the data) the wrapper would need to compute a join using the underlying database system. For the XML database the export schema is an XML Schema that exactly corresponds to the XML DTD. Thus also the transformation of the XML data is simple, because it needs not to be transformed at all !

Federated DBMS Example



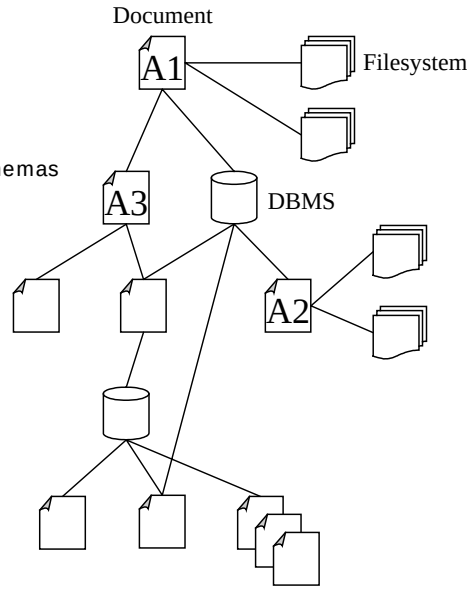
Next we zoom into the internal workings of the federated DBMS. First the federated database determines the import schemas. As it is not using the element "pages" from DB1 the import schema drops this element. Otherwise the schemas remain unchanged. The two import schemas, though similar, still contain differences in the names. Both "Book"-"book" and "authors"-"author" are incompatible names. Therefore in order to arrive at an integrated schema the federated DBMS has to map the two import schemas into one integrated schema as shown above (of course in general the differences among the schemas will be much larger – and how to produce mappings to integrated schemas in the general case will be discussed later). What is not shown in the illustration is the corresponding mapping at the data level. Since now books from two databases are mapped into one integrated database, it can occur that the same book is contained in both component databases. Then the federated DBMS has to recognize such a situation and take the corresponding action, e.g. by instantiating such a book only once in the integrated database.



Once the integration is established at the schema level, the federated DBMS needs to support standard operations on the integrated databas, in particular DML operations. Since multiple schemas and data models are involved, this task is considerably more complex than in the case of multidatabases. This is illustrated in this example: assume a query for all authors of books is now submitted to the integrated database in XML Query. First the federated DBMS has to determine which databases can contribute to the answer and generate from the query Q that is posed against the integrated schema, queries that can be posed against the import schemas (Q1 and Q2). In that step it has also to invert the mappings that have been performed during integration (e.g. mapping back to the names "book" and "authors"). Then the query is forwarded to the wrapper without change since the import schema is just a projection on the export schema. The wrapper has now the task to convert the query from a query in the canonical query language and data model to a query in the model of the component database. For the relational database this requires the transformation to an SQL query, whereas for the XML database the query is converted to an XPath query. Then the wrappers submit those queries to their respective component database systems and receive results in their respective data models. For the relational database this means that now the wrapper has to convert the relational result into an XML representation. For the XML database no change is required since the data representation (namely XML documents) is the same independently of whether XML DTDs or XML Schemas are used as XML type definitions. Once the federated DBMS receives the results it has to map the result data from the import schema to the representaiton of the integrated schema. For our example this requires a renaming of the element name "authors" to "author" for result A1. (note that the renaming of "book" is not required as this name does not occur in the result). Then the federated DBMS computes the result of the query by performing a union (and thus eliminating duplicates). Note that for more complex mappings from the import schemas to the integrated schema this step can be considerably more complex). Then the final result can be returned to the application.

The Web as Loosely Coupled Federated Database

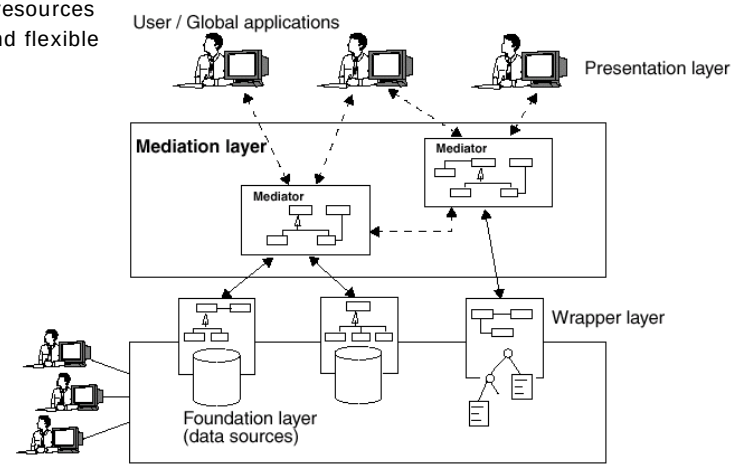
- Many different, widely distributed information systems
- Heterogeneity
  - Structural homogeneous: HTML and URL
  - Semantically heterogeneous: no explicit schemas
- Autonomy
  - Runtime autonomy: pages change on average every 4 weeks, dangling links
- Distribution
  - Replication (proxies) and caching frequently used



Whereas traditionally federated DBMS assumed a fairly controlled and static environment for database integration (as we would encounter it for example in the merger of two companies that need to integrate their DBMS), the problem of data integration underwent an evolution with the arrival of the WWW. We may consider also the Web as a collection of data sources for which integrated data access is desireable. Even without any additional technology we can view the Web as a very loosely coupled federated database. The canonical data model would be HTML, the web sites would be very autonomous and the distribution is wide.

Mediator Systems

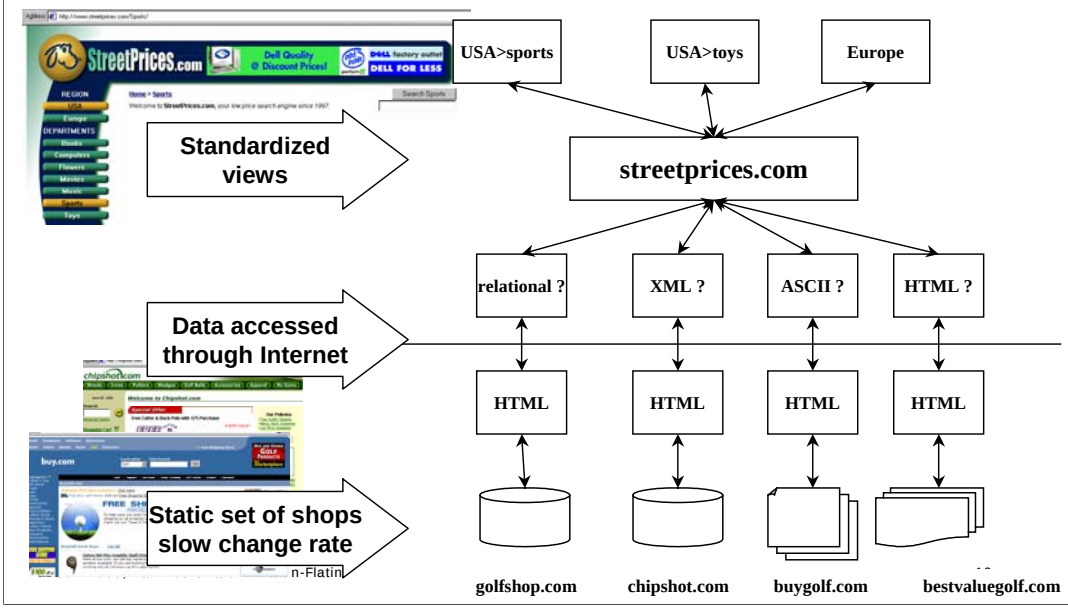
- Main differences to conventional federated databases
  - Read-only access
  - Non-database resources
  - Light-weight and flexible



