Of Objects and Databases: A Decade of Turmoil

Carey & DeWitt

Ten Years Ago

1. Extended RDBs: adding abstract data types
2. Programming languages: persistence, transactions, OO
3. OODBs
4. DB Toolkits

Extended RDBs

Evolve RDBs:

- open up type system to allow ADTs
- query optimization for ADTs
- procedures as data types
- just say no to pointers

Persistent PLs

Evolve PLs:

- keep standard programming model

+ persistence
+ transactions

$=$ P{Smalltalk,CLOS,CLU,Trellis/Owl,C++,ML,Modula-3,Java}

Issues:

- orthogonality
- persistence designation
- binding/namespace management for roots
- type systems and safety
- implementation: transparent navigation, maintenance, GC

OODBs

Revolution:

- full-blown database system
  - query language
  - indexing
  - query processing and optimization
  - version management
- \+ OO data model
  - OO language
  - navigation

Revolution $\Rightarrow$ lack of standards

DB toolkits/components

Extensibility:

- DBMS kernel
- \+ tools for rapid development of domain-specific DBMS
- specialized DBMS per application (CAD/CAM, OIS, GIS)
  - query language
  - access methods
  - storage organization
  - transaction model

Today

The dead and wounded:

- toolkits and PPLs commercially dead
- OODBs failed to deliver

New approach: OO language wrappers for RDBs

And the winner is:

Extended RDBs, aka *object-relational DBs*

RIP

Toolkits:

- too much expertise required
- inflexible, awkward, incomplete
- OO and OR yield enough extensibility

PPLs:

- still active in academia

but no commercial implementations

but significant impact on navigational interfaces of OODBs

+ persistence models, swizzling, GC $\rightarrow$ OODBs

but OODBs are failing to deliver

OODBs

Prescription:

- complex objects
- object identity
- encapsulation
- inheritance and substitutability (i.e., subtyping)
- late binding
- computational completeness
- extensible types
- persistence
- secondary storage management
- concurrency control and recovery
- *ad hoc* queries

+ multiple inheritance
+ static v dynamic type checking
+ distribution
+ long transactions
+ version management

OODB achievements

- data models: composite objects, schema evolution
- query languages and processing: path expressions, nesting
  - indexing: path expressions, path updates, class hierarchy queries
  - pointer-based joins, complex objects
- client/server architectures: transactional caching, crash recovery
- benchmarks and performance studies
- version and configuration management
- prototypes and products
- draft ODMG-93 standard for ODL and OQL
  - C++, Smalltalk, Java bindings

OODB failings

- still no real standards
- ODMG not yet entirely adopted by vendors
  - only O_2 has standard OQL
- no views (research still needed)
- schema evolution bites compared to RDBs
- tight coupling between OODB and application PL (C++)
- lack of robustness, scalability, fault tolerance
- fewer end-user and application development tools
- new standards for interfacing to RDBs: ODBC, thin clients/fat servers
- ODBC is not OO
- conservative users: relational only just caught on, let alone OO

ORDBs: evolution not revolution, or "steady as she goes"

- start with relational model and SQL
- adopt select OODB data model and query language features
  - support richer object structures and rules: ADTs, *row* types
    - subtyping
    - functions
    - encapsulation
    - optional UIDs (i.e., say maybe to pointers)
    - rules/triggers
- subsume RDBs
  - top-level named relations
  - high-level queries
  - stored and virtual collections
  - updatable views
  - separation of data model and performance
  - SQL + object queries (path exprs., methods, nested sets)
- interoperate with tools and multidatabase middleware
  - access from multiple languages
  - layered persistence-oriented language bindings
  - SQL support
  - query-shipping client/server interface

Still no standards but vendors are concerned (SQL3)

OO client wrappers

- language-specific
- classes act as proxies for data in underlying database
- object definition/construction tools
- allow OO apps against legacy databases

Weaknesses: impedance mismatch

- *ad hoc* queries in SQL against underlying schema
  $\Rightarrow$ paradigm mismatch between programming and querying
- what to code in OO language v database facilities

Related developments

CORBA: for large distributed OO apps

- interoperable object RPC
- name services for distributed resources
+ factorable services:
  - persistence
  - collections
  - indexing
  - transactions
- will fail on performance:
  - factored services never performed for DB systems
  - fine-grained access

OLE, COM, DCOM, OLE/DB *the evil empire*

Java

Middleware for multidatabase access

- Microsoft Access client (via ODBC)
- IBM DataJoiner RDB query server to relational backends
- UniSQL/M

2006: a database odyssey

Commercial products will incorporate today's research

Servers:

- ORDBs will realise OODB promises: ADTs, language integration
- OO row types
- SQLn: OO views, authorization, triggers, constraints
- working with . . .

Clients:

- high-function
- OO
- caching front-ends

Clients and servers together provide:

- unified object model for queries and navigation
- method execution at either
- queries and query fragments at either
- triggers, referential integrity, other constraints at either

OODBs will remain niche solutions for moderate DBs

OO client wrappers will evolve to be integrated with the server

Research

- server functionality and algorithm improvements
- client integration
- parallelization
- legacy data sources
- standards