Integrating Programming Languages & Databases

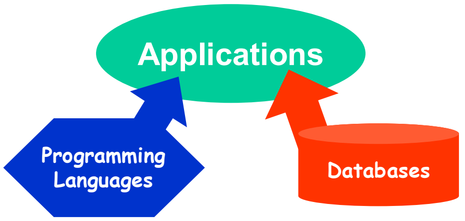
William Cook

Assistant Professor, UTCS

(with thanks to students in CS 395T fall 2003)

Programming Language + Database

- System = Computation + Persistence



Applications are point of integration

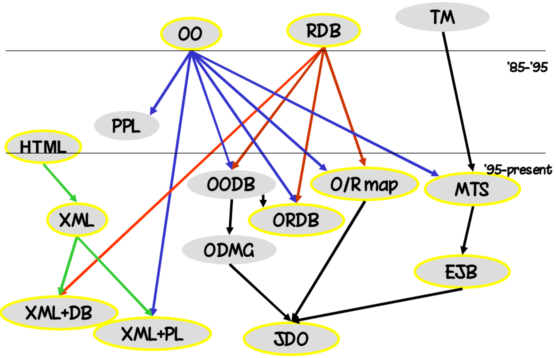
Examples

- Mail/news/IM server/client
- E-Commerce application
- Spreadsheet, word processor
- Multi-user games
- Web applications
- Business (ERP, CRM, PRM, HRM, SCM)
- Source code control, file server
- Bibliography DB
- Factory/process control systems
- *Just about any system you can thing of…*

Approaches

- Lots of solutions
  - Embedded SQL
  - Call Level Interfaces (CLI)
  - Persistent programming language (PPL)
  - Database programming languages (DBPL)
  - Object-oriented database (OODB)
  - Transaction middleware (EJB, COM+)
  - Object-relational mapping (O/R)
- Lots of *partial* success…

History



Goals

- Persistent systems that are
  - High performance, scalable, reliable
  - Logical, clean programming model
    - consistency, static typing
  - Scales to multiple, concurrent…
    - Users (concurrency)
    - Machines (clustering, redundancy)
    - Developers (modularity)
  - Effective design, maintenance & evolution

What's the problem?

Negative Synergy

- Connecting PL and DB is hard because
  - Models don't match: "Impedance Mismatch"

Flat tables	Complex objects
Declarative queries	Procedural programs
Transactions	Synchronization
Optimization	Modularity

  - Cultural mismatch
    - DP people don't understand PLs
      - "everything is a database"
    - PL people don't understand DBs
      - "why can't I write everything in Java?"

Factors for Evaluating Solutions

- **Technical metrics**
  - Performance
    - throughput
    - latency
  - Reliability
  - Scalability
    - Amount of data
    - Number of users
    - Complexity
    - Rate of change
    - Team size
  - Consistency
  - Correctness
- **Human metrics**
  - Modularity
  - Encapsulation
  - Development effort
  - Maintenance costs
  - Scalability of group
  - Clarity
  - Beauty
  - (Hard to measure)

Most solutions only address some of these factors

What Are Databases Good For?

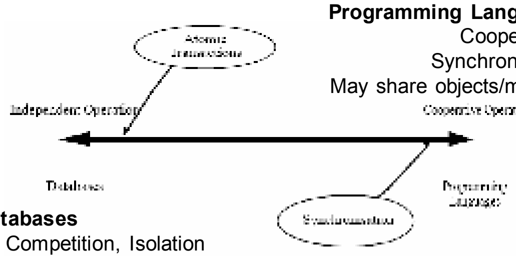
1. **Search algorithm compiler**
   - Queries specify what to find, not how
   - Optimizations
     - Ordering of operations
     - Indexes, content heuristics
     - Physical characteristics (e.g. page size)
   - Runtime compiler
2. **Concurrency control**
   - Manage concurrent reads and writes
   - Transactions
   - ACID: Atomic, Consistent, Isolated, Durable

Programming Languages Good For?

- **General-purpose computation**
  - Algorithms
    - Cooperative concurrent computation
  - Abstraction
    - Reuse, Modularity
- **Performance**
  - Good at local optimizations
  - Global optimization is much harder
    - Object-oriented programs are difficult to optimize
- **Summary: anything and nothing…**

Two Views of Concurrency

Programming Languages
Cooperation,
Synchronization
May share objects/memory



Databases
Competition, Isolation
Independent operation
Only point of sharing is database
Atomic

[S. Blackman: Concurrency – the Fly in the Ointment]

How to put them together?