Source: Kutsche

However, for more coordinated forms of access to Web data one might want to take an approach that follows the federated DBMS paradigm, with some adaptations. For example, by providing wrappers, we could map data from the HTML model to the XML model. This mapping is of course in general more difficult to achieve, since data represented in HTML does not have a schema (non-database resources), and thus the "semantics" has to be extracted from the content of the data. Also mediators in general would not be required to be able to deal with updates, since foreign Web sites cannot be changed anyway. Thus a federated DBMS on the Web would focus on read-only access. A practical requirement is that the infrastructure for data integration on the web must be more light-weight and adaptable. Integrated database systems that carry these characteristics are nowadays called mediator systems.

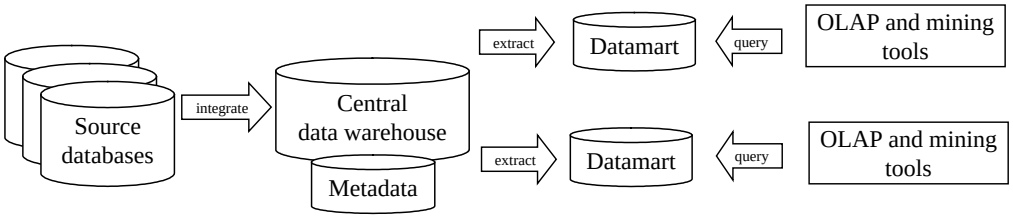
Example: Integrated Product Information



An application where the mediator approach is frequently being used on the WWW is product and/or price comparison. In this application one extracts from online shops the essential product information (typically by analysing the HTML code using hand-written wrappers) and integrates this information for comparison purposes.

Data Warehouses

- Are a specialized form of federated database system, where the integrated database is materialized
  - perform efficient data analysis
  - OLAP (on-line analytical processing)
  - Data mining
- Uses typically limited forms of database schemas (e.g. relational star schemas)
  - Supports multidimensional data model
- Main problems
  - Updates and data consistency
  - Wrapper development
  - Efficient processing of OLAP and data mining queries



Another variation of the federated databases approach are data warehouses. These are used to globally analyse large datasets that are composed from the data originating in many OLTP databases (e.g. the transaction data from all stores of a retailer all over the country) in order to derive strategic business decisions. Since updates play no role here, and efficient processing of large datasets is the critical issue, the data is not directly accessed in the component databases, but materialized in the integrated database, the central data warehouse. From this warehouse typically application-specific views are derived by materializing them in so-called datamarts, to which then the different data analyis tools are applied. The canonical data model for data warehouses is typically a specialized form of the relational data model, the so-called multidimensional data model, where all data is kept in one central table with different attributes that represent the different dimensions. It differs insofar from the relational model as it provides much more powerful operations on the multidimensional data for aggregation (similarly as in an Excel spreadsheet)

Important problems in data warehouses include the development of wrappers, which are not only used to map from different data models and schemas into the datawarehouse schema, but also to perform so-called data-cleansing, i.e. remove errors and noise from the original data.

Aspects to Consider for Integration

- General issues
  - Bottom-up vs. top-down engineering
    - from existing schema to integrated and vice versa
  - Virtual vs. materialized integration
  - Read-only vs. read-write access
  - Transparency
    - language, schema, location
- Data model related issues
  - Type of sources
    - structured, semi-structured, unstructured
  - Data model of integrated system
    - canonical data model
  - Tight vs. loose integration
    - Use of a global schema
  - Support for semantic integration
    - collection, fusion, abstraction
  - Query model
    - Structured queries, retrieval queries, data mining

Top-down approaches start by first specifying the global information need (in form of a global schema) and then engineering the access to relevant data sources. This approach is suitable when the integrated database serves a specific global information need (e.g. price comparison). Typically information sources can be easily changed and updates are performed at the local databases.

Bottom-up approaches start from a given set of databases with the goal to provide a single point of access to them. Thus the schema is constructed from existing ones, updates occur typically at the global level, and the question of providing a complete global schema, that covers all the concepts modelled in the local schemas becomes an issue. Generally, bottom up approaches lead to more tightly integrated systems and are harder to develop and maintain.

The tradeoffs between virtual and materialized integration have already been discussed in the context of data warehouses. Materialization has the following advantages: High performance for queries against the materialized data set and control over materialized data. It has also disadvantages: Keeping the data up-to-date requires possibly complex update procedures, and the federated DBMS has to provide a possibly large amount of storage space.

Write access is often disregarded in database integration for practical reasons, like, many interfaces, e.g. on the WWW, do not allow a write, writing through integrated views raises all problems of updating data through views, as mentioned earlier, writing through an integrated schema e.g. raises the question of which source should be used if a class is present in more than one data source and global transactions require complex protocols. ./.

Properties of Integrated Information Systems (typical)

	Multidatabases	Federated databases	Mediator Systems	Data Warehouses
Bottom-up vs. top-down	none	bottom-up	top-down	bottom-up
Virtual vs. materialized	virtual	virtual	virtual	materialized
Read-only vs. read-write	read-write	read-write	read-only	read-only
Type of sources	structured	structured	all	all
Canonical data model	relational	relational, OO, XMLSchema	semi-structured, XML	multi-dimensional
Global schema	no	yes	yes	yes
Support for semantic integration	collection	collection, fusion, (abstraction)	collection, fusion, abstraction	collection, fusion, aggregation
Transparency	(location), language	location, schema, language	location, schema, language	location, schema, language
Query model	SQL	SQL/XQL	Any	OLAP

./. Different data integration approaches provide different forms of transparency (i.e. location transparency to overcome distribution, language transparency to overcome differences in data models, schema transparency to overcome differences in schemas)

Obvious criteria to distinguish different database integration approaches are the data models supported both for component databases and at the integrated level and the presence of a global schema.

Different approaches can be distinguished also according to the support of semantic integration that is given:

-in the simplest case only collection both at schema and data level is supported (i.e. as in multidatabases)

-At the next level of complexity the capability of fusion of data objects is supported: objects from different sources can be identified as being equal and are mapped into a single object in the federated database

-The most complex form of integration is when the abstract model of the objects is manipulated (what operations are possible will be discussed in the section on schema integration), i.e. new abstractions are derived in the integrated schema. A special form of abstraction is aggregation, where multiple objects are aggregated into single objects which summarize their properties.

