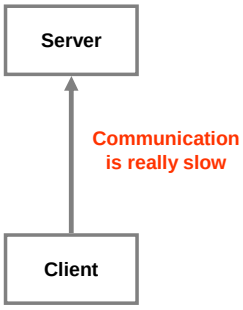
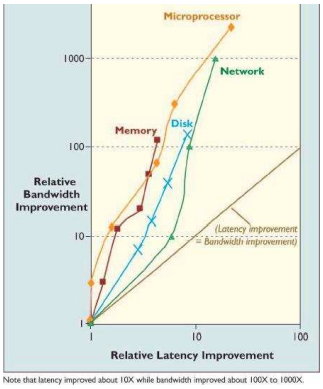
A Meditation on AppleScript, Latency, Databases, Web Services, etc.

William R. Cook
University of Texas at Austin

A Design Problem



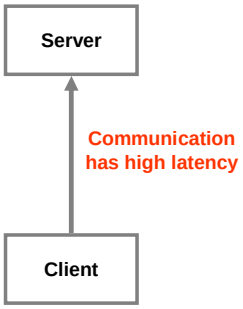
Latency Lags Bandwidth



Note that latency improved about 10X while bandwidth improved about 100X to 1000X.

David Patterson
CACM
October 2004

Refined Design Problem



Pure Object-Oriented Solution

- **Design**
  - Objects, methods, inheritance, interfaces, etc…
  - Automatic remote proxies, marshalling, etc…
- **Pluses**
  - Familiar, elegant solution
  - Lots of trained developers
- **Negatives**
  - Doesn't work in practice

> "Abstraction is great… unless the **properties being abstracted** are the **essence of the problem** you are try to solve" – **Steve Cook**

If HTTP were designed on CORBA?

- **Similar to FileSystem interfaces, etc**

```
interface Container {
  Container Sub(String n)
    throws FileNotFound;
  File Get();
  File Invoke(HashMap p);
}
```

```
interface File {
  int Length();
  String Text();
  String Type();
  String Encoding();
  long Modified();
}
```


Lots of Round-Trips

- **CORBA-HTTP Client**

```
Container c;
c = root.Sub("papers").Sub("index.htm");
String s = c.Get().Text();
```

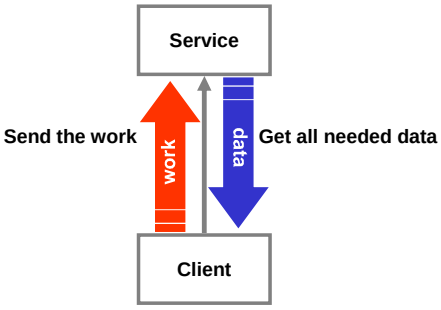
- **Automatic proxies for each intermediate object**
  - Round-trip for each "."
  - Work-arounds are possible, but complex
- **Web would have failed?**

> Can't take something that works locally and make it remote. But taking something that works remotely and using it locally is ok. – **Don Box**

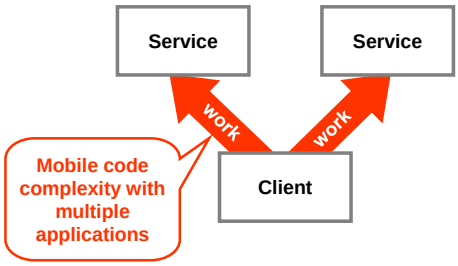
Abstraction Trap

- **Pesky "Non-functional" requirements**
  - …like performance
- **Simple elegant solutions don't always scale up**
  - Can't profile and optimize…
  - Inefficiency is spread throughout architecture
- **The question is…**
  - Can approaches that do scale up be made simple and elegant?

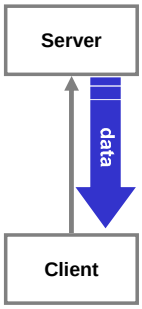
Solution Directions



Mobile Code



Which Data?



- **How does the client communicate what data it needs?**
- **How does it know what it needs?**
- **"Value Object" Pattern**
  - not enough

AppleScript (1991)

- **MacOS allowed 60 process switches a second**
  - Client = Script
  - Server = Application
- **Send the work**

```
tell application "Word"
  set the color of every character
      of document 1 whose font is "Courier" to Red
end tell
```

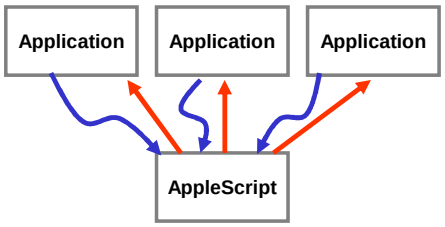
- **How it works**
  - Applications publish terminology
  - Programs contain object references that use terminology

