CS 332/780: Object-Oriented Databases Spring 2018
Course Notes #1
Keitaro Yukawa
Department of Computer Science
Queens College, CUNY

Comparison of Relational, Object-Oriented, and Object-Relational Databases

1 Representation of Entities and Relations

	Relational DB	Object-Oriented DB	Object-Relational DB
Entity Set	table	class	table
Hierarchical (Nested) Entity Set	decomposed into a set of tables	hierarchical (nested) class	nested table with/without reference
Relation among Entity Sets	table among key attributes of entity sets	collection-type attribute in class, or relation class	nested table with/without reference

2 Characteristics

Relational DBs

- ❖ Pros
  - The table structure is simple to understand and easy to use.
  - Especially good for record-oriented data in one-dimensional tabular form.
  - SQL is firmly based on the theoretical foundation of relational algebra and relational calculus, and has become the industry-standard query language.
  - Much research and development has been done on theoretical foundations, implementations, index structures, query processing and optimizations, etc.

- ❖ Cons
  - Hierarchically or network structured objects need to be "flattened" to a set of tables.
  - A meager set of data types. No data structures except for the table structure itself.
  - Need for normal forms to reduce redundancy and update anomaly.

- ❖ Neutral
  - A separation exists between the query language (SQL) and application-development languages, which often need to be interfaced by complex protocols. (Exceptions exist, like PL/SQL which is Oracle's language extension to SQL.)
  - Identification of entities is done by values of their key attributes. Traversal between tables is done by join operations with respect to common values of common table columns. In this sense, relational DBs are said to be "value oriented". Generally, traversal by join operations is less efficient than traversal by reference pointers used in object-oriented DBs, but supporters of relational DBs think this is well offset by the simplicity and clarity of tables.
  - The field of relational DBs has matured.

Object-Oriented DBs

- ❖ Pros
  - A rich set of data types and data structures is supported: E.g., arrays, lists, sets, classes. Linked data structures can be stored in databases.
  - Good for complex, hierarchically or network structured objects.
  - Straightforward mapping from OO models to implementations.
  - OO methodologies directly applicable, such as extensibility and reuse of code by inheritance and polymorphism of method functions. Persistent and transient classes can be integrated seamlessly and transparently.

- ❖ Cons
  - No industry-standard query language exists. (The Object Data Management Group Standard was proposed in 2001 and has been influential, but has not become standard.) However, a probable reason for this is that in object-oriented DBs, clear separation between the query language and the application development language is not necessarily to advantage – more about this later in the section *Three Types of Organization of OODB Systems*.
  - Difficult to learn for users with little or no background in CS/programming.

- ❖ Neutral
  - Little or no separation between query languages and application-development languages.
  - Provide options for composing queries declaratively by a query language and algorithmically by conventional programming; the latter allows highly efficient queries. Declarative and algorithmic queries and conventional programming can be mixed.
  - Identification of objects can be done by reference pointers to them as well as by values of key attributes. Traversal of relations between classes is done by reference pointers representing relations, possibly with relation objects. In this sense, object-oriented DBs are said to be "reference oriented". (Exceptions are OODBs with object-relational mapping to be discussed later.) Generally, traversal by reference pointers is more efficient than traversal by join operations used in relational DBs, but it may lead to less perspicuous and less declarative queries.

Object-Relational DBs

Object-relational DBs as typified by the SQL-99 Standard extend RDBs with object-oriented concepts in the form of nested relations, reference pointers, inheritance, and method functions. Method functions are attached to relations and coded in languages like PL/SQL, Java, and PSM (Persistent Stored Module). Most RDB vendors have now incorporated at least some of the object-oriented features of the SQL-99 standard. Object-relational DB schemas using reference pointers and inheritance can be similar in their essentials to object-oriented DB schemas.

Languages Associated with Database Systems

- Data Definition Language (DDL): Used to define schema (format), type, and structure of data. **Static definition of data schema**.
- Data Manipulation Language (DML): Used to insert, delete, update, and retrieve data. Query languages are usually considered part of DMLs. **Dynamic manipulation of data**.
- Application Development Language (ADL): Used to build database applications software. Can be a stand-alone language specifically designed for the DB system, or an existing general-purpose language like C++ and Java, or a script language like JavaScript.