Finally not only the structural model is essential, but also the possible access to the data by means of a query language that is supported.

The table on this slide summarizes the different criteria for the approaches to database integration that we have introduced.

Data Integration Checklist

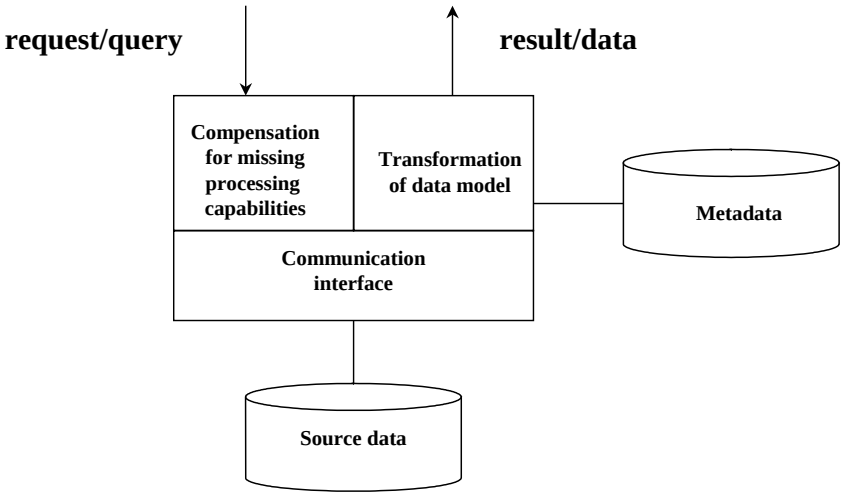
- Task: integrate distributed, heterogeneous (and probably autonomous) data sources

- Abstract Model ✓
  - Relational, OO, XML, XMLSchema
- Embedding (✓)
  - XSLT, XQuery
- Architectures and Systems ✓
  - Multidatabases, federated databases, mediator/wrapper, data warehouse

- Methodology

2. Wrappers: Transparent Access to Data Sources

- Wrappers are software to overcome syntactic heterogeneity (communication, data model, functionality)



We give now a more detailed overview of wrapper technology. As a consequence of the variety of possible mappings they have to support wrappers vary substantially in their approaches and architectures. Nevertheless we can identify some fundamental components that every wrapper contains.

-it has to be able to receive requests and return results acccording to the canonical data model through it's API

-It has to transform requests and while doing that to compensate for missing functionality that is expected by the API it supports and that is not supported by the database that manages the data source

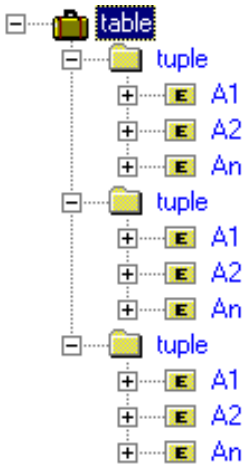
-It has to have a communciation interface with which it contacts the source database

-It has to transform data from the data source to the canonical data model

-It maintains metadata for the data models and schemas involved and the mappings from the source to the canonical data model.

Mappings between Different Data Models

- Data Model consists of
  - Data types
  - Integrity constraints
  - Operations (query language)
- Criteria for mappings between data models
  - Completeness
  - Efficiency
- Example
  - The mapping of a relational table instance R(A1, …, An) into an XML structure is straightforward
  - The problems are
    - Which constraints can be maintained ?
    - What query processing capabilities are available ?
    - e.g. joins are missing
  - XML Schema + XML Query solve many of these problems !



Mappings between data models concern not only the mapping of structural data, but also the mapping of the associated constraints on the data and the mapping of the operations on the data. Principally any data structure can be mapped into any other structure in some way (since there exists always a mapping between countable sets). The question is to which extent the constraints that are imposed by one data model are maintained by the mapping (completeness), to which extent the operations can be mapped and how efficiently the mapping is both in time (when performing operations) and space (for storing the mapped data). For the example of mapping a relational table into XML it is obvious that every table can be respresented by some XML document easily, as shown in the figure. Problems occur when operations should be performed on the mapped data. For example, since XML has no concept of keys (neither primary or foreign) updates on the XML data can violate key constraints that have been specified in the relational model. Also queries against a relational database cannot always be mapped to corresponding queries in XPath, if XPath would be the query language used.

This explains why XML schema and XML Query have been defined in order to overcome these deficiencies of the original XML model.

Categories of Wrappers

- There exists no standard approach to build wrappers
- Functionality
  - One-way: only transformation of data (e.g. for data warehouses)
  - Two-way: transformation of requests and data
- Development
  - Hard-wired wrappers, for specific data sources
  - Semi-automatic generation: wrapper development tools
  - Automatically generated wrappers
- Availability
  - Standalone programs (data conversion)
  - Components of a federated or multi-DBMS
  - Database interfaces to foreign data

Since there exist a multitude of ways of how wrappers are built, we want to classify them first according to some criteria. A major difference is whether wrappers are used only to transform data into one direction (which is always sufficient when the data is materialized in the federated database and no updates are made) or both queries and updates need to be processed by the wrapper,which makes them considerable more complex.

The second class of criteria applies to the development of wrappers. Today, typically many wrappers are still custom-developed software. There exist for specific classes for applications, in particular for data warehousing, tools that support developers in the definition of the data mapping, typically using visual tools, and generated from the mapping definition the necessary transformation code. Automatically generated wrappers are largely a research area or only applicable in very specialized settings.

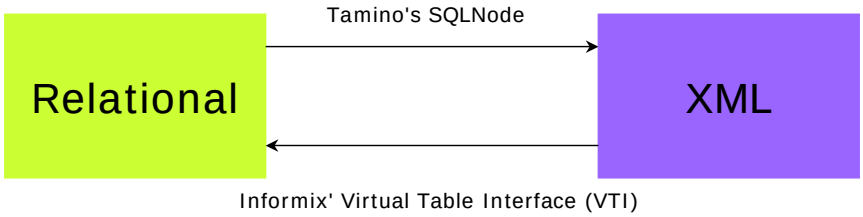
Also with respect to deployment there exist different possibilities: transformation or data conversion tools are often deployed as stand-alone programs, as they have little interaction with the operation of the federated DBMS. Data conversion tools are specific software that support conversions among different formats, perform standard conversions semi-automatically, allow definition of complex conversion logic by the developer, and maintain a repository of conversion methods. Examples of such tools are the Microsoft data import tools and XMLJunction. Also XSLT can be considered as a possible appraoch to data conversion in the XML world, e.g. for conversion of XML data into relational data.