Approaches to discuss

- **Database APIs**
  - "Call Level Interfaces"
- **Persistent objects**
  - Object-Oriented Databases
  - Persistent Programming Languages
  - Object-Relational Mappers
- **Transaction managers**
  - MTS/COM+, EJB
- **Blend of the above**
  - Java Data Objects

Approaches (not discussed)

- **Other ideas**
  - Embedded programming languages
  - Active databases
  - Database Programming Languages
  - Object-relational databases
  - XML
- **High rate of change…**
  - Many new proposals every year for last 10 years

Call Level Interfaces

The "state of the art" in practice

Call Level Interface (CLI)

- **Set of APIs to run SQL commands**
  - These are the workhorse of database interfaces technologies
- **Basic operations**
  - Connect to database
  - Execute SQL commands (with parameters)
  - Iterate over result set (if there is one)
- **Variations**
  - Access meta-data, convert data
- **Note**
  - An interface to the database engine, not to a particular logical database

Some DB Interface APIs

Embedded SQL	???	Required preprocessor
ODBC	1992	For "C"
SQL/CLI	1995	Standard based on ODBC
DAO	~1992	VB and Jet DB engine
JDBC	1996	Java version of ODBC
RDO	~1996	VB and any DB
OLE DB	~1996	high-performance, C level
ADO	~1996	VB and web scripting
ADO.NET	~2001	All languages, uses

ADO Example

```
Dim db as new ADODB.Connection
Call db.Open("ODBC;DSN=" & DatabaseName
         & ";UID=" & UserName & ";PWD=" & UserPassword)

Dim rs as new ADODB.recordset

Call rs.Open(db, "SELECT Name, Phone FROM Employee")
Write "<Table>"
Do while not rs.EOF
   Write "<TR><TD>" & rs.Field("Name").value & "</TD>"
   Write "<TD>" & rs.Field("Phone").value & "</TD></TR>"
   rs.MoveNext
Loop
Write "</Table>"
```

Calling Database Procedures

- **Call a simple database function**
  - pass a status parameter
  - return list of rows and number of rows
- **What we would like to write**

(List, NumRows) = DB.GetRecords(Status)

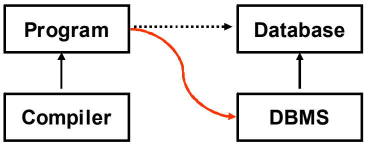
```
Set objCon = New ADODB.Connection
Set objCom = New ADODB.Command

'Creating the DB connection string
'Please change the below connection string as per your
server and database being used.
objCon.ConnectionString =
"PROVIDER=SQLOLEDB.1;PASSWORD=;PERSIST
SECURITY INFO=TRUE;USER ID=sa;INITIAL
CATALOG=TestSQL;DATA SOURCE=Rockets"

'Opening the connection
objCon.Open objCon.ConnectionString

'assigning the command object parameters
With objCom
 .CommandText = "GetRecords"
    'Name of the stored procedure
 .CommandType = adCmdStoredProc
    'Type : stored procedure
 .ActiveConnection = objCon.ConnectionString
End With

'Create 2 output parameters
Set objPara = objCom.CreateParameter("rows",
                              adInteger, adParamOutput)
Set objpara2 = objCom.CreateParameter("Status",
                              adVarChar, adParamIn, 50)
objpara2.Value  = InputStatus

'Append the output parameters to command object
objCom.Parameters.Append objPara
objCom.Parameters.Append objpara2

'Store the result in a recordset
Set objRS = objCom.Execute

'Open the recordset
Do While Not objRS.EOF
 For k = 0 To objRS.Fields.Count - 1
  write objRS(k).Name & ": " & objRS(k).Value
 Next
 objRS.MoveNext
Loop

'retrieve the output parameters values
MsgBox "Totalrecordsreturned: " & objPara.Value
MsgBox

'closeconnection
objRS.Close
objCon.Close
```