Three Types of Organization of OODB Systems

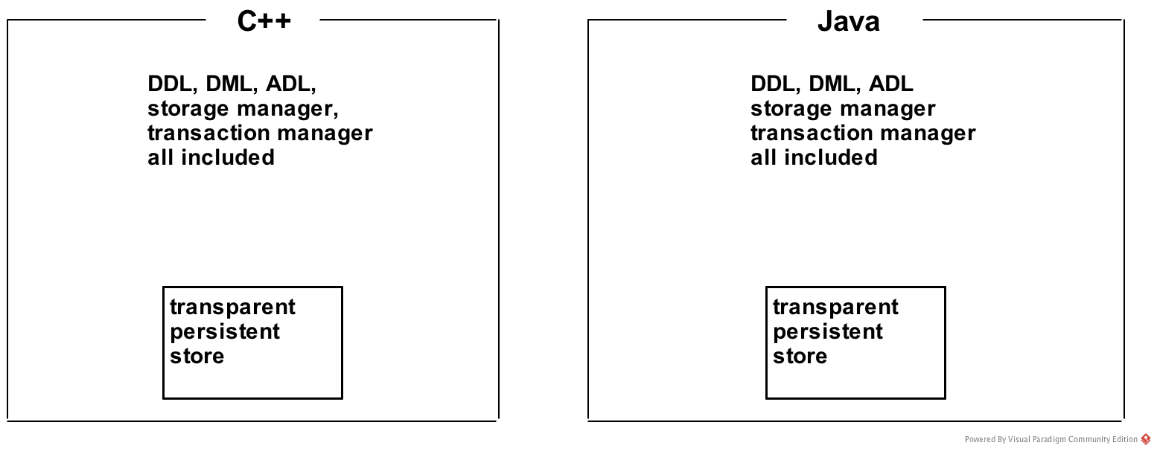
Presently we may distinguish three broad types of organization of OODB systems. These are not clear-cut distinctions, and there are and will be DB systems that blur the distinctions. In this section, “programming languages” will be used in a wide sense to include script languages that are powerful enough for building specific application software.

1 Transparent Persistent Object Store Systems

A persistent object data store is implemented within the framework of a general-purpose programming language (like extensions of Java and C++) or a special-purpose database programming language. Such a language supports all of DDL, DML (including a query language), ADL, storage/transaction managers, and is tightly coupled with the object store. This is called a transparent persistent object store because it can be accessed (“viewed”) from all parts of the programming language directly without going through complex interface protocols.

The advantage of this organization is that a single programming language supports everything needed for database applications software development: DDL, DML, ADL, and an object store. Persistent classes are simply part of the language and are integrated seamlessly with transient classes. Effectively, there is no separate “database system” as such. The disadvantage is that database applications development needs to commit to a single language, making this organization unsuitable for multi-language, heterogeneous environments.

This is a natural organization for OODBs since the tight coupling of data (class objects) and operations (method functions) is an essential feature of the object-orientation concept.