Multidatabase products such as IBM DataJoiner, Miracle, or EDS allow to access multiple relational databases (from different vendors) and support as such components to access relational database through the corresponding database clients, as discussed earlier. We can consider these components as wrappers that overcome syntactic heterogeneity among relational DBs, and thus support properties such as location transparency, SQL dialect transparency, error code transparency, or data type transparency. They are examples of two-way wrappers.

Finally, many database products provide (tightly) integrated functionality for integrating access to foreign data sources, such that the database products becomes a federated database platform.

Database Interfaces to Foreign Data

- Some database products allow to define user-defined mappings to data from arbitrary sources
  - Data appears like local data
  - Mapping includes (some) compensation for missing functions at the source
- Examples
  - Informix' Virtual Table Interface (VTI)
    - Any data to relational data
  - Tamino's SQLNode
    - relational data to XML data

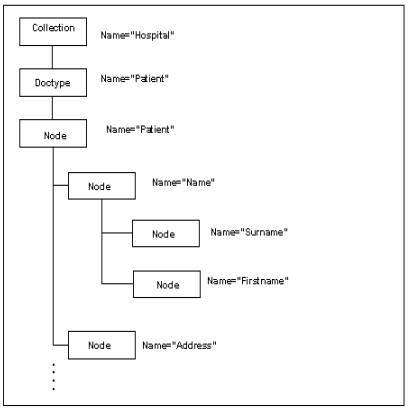


We give in the following two examples of how current database system products support the wrapping of foreign data sources into their own data model. The first example, Informix – in the meanwhile a product of IBM -, is an example of how data of any origin may be wrapped into a relational DBMS, thus making the relational DBMS a platform for managing heterogeneous databases. The second example, Tamino – a product of Software AG -, shows of how a native XML DBMS can be used in order to integrate relational data from SQL database sources. In fact, Tamino has been developed with the intention of providing a XML-based federated DBMS layer.

Example: Tamino

- Native XML storage system
- Tamino allows to provide an XML view on a relation
  - Definition of a DTD
  - Transformation to Tamino's schema language
  - Setting of attribute values that specify of how the data is accessed in the RDBMS

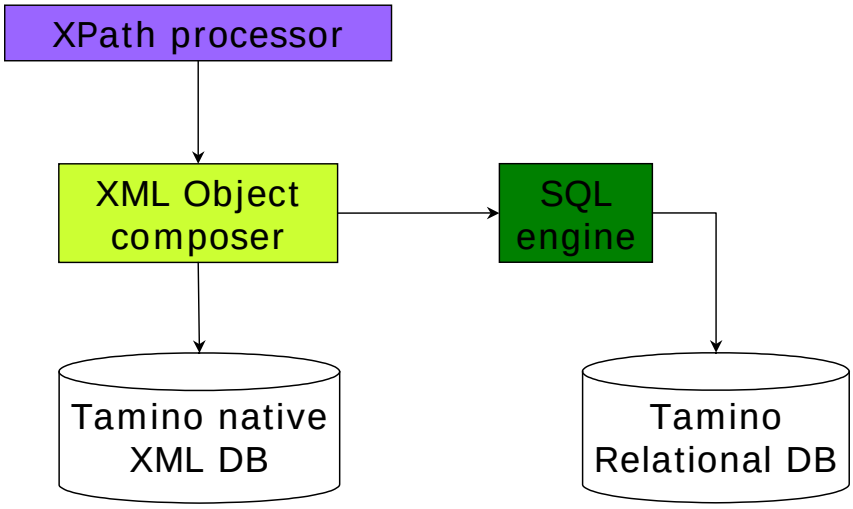
```
<!ELEMENT patient      (name, address, sex, born)>
<!ELEMENT name         (surname, firstname)>
<!ELEMENT surname      #PCDATA>
<!ELEMENT firstname    #PCDATA>
<!ELEMENT address      (street, housenumber, city,
                        zip, phone)>
<!ELEMENT street       #PCDATA>
<!ELEMENT housenumber  #PCDATA>
<!ELEMENT city         #PCDATA>
<!ELEMENT zip          #PCDATA>
<!ELEMENT phone        #PCDATA>
<!ELEMENT sex          #PCDATA>
<!ELEMENT born         #PCDATA>
```



Tamino, a recent product by Software AG, supports native storage of XML documents. "Native" means that XML documents are not mapped to another storage systems (e.g. a relational database), but a specialized storage system for XML documents is used. In addition, a native storage system for XML allows to pose queries in a XML query language. In the case of Tamino this language is Xpath (support for XML Query is planned).

The interesting property of Tamino is, that it allows to use the XML engine not only to store XML documents, but also to integrate data from other datasources (like relational) under a common XML view. Thus it is an XML-based federated DBMS (i.e. using XML as canonical data model) . In order to support this feature Tamino needs a mechanism in order to wrap other data sources in an XML view. We describe this mechanism now more closely: In order to store XML data in Tamino first a DTD needs to be provided (which is the database schema). We give also a schematic representation in tree format of the DTD on the right hand side. Afterwards this DTD is converted to a proprietary schema language, that bears some similarity with the XML schema language. Based on this schema representation the access to SQL data sources is specified. We look at this step in more detail on the following slides.

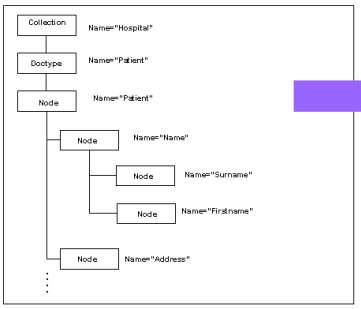
Tamino Integration Architecture

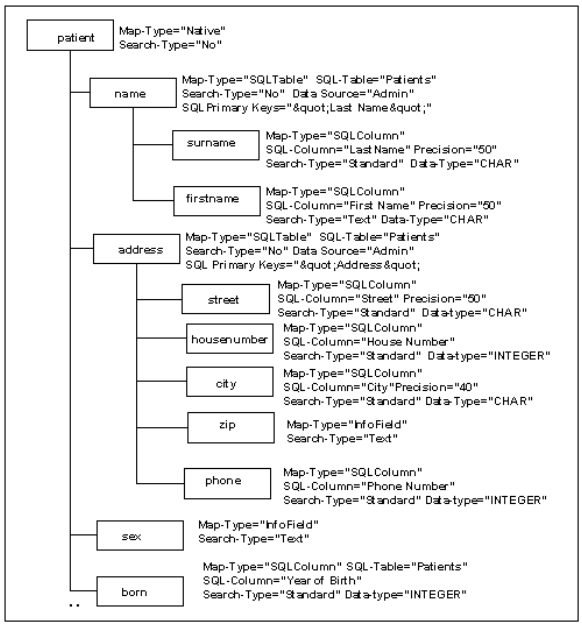


Integrating Relational Data Into Tamino

Relational Table:

Patients(LastName,FirstName,
Street, HouseNumber,
City, PhoneNumber,
YearOfBirth)





Source: SAG

Now let us assume that the data of the patient documents originates from an SQL table with the schema shown above. As a result certain parts of the document are no longer stored as an XML document in Tamino, but rather dynamically derived from the SQL table. In order to enable the mapping the DBMS requires certain information:

- The database from which the table originates
- The table from which the data originates
- The mapping of table columns to XML elements
- The data types of the mapped columns

The corresponding information is annotated at the right side of the elements in the form of (XML) attributes. We can distinguish two major cases:

- The mapping of a table onto an element (name, address): Here subsets of the attributes of a relation are associated with an element. The content of these elements then is derived from relation attributes (columns). In order to specify this mapping the database name ("Data Sources") and the table name ("SQL-Table") need to be known.
- The mapping of a relation attribute (column) onto an element (lastname, firstname, street etc.): These are subelements of an element that has been mapped from the corresponding relation. In order to specify this mapping the name of the column needs to be known (the table is already implied by the super-element). Additionally some information on the data type of the column is supplied in order to correctly interpret the contents and search predicates.