CLI Issues

- **No static syntax checking (!)**
  - rs.**Open("SELECT Name, Phone FROM Emp")**
- **No static type checking**
  - **rs**.Field("Phone").value
- **Complex, error-prone programming**
  - lots of code that doesn't do much
- **Hard-coded dependencies**
  - difficult to maintain

CLI Issues

- **No *semantic* connection between database and program**



Program

Database

Compiler

DBMS

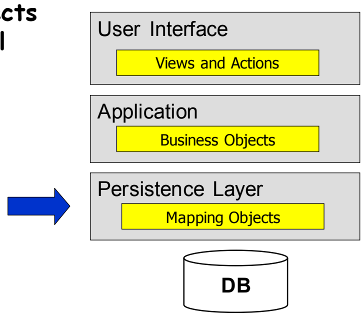
CLI Summary

- **Everyone knows it is terrible**
- **Lots of effort to do better**
- **Yet CLI is still ubiquitous**

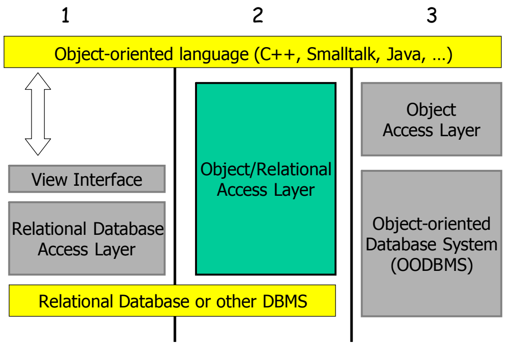
Object/Relational Mapping

Architecture of Business System

- **Create a mapping between objects and relational database**


User Interface
Views and Actions
Application
Business Objects
Persistence Layer
Mapping Objects
DB

Database Access Layer Options

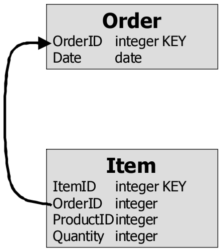

1
2
3
Object-oriented language (C++, Smalltalk, Java, …)
View Interface
Relational Database Access Layer
Relational Database or other DBMS
Object/Relational Access Layer
Object Access Layer
Object-oriented Database System (OODBMS)

Mapping Classes to Tables

- **Instance variables in object**
  - Columns in table
- **References to other objects**
  - Foreign keys
    - Single valued and multi-valued
    - Relationships have "two sides"
- **Inheritance**
  - Several strategies

Many-to-One

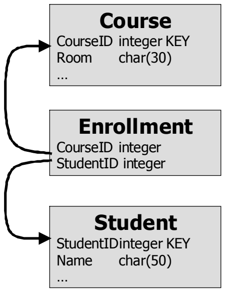
```
class Order {
   int       OrderID;
   date      Date;
   Item      items[];
}
class Item {
   int       ItemID;
   Order     order;
   Product   product;
   int       quantity;
}
```


Order
OrderID integer KEY
Date date

Item
ItemID integer KEY
OrderID integer
ProductID integer
Quantity integer

Many-to-Many

```
class Course {
   int       CourseID;
   String    room;
   Student enrolled[];
}
class Student {
   int       StudentID;
   String    name;
   Course    registered[];
}
```

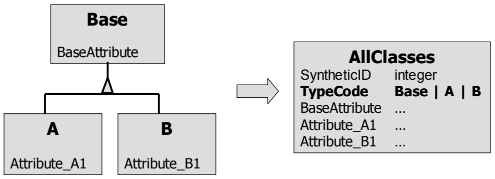

Course
CourseID integer KEY
Room char(30)
…

Enrollment
CourseID integer
StudentID integer

Student
StudentID integer KEY
Name char(50)
…

One Inheritance Tree = One Table

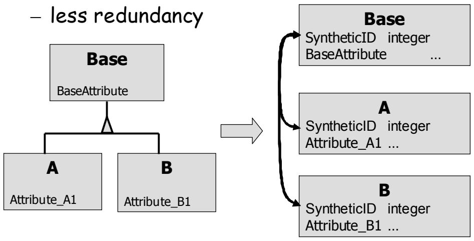
- **All in one table**
  - fast query with cost of overloading
  - ambiguity: if attributes can be null


Base
BaseAttribute
A
Attribute_A1
B
Attribute_B1

AllClasses
SyntheticID integer
TypeCode **Base | A | B**
BaseAttribute …
Attribute_A1 …
Attribute_B1 …