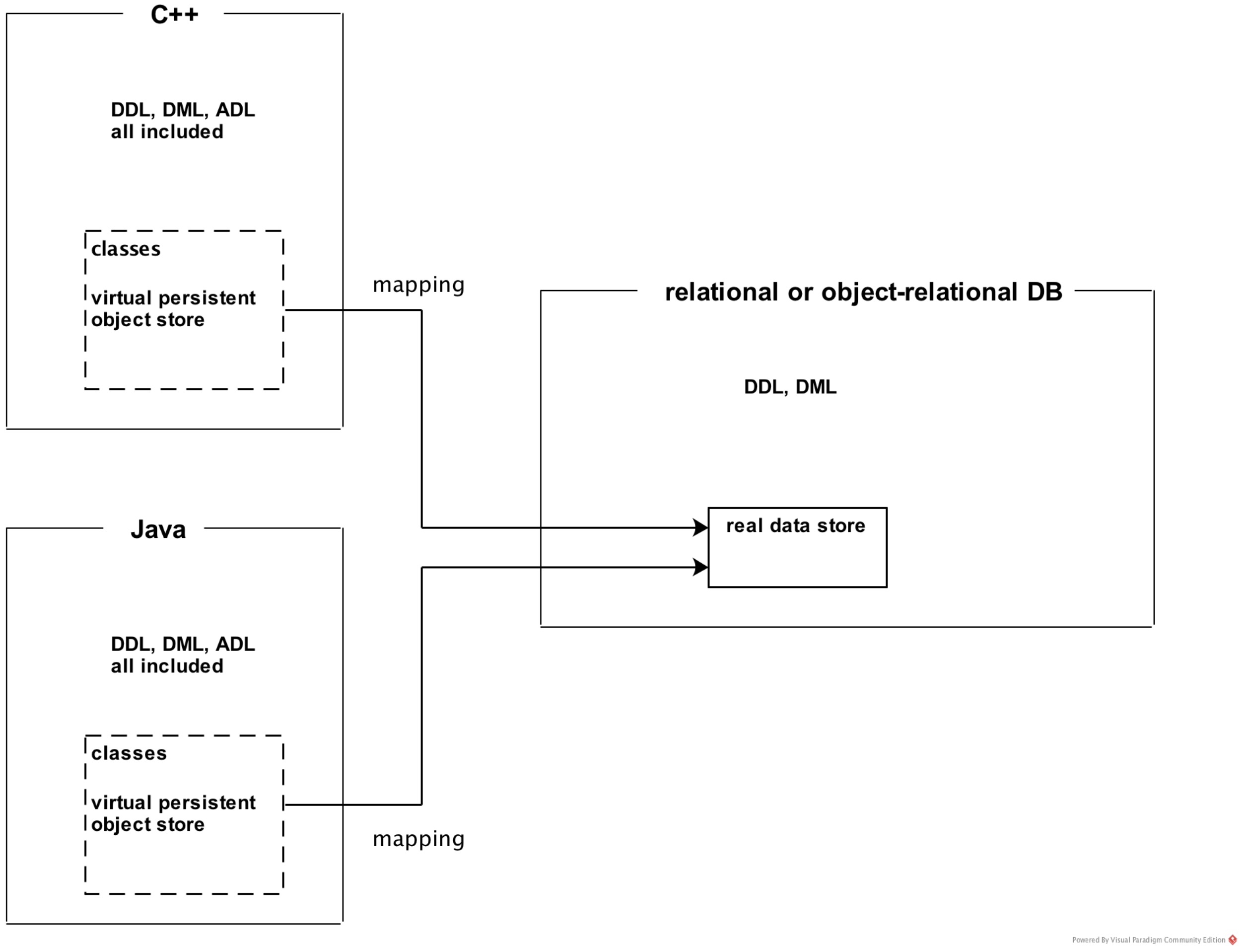

C++
DDL, DML, ADL, storage manager, transaction manager all included
transparent persistent store
Java
DDL, DML, ADL storage manager transaction manager all included
transparent persistent store
Powered By Visual Paradigm Community Edition

2 Object-Relational Mapping Systems

As in transparent persistent object store systems, the database system provides a full object-oriented database language supporting DDL, DML, and ADL. This language may be based on an existing general-purpose programming language like Java and C++, or may be an independent, self-contained language. In this type of system, *a real persistent object store does not exist, but is implemented by mapping class schemas and class objects to a relational or object-relational database*. From the users’ standpoint, there is a “virtual persistent object store”, which is implemented by a real data store in a relational or object-relational database. It is desirable that the virtual object store is made transparent (as described in §1) so that developers are not hindered by interface protocols and low-level mapping details.

The advantage of this organization is that it gives the best of both worlds. Developers have the choice of working with the virtual object-oriented database, the relational/object-relational database, or both. In this sense, DB systems of this kind are hybrid systems. A potential disadvantage is that if relations between classes are implemented by pure tables without use of references, traversal between classes is mapped to join operations on tables, hence traversal efficiency of references is lost.