There exist a number of constraints that have to be observed, like:

- for a node SQLTable the objtype PCDATA/CDATA is not possible
- Similarly for SQL Coumn SEQ/CHOICE are not possible (for obvious reasons)
- The multiplicity for SQL Column can only be 1 or 0 (if nothing is mentioned the default is 1, as in the example)
- Infofields are used for XML elements that are not derived from the SQL table but occur within the scope of an SQL Table derived element

Integrating Relational Data in Tamino Schema Language

```
<?xml version="1.0" ?>
   <!-- <!DOCTYPE Collection SYSTEM "inorep.dtd">   -->
   <ino:collection
        xmlns:ino="http://namespaces.softwareag.com/tamino/"
        ino:name="hospital" ino:key="ID001">
   <ino:doctype
        ino:name="patient" ino:key="p1"
        ino:options="READ INSERT UPDATE DELETE">
   <ino:node
        ino:name="patient" ino:key="p2"
        ino:obj-type="SEQ" ino:parent="p1"
        ino:search-type="no" ino:map-type="Native" />
   <ino:node
        ino:name="name" ino:key="p3"
        ino:obj-type="SEQ" ino:parent="p2"
        ino:search-type="no" ino:map-type="SQLTable"
        ino:datasource="Admin" ino:sqltable="Patients"
        ino:sqlprimarykeys=""Last Name"" />
   <ino:node
        ino:name="lastname" ino:key="p4"
        ino:obj-type="PCDATA" ino:parent="p3"
        ino:search-type="text" ino:map-type="SQLColumn"
        ino:sqlcolumn="Last Name" ino:data-type="CHAR" ino:precision="50" />
```


This example shows of how the declarations of the mapping from the SQL database into the XML format are included in a Tamino schema by using the attribute values SQLTable and SQLColumn (rather than "Native" as originally) and including the attributes for specifying the detailed mapping, that have been introduced on the previous slide.

This is a sample of Tamino's schema language, including the representation of the first three elements of the DTD, namely patient, name and lastname. The first element <ino:collection> declares a Tamino collection, which is a set of XML documents that the Tamino system manages. There also the "ino" namespace is introduced (short for Tam-ino). Every schema constituent obtains a name using "ino:name" and a unique identifier using "ino:key". The second element <ino:doctype> starts the declaration of a document type definition. The options declare that documents corresponding to this DTD may be accessed in any of the possible ways listed. The <ino:node> elements are used to declare elements of the document type. They correspond exactly to the elements of the DTD we have introduced before. For each element

•a name is declared using attribute ino:name, which matches the name in the DTD,

•a unique key is declared, which is used to express the parent-child relationships in the DTD,

•the type of the element is declared by using ino:objtype, where the possible types correspond to the possible types in a DTD,

•the parent node is declared using ino:node, and the reference is given by the value of an attribute ino:key

•the search-type is declared, specifying whether this field is indexed, there are two possiblities SQL compliant indexing (standard) and full text indexing (text)

•and the map-type is declared which is "Native" in case the element is stored in XML format within Tamino.

For practical purposes Tamino provides tools in order to automatically generate from an existing DTD such a Tamino schema representation. We will see in the following how be declaring other map-types data from external sources can be integrated into Tamino.

Data Integration Checklist

- Task: integrate distributed, heterogeneous (and probably autonomous) data sources

- Abstract Model ✓
  - Relational, OO, XML, XMLSchema
- Embedding ✓
  - XSLT, XQuery, various wrapping technologies
- Architectures and Systems ✓
  - Multidatabases, federated databases, mediator/wrapper, data warehouse

- Methodology

3. Schema Integration

- 3.1 Schema Integration Method
- 3.2 Identification of Correspondences
- 3.3 Resolution of Conflicts
- 3.4 Implementation of an Integrated View

In this part of the lecture we will address a methodological question: even when we have the complete infrastructure for integrating information systems at the data level in place and the canonical data model is chosen, there remains the important task to develop the necessary mappings for the data from the heterogeneous data sources into the integrated database. This is considered as being a very hard problem on which a substantial amount of research has been and is being devoted. While this problem originally was studied in the context of database integration, it is today drawing additional attention in other contexts, such as e-business (integration of message formats and electronic catalogs) or the Semantic Web (integration of knowledge bases and ontologies).

The creation of an integrated schema from heterogeneous schemata is required in order to make heterogeneity at the schema level transparent for applications. Schema integration is more complex than just producing the union of the component schemas (as it is done in multidatabases):

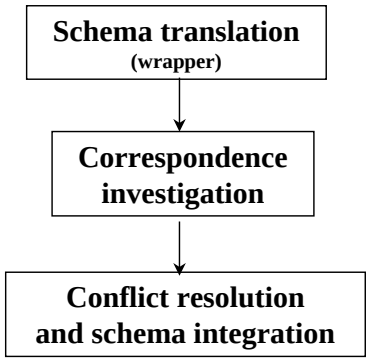
-Data from the different sources can be complementary or overlapping, therefore data objects from different sources may be corresponding to each other or data objects belonging to different classes in different databases need to collected into a common class.

-Another problem is that the same data can be represented in different ways. This requires mappings (both at the schema and data level) that can overcome the differences (schema integration)

-Once an integrated schema is provided the capability to process queries against the integrated schema is required. Therefore also mappings in order to decompose and translate from the global query to queries against the component databases need to be provided.

3.1 Schema Integration

- Creation of an integrated schema from heterogeneous schemata
- Integration Method



In the following we will focus on the first two issues, of how to create an integrated (global) schema. The fundamental approach consists of three steps:

- The schemas of the component databases need to be translated from the original data model into the canonical data model. This is done by a wrapper and we have seen in the previous section different possibilities of how this can be achieved. Therefore we have to treat this point no further.
- When all component schemas are available within the same data model the next step is to explore which concepts in the different original schemas correspond to each other, i.e. which are semantically related (describe the same, similar or otherwise related concepts of the "real world")
- After identifying corresponding concepts in general it will be the case that they are represented in different ways - one says they have conflicting representations. Therefore in a next step corresponding concepts that are structurally different but should be represented in the same way in the integrated schemas need to be mapped into a common representation. This step is called conflict resultion. The result of this step is an integrated schema.