One Class = One Table

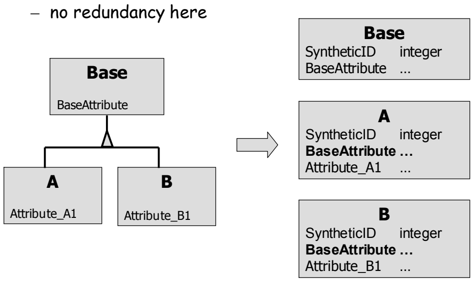
- **Map each class to a separate table**
  - fast query for base type
  - requires join for children
  - less redundancy


Base
BaseAttribute
A
Attribute_A1
B
Attribute_B1

Base
SyntheticID integer
BaseAttribute …
A
SyntheticID integer
Attribute_A1 …
B
SyntheticID integer
Attribute_B1 …

One Inheritance Path = One Table

- **Map each class to a separate table, include parent attributes**
  - fast query for children, slower for base type
  - no redundancy here



Base
BaseAttribute

A
Attribute_A1

B
Attribute_B1

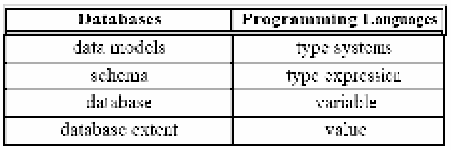
Base
SyntheticID integer
BaseAttribute ...

A
SyntheticID integer
BaseAttribute ...
Attribute_A1 ...

B
SyntheticID integer
BaseAttribute ...
Attribute_B1 ...

Issue: Type Mismatch?

- **Object-relational mapping shows..**
  - Object and relational types are compatible



Databases	Programming Languages
data models	type systems
schema	type expression
database	variable
database extent	value

- **Is it really Object-Oriented?**
  - Object Behavior (methods) are not in DB
    - some OODBs have done this

Issue: Efficiency

- **Objects are loaded *when needed***
  - Leads to "one at a time" load model
  - Many-valued sets can be loaded together
- **Result:**

  Many queries (could be hundreds!)

  → fixed by caching?

  → cache coherence across machines?
- **"Clustered Read" problem**
  - How do you provide high performance access to large chunks of data via an O/R access layer?

Scalability via Replication

- **Load-balancing Multiple machines**
  - Load is distributed across machines
    - Scalability
    - Availability
  - Use of shared resources must be controlled
- **Problems**
  - Cache coherency
    - ensuring that changes on multiple machines are consistent
  - Locking
    - Distributed transactions

Concurrency

- **Two transactions accessing same *object***
  - How do you know what operation they will perform?
  - Both read, or will either of them write?
- **Approaches to Isolation:**
  - 1) Copying
    - Each transaction have a copy, in case one writes
    - What if both update their copy of the object?
    - How will the resulting changed be merged?
  - 2) Locking
    - Only one transaction at a time can access the object
  - 3) Distinguishing reads/write methods
    - Difficult to do for general OOP

Middleware

Middleware

- **Transaction is a *unit of work***
  - *Begin* Transaction
    - Do work…
  - *Commit* or *Abort*
- **Key issues**
  - Concurrency
    - Multiple transactions running together
  - Failure
    - Handling catastrophic system failures

ACID Transaction Principles

- **Properties that must be preserved by DBMS**

A	**Atomic**	**Either *all* the operations in a transaction are performed or *none* are**
C	**Consistent**	**The database must be in a consistent state at the start and end of every transaction**
I	**Isolated**	**There is no interference between concurrent transactions**
D	**Durable**	**Once a transaction completes, its affect is permanent even in the event of complete system failure**

Transactions from Client Viewpoint

- **Client code must indicate transaction boundaries**
  - BeginTransaction
    - Do work…
  - EndTransaction
- **This is a problem for modularity**
  - How do we assemble a composite transaction from multiple parts, if each is beginning/ending its own transaction
- **Review solutions in Middleware area**

Microsoft Transaction Server

- **Problem**
  - Programs that use begin/end transaction are not **reusable**
  - Transactions may involve multiple machines and distributed computation
  - How do transactions and objects interrelate?
- **Need for**
  - Compositional distributed transactions