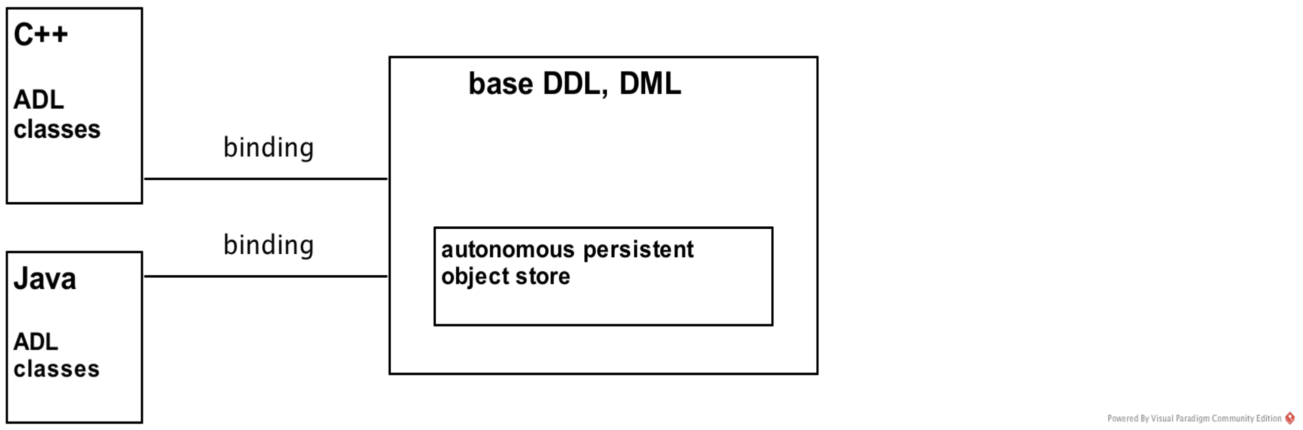
This organization is suitable for multi-language, heterogeneous environments using a relational/object-relational DB as the common data store. Hibernate and DataNucleus are examples taking this approach.


C++
DDL, DML, ADL all included
classes
virtual persistent object store
mapping
relational or object-relational DB
DDL, DML
real data store
Java
DDL, DML, ADL all included
classes
virtual persistent object store
mapping
Powered By Visual Paradigm Community Edition