3.2 Identification of Correspondences

- Determine which concepts in two schemas correspond to each other
  - no automatic solution
  - manual or semi-automatic approach
- Sources for missing information
  - source schema
  - source database
  - source application
  - database administrator, developer, user
- Semi-automatic approaches (source schema or database analysis)
  - semantic correspondences (e.g. related names)
  - structural correspondences (e.g. reachability by paths)
  - data analysis (e.g. distribution of values)

Since correspondences among schemas and databases are always dependent on the application context (essentially the human interpretation of the databases in terms of real-world concepts) no automatic solution to correspondence identification is possible. Besides of directly consulting the human users involved (system administrators, developers or end users) one can attempt to extract from the available sources of information (schema, database, applications) certain correspondences by semi-automatic methods. Examples of such analyses are mentioned:

-correspondences which can be detected at the language level (words). For that purpose the use of dictionaries or thesauri is a possibility.

-Correspondences which can be detected at the structural level (data structures). The underlying assumption is that if something is modelled by using the same or a similar data structure the semantics might be similar.

-Finally a similar argument can also be applied at the data level. If characteristics of the data (like data distribution) are similar this can be an indication that the data corresponds to the same semantic concepts.

Example: Integration of Two Relational Schemata

- SCHEMA 1

```
PUBLICATIONS(PNR, TITLE, AUTHOR, JOURNAL, PAGES)
AUTHORS(ANR, TITLE, NAME, AFFILIATION)
JOURNAL(JNR, NAME, PUBLISHER, ISSUE, MONTH, YEAR)
```

- SCHEMA 2

```
PAPERS(NUMBER, TITLE, WRITER, PUBLISHED)
WRITER(FIRSTNAME, LASTNAME, NROFPUBLICATIONS)
PUBLISHED(TYPE, PROCEEDINGS, PP, DATE)
```


In the following we will use a simple example of two relational databases (respectively schemas) in order to illustrate the process of schema integration. We choose the relational model for the purpose of compactness of presentation (relational schemas are syntactically more compact than XML schemas, for example). The analogous examples could of course be goiven also for other data models, like XML Schema.

The two example schemas use slightly different representations of publication data and our goal is to derive now one integrated schema for integrating data that originates from databases corresponding to these two database schemas.

Example: Detecting Correspondences

- Analysis of relation and attribute names (semantic correspondences)
  - C1: S1.PUBLICATIONS related to S2.PAPERS (relevant)
  - C2: S1.AUTHORS.TITLE related to S2.PAPERS.TITLE (irrelevant)
  - C3: S1.PUBLICATIONS.TITLE related to S2.PAPERS.TITLE (relevant)
- Analysis of structural correspondences
  - Shows that C3 is more relevant than C2
  - S1.PUBLICATIONS.TITLE related to S2.PAPERS.TITLE
    - same path leads to TITLE
  - S1.AUTHORS.TITLE not related to S2.PAPERS.TITLE
    - different path leads to TITLE
  - Shows a further relationship: C4: S1.JOURNALS.YEAR related to S2.PUBLISHED.YR since both share the data type DATE
- Analysis of data values
  - C5: S1.PUBLICATIONS.PAGES related to S2.PUBLISHED.PP
  - Both attributes contain always two substrings that are up to 3 digit numbers

We start by investigating the correspondences:

Semantic correspondences can be typically derived from an analysis of names (attribute and relation names here). For example we might use a thesaurus that provides homonyms and synonyms for words (Homonyms are words that denote different real-world concepts, e.g. paper could denote a published paper, but also waste paper, which in the context of this applications makes a difference. Synonyms are different words that denote the same real world concept). From the thesaurus we can take that "publications" and "papers" can be synonyms and therefore that the relations S1.PUBLICATIONS and S2.PAPERS might be corresponding (we prefix relations by their source schema name similarly as in the multidatabase approach to identify them uniquely). By consulting a human user we would confirm that this correspondence is in fact relevant (This appears to be trivial for such a small example, but for very large schemas a system that identifies such candidates for correspondences can be very helpful) A second correspondence that can be detected is between the attributes S1.AUTHORS.TITLE and S2.PAPERS.TITLE , which are actually the same. However, in that case we have a homonym, and a human user would decide that this correspondence is not relevant. A third correspondence at name level, between S1.PUBLICATIONS.TITLE and S2.PAPERS.TITLE is again relevant.

The fact that correspondence C2 is more relevant than correspondence C3 could also have been detected at the structural level: Since the path (here the relation) that leads to title in C3 is the same, whereas it is different in C2, it is more likely that C3 is a correspondence.

Another correspondence can be detected at the structural level: The path S1.JOURNALS.YEAR and S2.PUBLISHED.YR lead both to an attribute of type DATE and could therefore be related. This structural relationship could further be strengthened by a semantic one, namely that the terms JOURNALS and PUBLISHED are semantically close, as they relate to the same real world concept. Detecting this would require however a more sophisticated knowledge base than a thesaurus. These are called ontologies, which provide knowledge bases (concepts, but also rules) that model certain aspects of the real world.

Finally we can also analyse that data values that we find in the attributes. There we might detect that the attributes S1.PUBLICATIONS.PAGES and S2.PUBLISHED.PP always contain Integer values of roughly the same range (typically at most 3 digit numbers) and therefore the two attributes might be related (which again is a relevant correspondence)

3.3 Resolution of Conflicts

- Types of conflicts encountered at schema level
  - Naming conflicts
  - Structural conflicts
  - Classification conflicts
  - Constraint conflicts
  - Behavioral conflicts
  - Data conflicts
- Types of conflicts encountered at data level
  - Naming conflicts
  - Representational conflicts
  - Errors

After the determination of correspondences one will in general encounter the situation that schema concepts that have been identified as being corresponding are represented in different ways. For example, in correspondence C1 different names are used for the same concept (relation). This is an example of a naming conflict.

In addition to conflicts at the schema level, which are more obvious, we may also encounter conflicts at the data level, which need to be equally resolved.

In the following we will illustrate which types of conflicts may occur and what resolution strategies can be applied in order to resolve them.