MTS – Approach

- **Declare certain *classes* as <u>transactional</u>**
  - new/require/support transaction
- **Unify object and transaction <u>lifetime</u>**
  - creating new/required object <u>starts</u> transaction
  - supporting objects <u>enlisted</u> in transaction
  - transaction <u>commits</u> when main object is freed
- **Resource dispensers track operations**
  - database, email, message queue, (file system)
- **No explicit entity-relational mapping**

MTS → EJB

- **Evaluation**
  - Good model of *modular transactions*
- **Basis for design of EJB**
  - session beans = MTS transactional objects
  - entity beans were added
    - (have to be different in some way)
    - Used for entity-relational mapping

Java Data Objects

JDO Introduction

- **Standard for transparent Java object persistence**
  - Developed through the Java Community Process (JCP).
  - JDO became a standard in March,2002
  - Designed to allow "pluggable" vendor drivers
- **Combination of..**
  - Orthogonal persistence
  - CLI
  - Object-relational mapping

Goals

- **Transparent object persistence**
  - Minimal constraints on building classes
  - No new data access language
- **Use in a range of implementations**
  - J2SE (client-server)
  - J2EE (Enterprise Java Beans)
- **Data store independence**
  - Relational
  - object, object relational
  - file system…

Why JDO?

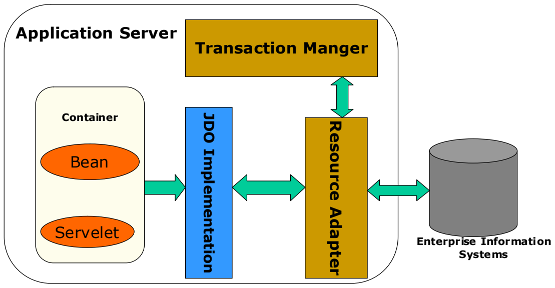
- **From an Application developer's perspective:**
  - No need to write persistent management code
  - Applications view data and relationships as a class hierarchy
  - Data store independence
    - No vendor lock-in
    - Portability between relational and object data stores
  - Object oriented features are supported
  - No coding using SQL

Using JDO

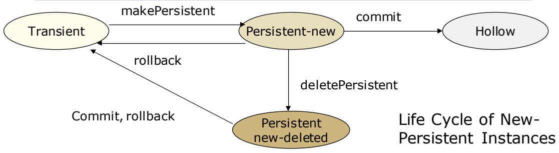
- **Build process**
  - Write your classes
  - Describe persistence needs in a XML file
  - Apply JDO enhancer to add hooks to .class
- **Main classes**
  - Use the PersistentManager to create a Transaction or a Query
  - Use Transaction to control transaction boundaries
  - Use a Query to find objects

JDO Architecture

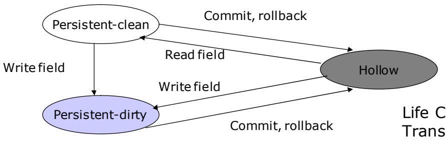
- **Managed JDO architecture - EJB**
  - Implicit connection and transaction



Life Cycle of JDO Instances



Life Cycle of New-Persistent Instances



Life Cycle of Datastore Transactions

JDO Query Example

Selects all 'Employee' instances from the candidate collection whose 'salary' is greater than 3000

```
Class Employee
{
    String name;
    Float salary;
    Department dept;
    Employee boss;
}
Class Department
{
    String name;
    Collection emps;
}
```

```
Class eClass = Employee.class;
Extent cEmp= pm.getExtent(eClass, false);
String fil = "salary > 3000";
Query q = pm.newQuery(eClass, cEmp, fil);
Collection emps = (Collection) q.execute();
Employee e = (Employee) emps.getItem(1);
print ( e.getName() );
```

The salary comparison value is parameterized.

```
String param = "float sal";
q.declareParameters(param);
Collection emps =
    (Collection) q.execute(new Float(30000));
```

JDO Issues

- **Like Orthogonal Persistence and O/R:**
  - Does not solve "clustered read"
  - Issues with distribution
- **Like CLI:**
  - No syntax static of database code
  - No static typing of database interface

Summary

- **Negative Synergies**

Programming Languages	Databases
Modularity	Query Optimization
Object Sharing	Transactions
Static Typing	Dynamic SQL
Imperative Programming	Declarative Queries
Sequential execution	Batch operations

- **Important research opportunity**