3 Autonomous Object Store Systems

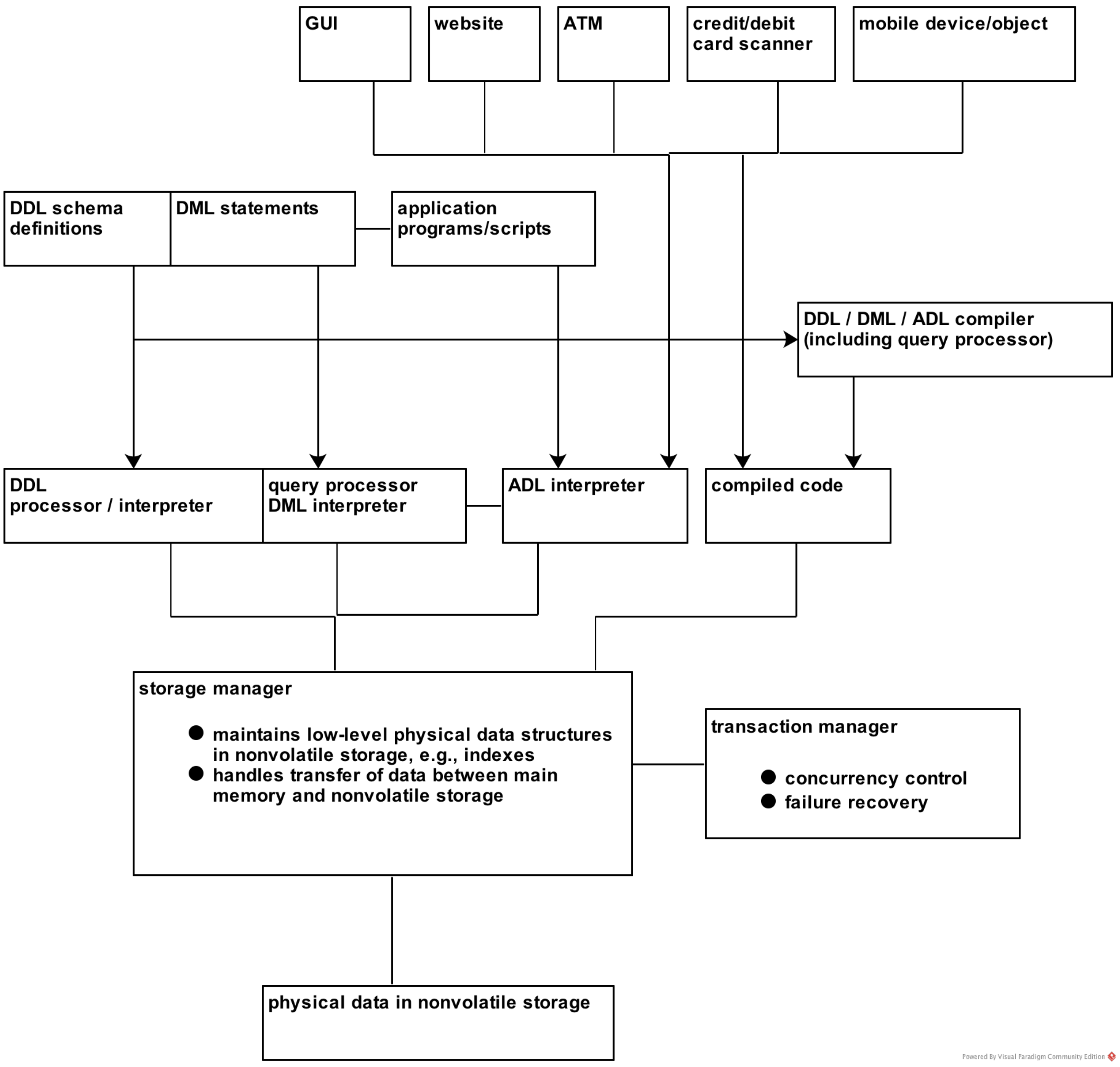
A real persistent object store exists autonomously and independently of ADLs. It is coupled with its own DDL and possibly DML – let us call these the "base" DDL and DML. For applications software development, the persistent class schemas defined in the base DDL need to be interfaced to separate ADLs by protocols. These ADLs may be object-oriented or non-object-oriented. If an object-oriented ADL supports its own DDL and class schemas defined in it are automatically mapped to the base DDL in a way that preserves the semantics of the associated method functions, the view of the object store from the ADL is transparent. In contrast, if manual interfacing is required between class schemas and/or method functions in the ADL and the class schemas in the base DDL, the view of the object store is non-transparent. In the latter case difficulties may arise because of the non-transparent separation of class objects residing in the object store and method functions coded in ADLs, since this separation runs counter to an essence of the object-orientation concept, namely the tight coupling of class objects and method functions.

The advantage of this organization is suitability for multi-language, heterogeneous environments – database applications development need not commit to any single ADL. The Object Data Management Group Standard proposed in 2001 takes this approach.


C++
ADL classes
binding
binding
Java
ADL classes
base DDL, DML
autonomous persistent object store
Powered By Visual Paradigm Community Edition