Naming Conflicts: Homonyms and Synonyms

- Synonyms
  - different terms for the same concepts, e.g. `S1.PUBLICATIONS.AUTHOR` vs. `S2.PAPERS.WRITER`
  - Resolution: introduce a mapping to a common name e.g. (`S1.PUBLICATIONS`, `S2.PAPERS`) `-> S.PUBLICATIONS`
- Homonyms
  - same term used for different concepts, e.g. `S1.AUTHORS.TITLE` vs. `S2.PAPERS.TITLE`
  - Resolution: introduce prefixes to distinguish the names, e.g. `S1.AUTHORS.TITLE -> S.AUTHORS.A_TITLE` and `S2.PAPERS.TITLE -> S.PUBLICATIONS.P_TITLE`
- Use of Ontologies and Thesauri to detect and resolve naming conflicts

We can distinguish two types of naming conflicts: different terms used for the same concept (synonyms) and the same terms used for different concepts (homonyms). In the first case the resolution strategy is to introduce a mapping of the different names in the component schemas onto the same name in the integrated schema. In the second case the same names are renamed, for example, by adding a prefix.

Structural Conflicts

- Same concept represented by different data structures
- Example 1: Different attributes
  - `S1.AUTHORS.AFFILIATION` and `S2.WRITER.NROFPUBLICATIONS` have no corresponding attributes in the other schema
  - Resolution: create a relation with the union of the attributes
- Example 2: Different datatypes
  - `S1.JOURNAL.MONTH` and `S1.JOURNAL.YEAR` contain together the same information as `S2.PUBLISHED.DATE`
  - Resolution: build a mapping function, e.g. `DATE(MONTH, YEAR)`
- Example 3: Different data type constructors
  - attribute vs. relation: `S2.PUBLISHED.TYPE="Journal"` means that the publication is a journal, while `S1.JOURNALS` contains only journals
  - Resolution: requires higher order mapping languages for manipulating database schemas

Structural conflicts always occur when the same semantic concept is represented in the different component schemas by using different data structures. Structural conflicts become more probable the richer the data model is in terms of data structures provided. E.g. the object-oriented model allows many more alternative ways of modelling the same concept than the relational model. The kind of conflicts that may occur and the resolution strategies also depend heavily on the kind of data model that is used. We therefore give just a few typical examples of conflicts as the can occur in the integration of relational database schemas.

The first example is the case where the same objects from corresponding classes (=relations) have different attributes. This can be easily overcome by combining all the attributes from the different relations in the integrated relation. This means of course that many data values in the integrated database may be NULL if objects occur only in one of the component databases.

A second frequent case is where different atomic datatypes or combinations of atomic datatypes are used to represent the same data values. This type of conflict can be overcome by providing appropriate data transformation functions, as shown in the example. This requires of course that the federation layer is supporting the use of user-defined functions, one of the reasons why OO or OO-relational DBMS are considered as suitable platforms for data integration.

A slightly more complex type of structural conflict, which occurs in many variations, is the use of different type constructors to model the same (or better: semantically corresponding) data values. A typical example for this case occuring in relational schemas is the problem of using classification attributes vs. different relations in order to classify data objects.

Classification Conflicts

- Extensions of concept are different
  - Modelling constructs A and B that are recognized as corresponding can cover sets with different scopes
  - All possible set relationships
    - A=B: equivalent sets
    - A⊆B: A is a subset of B
    - A∩B: A and B have some elements in common
    - A≠B: A and B have no elements in common
- Example
  - `S1.AUTHORS` are authors of journal papers only
  - `S2.WRITERS` contains also authors of other types of publications
- Resolution
  - build generalization hierarchies
  - requires data model supporting generalization/specialization (IS-A), e.g. object-oriented
- Additional problem
  - Identification of corresponding data instances
  - "real world" correspondence is application dependent
  - Example: check the reliability of the bibliographic references in the two databases, and in case of error to trace back to the source of error

Another type of conflict can occur even if the structural representation of the data objects is identical. Classification conflicts occur when the EXTENSIONS of (possibly but necessarily structurally identical) classes/relations are different in the component databases. So it can be the case that two relations are recognized as being semantically related, but having a different scope of objects that belong to the extension. Any of the possible set relationships can describe of how the relations are related. In order to correctly represent these relationships (and thus to resolve the classification conflict) they need to be modelled in the integrated schema. For this purpose generalization (IS-A) hierarchies, as we know them for example from extended entity-relationships models, need to be constructed. After the conflict is resolved at the schema level there still remains the problem of identification of corresponding data instances from the component databases. The specification of the identification predicate is highly application-dependent, it is not unique, and in many cases it is even not definable in terms of the available attributes of the data objects, but it has to be supplied from external sources (e.g. humans manually identifying the corresponding instances)

3.4 Implementation of an Integrated Schema

- Use of a DML (e.g. SQL)
  - approach used in multidatabases, like Data Joiner
  - problem: limitation on possible mappings
- Programming
  - with access to the DDL and DML
  - problem: correctness of transformations not guaranteed
- Extensible databases (e.g. object-relational or OODBs)
  - Rich data types, access to DDL, functions
  - object-oriented models (including object-relational) are the preferred canonical models for database integration
  - problems: complexity, consistency, design methodology, …
- Schema manipulation languages (e.g. rule based, procedural)
  - most promising approach
  - problem: are currently available in research prototypes only

Different approaches can be considered in order to implement the mapping from the import schemas to the global schema. Remember: in case the data is not materialized in the integrated database, this requires a mapping of the data from the component databases into the integrated database, as well as a mapping of queries against the integrated databases to queries against the component databases.

- A simple approach is to use the SQL-view mechanism when the canonical data model is relational. Since SQL fully supports view (querying, access to data in views) all the necessary mappings are available. The problem is that more complex mappings cannot be expressed as views and access to views is restricted (e.g. limitations on updates for complex views).
- The mappings can also be implement in any programming language from which access to the DML and DDL is possible (e.g. by means of ESQL or JDBC). This approach is general, but the problem is that no guarantees are given to the developer whether the mapping he implements are correct.
- An approach that is frequently used, is to implement the integrated schema in an object-oriented database schema. The advantages are a richer set of datatypes, the access to the DDL in case the data dictionary is provided through the same interfaces as the ordinary databases, and the possibility to encapsulate more complex mappings in user-defined functions. Still the problems are similar as with the programming approach. ./.

Example: Integration of Two Relational Schemata

- SCHEMA 1

```
PUBLICATIONS(PNR, TITLE, AUTHOR, JOURNAL, PAGES)
AUTHORS(ANR, TITLE, NAME, AFFILIATION)
JOURNAL(JNR, NAME, PUBLISHER, ISSUE, MONTH, YEAR)
```

- SCHEMA 2

```
PAPERS(NUMBER, TITLE, WRITER, PUBLISHED)
WRITER(FIRSTNAME, LASTNAME, NROFPUBLICATIONS)
PUBLISHED(TYPE, PROCEEDINGS, PP, DATE)
```


In the following we will use a simple example of two relational databases (respectively schemas) in order to illustrate the process of schema integration. We choose the relational model for the purpose of compactness of presentation (relational schemas are syntactically more compact than XML schemas, for example). The analogous examples could of course be goiven also for other data models, like XML Schema.

The two example schemas use slightly different representations of publication data and our goal is to derive now one integrated schema for integrating data that originates from databases corresponding to these two database schemas.