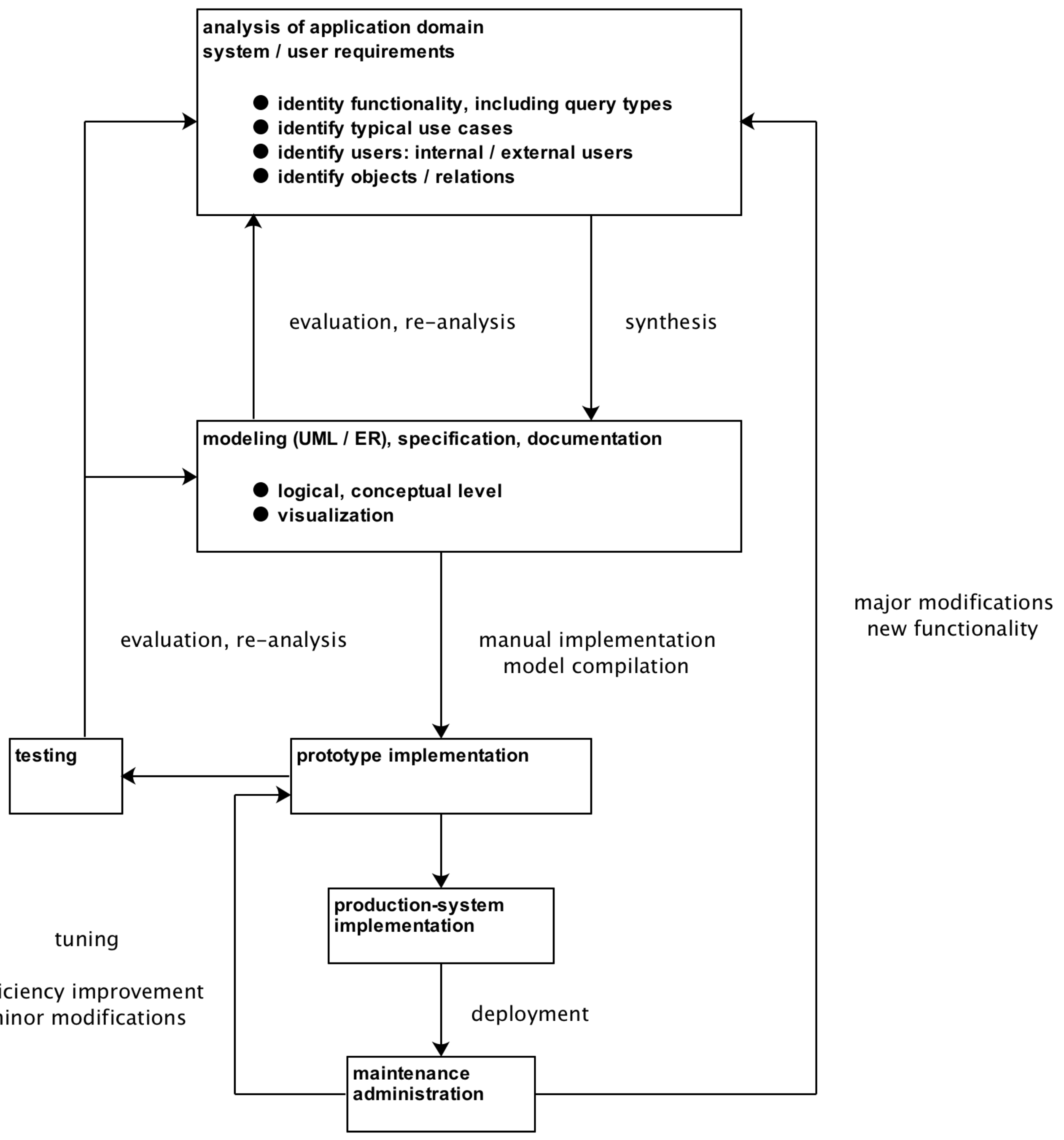
In comparison it is instructive to note that, historically, almost all relational DB systems have embraced the autonomous data store organization. When the relational DB concept was originally proposed in the ′70s, one of the primary motivations was in fact to *free* database systems from application-development programming languages so that data in table form can be defined, manipulated, and retrieved uniformly by declarative rather than algorithmic means.

DB System Architecture



GUI
website
ATM
credit/debit card scanner
mobile device/object
DDL schema definitions
DML statements
application programs/scripts
DDL / DML / ADL compiler (including query processor)
DDL processor / interpreter
query processor DML interpreter
ADL interpreter
compiled code
storage manager
● maintains low-level physical data structures in nonvolatile storage, e.g., indexes
● handles transfer of data between main memory and nonvolatile storage
transaction manager
● concurrency control
● failure recovery
physical data in nonvolatile storage
Powered By Visual Paradigm Community Edition

DB Development Cycle



Actors: People who

- **analyze application domain, system / user requirements**
- **model, specify, document**
- **implement -- programmers**
- **maintain, administer**
- **use -- internal / external users**