Example: Implementation of an Integrated View in SQL

- Integrated view on AUTHORS and WRITERS
  - Assume identical instances in AUTHORS and WRITERS are identified through a mapping table

    ```
    mapping(firstname, lastname, anr)
    ```

  - Implementation in MS ACCESS SQL (barely !) doable

```
SELECT
[s2_writer.firstname] & " " & [s2_writer.lastname] AS name,          (1)
Null AS title, Null AS affiliation, Null AS anr,                    (2)
s2_writer.nrofpublications
FROM s2_writer, mapping WHERE
[s2_writer.firstname] & " " & [s2_writer.lastname] NOT IN            (3)
(SELECT [firstname] & " " & [lastname] FROM mapping)
 UNION                                                              (4)
SELECT  s1_authors.name, s1_authors.title, s1_authors.affiliation,
s1_authors.anr, Null AS nrofpublications
FROM s1_authors LEFT JOIN mapping ON s1_authors.anr = mapping.anr;   (5)
```


- A systematic approach to schema integration would provide a schema manipulation language that allows to apply operators to the import schemas such that a sequence of such operators (or rules in the rule-based approach) transform the import schemas into the integrated schema. The neccessary mappings at the data level and for query decomposition are then automatically derived from the schema mapping that is defined.

THIS SLIDE:

In the following we give a very simple example of how SQL views might be used in order to obtain an integrated schema for our sample integration problem for tables AUTHORS and WRITERS. For the correct identification of data instances from the two relations we will need a mapping of the corresponding names. This we assume is given by a table (that for example someone has produced manually). Then we can provide an SQL query that can be used for defining an integrated view on the two relations. We comment the different steps in the example (as indicated by the numbers in the code) by referring to which data integration problem is addressed:

- Resolution of the use of different data structures by combining textual fields
- Resolution of the use of different attributes by including all attributes into the integrated view
- Exclusion of the instances with occurrence in both tables
- Union of the both tables, resolution of different coverage
- Identification of corresponding instances

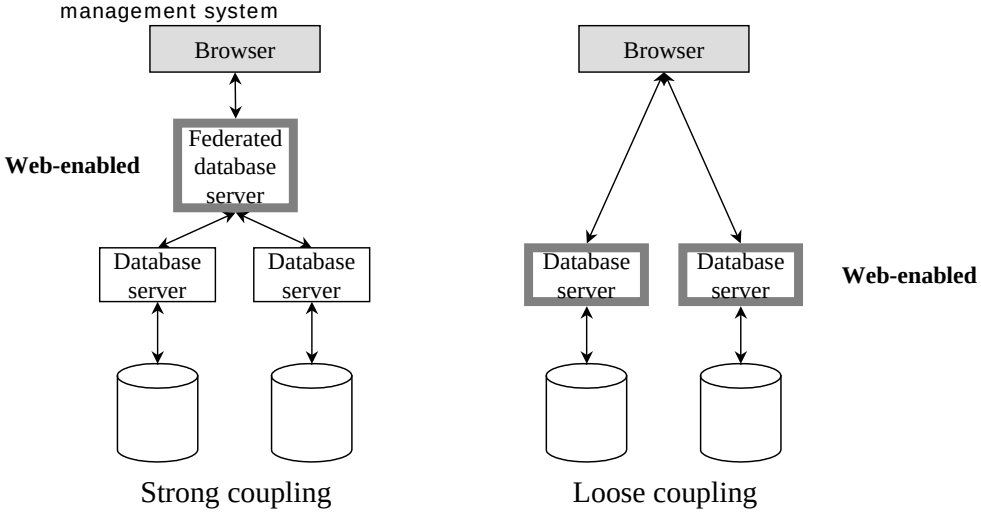
Data Integration Checklist

- Task: integrate distributed, heterogeneous (and probably autonomous) data sources

- Abstract Model ✓
  - Relational, OO, XML, XMLSchema
- Embedding ✓
  - XSLT, various wrapping technologies
- Architectures and Systems ✓
  - Multidatabases, federated databases, mediator/wrapper, data warehouse
- Methodology ✓
  - Schema integration

Web as a Loosely Coupled Federated Database

- Accessing databases through the Web can be seen as a way of loosely coupling heterogeneous data sources
  - The integration of the data occurs at the user interface rather than at the data management system



In the last part of the lecture we have seen of how data that originates from different data sources (typically databases) can be integrated using federated database system technology. In this part we will see of how the contents of a database can be made accessible over the Web, i.e. by accessing it through a web-browser. This requires that the database server is web-enabled. By web-enabling a federated database server we can make the contents of different data sources accessible through the Web (in a uniform manner). However, with web-enabling of database servers, we can also short-cut the federated database layer and directly access the data from different database servers, that are web-enabled. In that way we also obtain an integrated access to the data sources. Since in that case the integration takes place at the user interface layer, the coupling between the participating database is (very) loose, and the integration functionality is rather weak.

For illustration consider the case of accessing multiple databases by the same query (e.g. retrieve the same product from multiple electronic catalogues). Using a federated database layer such a query would be automatically decomposed and the results would be collected from the different databases by the federated database server and eventually be presented to the user. With integration at the user interface layer the user would have to "click" his/her way through the various query interfaces, formulate individual queries for each database and collect the results manually, e.g. by cut and paste. Nevertheless, in principle he/she has an integrated access, and in many cases users proceed actually exactly in that manner.

4. Summary

- Syntactic heterogeneity, communication and transaction problems have many solutions
  - Database Middleware
  - XML technology
  - Distributed objects
  - Transaction monitors
- Semantic heterogeneity remains a difficult problem
- Current developments
  - Use of more sophisticated knowledge-based tools, e.g. ontologies, declarative approaches
  - Logic language based approaches
  - Light-weight mediation on the Web based on XML technology
- Many of the heterogeneity problems illustrated for DBs also hold for application objects (except, e.g., the extensional aspects)

5. References

- Books
  - Özsu, Valduriez: Principles of Distributed Database Sytems, Second Edition, Prentice Hall, 1999.
- Articles
  - S. Busse, R.-D. Kutsche, U. Leser, H. Weber, *Federated Information Systems: concepts*, terminology and architectures, Technical Report Nr. 99-9, TU Berlin, 1999.
  - IEEE Data Engineering Bulletin, Volume 21, Number 3, September 1998, Special Issue on Interoperability.
  - Stefano Spaccapietra, Christine Parent, Yann Dupont: Model Independent Assertions for Integration of Heterogeneous Schemas. VLDB Journal 1(1): 81-126 (1992).
  - Fernando de Ferreira Rezende, Klaudia Hergula: The Heterogeneity Problem and Middleware Technology: Experiences with and Performance of Database Gateways. VLDB 1998: 146-157.
- Websites
  - www-4.ibm.com/software/data/datajoiner/
  - ibm.com/software/data/db2/extenders/xmlext
  - www.softwareag.com/germany/products/xmlstarterkit.htm

The Big Picture