Communication Model

- **Application Terminology**
  - Virtual object model: properties and elements
    - properties are single-valued: "font", "color"
    - elements are multi-valued: "document", "character"
  - Verbs
    - generic: set, copy, delete
    - also define specific ones
- **Object Specifiers**
  - Standard message format for object references
- **Apple Events**
  - Tagged tree-structured generic storage
  - Pretty much the same as XML

AppleScript as Glue

- **AppleScript → application** — **Apple Events**
- **Applications → AppleScript** — **Open Scripting**



More on Object References

- **First-class**

```
set x to a reference to
    every character of document 1
                whose font is "Courier"
set color of x to Red
```

- **Multiple applications**

```
set name of document 1 to application "Word"
  to name of document 1 to application "Excel"
```

  - Get the data from one, send work to the other

AppleScript Observations

- **Interesting "faux object" approach**
  - Applications are like object-oriented databases
  - No Proxies
- **Other features…**
  - Pioneered script management API (before WSH)
  - Recording
  - Attaching scripts to application objects
  - prototype-based object model
  - etc.

AppleScript Limitations

- **Apple Events**
  - Can't send multiple actions as one event
  - Hard for developers to create applications
  - Does not have a good model for retrieving objects
    - (more on this in a minute)

Databases

print name/manager of employees

whose

department name matches a prefix

&

salary is greater than limit

PL Viewpoint

```
foreach (Employee emp in DB.Employees() )
  if ( emp.Department.Name.startsWith(prefix)
       && emp.Salary >= limit )
    print( emp.Name + emp.Manager.Name );
```

Linear search

- **Orthogonal Persistence**
  - Automatically load objects as needed
    - "object faulting"
  - Approximations
    - Java Data Objects (JDO), EJB, Hibernate…
- **Optimization needed**
  - Even without latency issues

DB viewpoint

- **This is crazy!**
- **Databases already do this well…**
  - Choose algorithm (plan) based on
    - Structure of the query
    - Statistical properties of data
  - Orders of magnitude improvement
    - Program the way you want, let the system optimize

> "Whatever the database programming model, it must allow complex, data-intensive operations to be picked out of programs for execution by the storage manager, rather than forcing a record-at-a-time interface." – David Maier 1987

A Pragmatic Solution?

- **Complex dependencies between strings/API**
  - PL viewpoint: this is crazy!
- **Criteria in strings w/parameters (JDO style)**

```
Command q = new Query(Employee.class);
String paramDecl = "String prefix, int base";
String filter =
"emp.Department.Name.startsWith(prefix)"
          + " && emp.Salary >= base";
q.declareParameters( paramDecl );
q.setFilter( filter );
for ( Employee emp : q.execute(prefix, base) )
    print( emp.Name + emp.Manager.Name );
```


DLINQ

```
var empsAndMgrs =
  from emp in Employee
  join dept in Department
    on emp.DeptartmentID equals dept.ID
  where (dept.Name.StartsWith(prefix)
         && emp.Salary >= limit).
  join mgr in Employee
    on emp.ManagerID equals mgr.ID
  select new { emp.Name, MgrName = mgr.Name };
foreach (var result in empsAndMgrs)
    print(result.Name + result.MgrName );
```

emp.Department.Name

Artificial objects

Safe Queries (ICSE 2005)

```
for (Employee emp : db.query<Employee>(
  new Predicate<Employee> () {
    public boolean match(Employee emp) {
      return (emp.Department.Name.startsWith(prefix)
          && (emp.Salary >= limit); }}))
{
   print( emp.Name + emp.Manager.Name );
}
```

Function Boilerplate

Object Faulting

- Implemented by db4o for Java and C#
  - C# version uses delegates

Key Idea

Typical Database Operation has Two Aspects

1. **Find objects of interest**
   - Good: "Send the Work"
   - Query criteria shipped to database
2. **Do something with results (and related objects)**
   - Depends on what client *does*
   - DLINQ has "select" clause
     - Create result objects
     - Contain data that is needed by rest of the program
   - Safe Queries relies on dynamic object loading
   - Hibernate has **prefetch** capability

Hibernate Example

```
String q = "from Employee emp
            where e.overtime > 100";
for ( Employee emp : runQuery(q))
    print( emp.getName()
        + emp.getManager().getName()
    );
```

The rest of the program defines what is "needed"

Should **prefetch** managers

Manual Prefetch

```
String q = "from Employee emp
        left outer join fetch emp.Manager
        where e.overtime > 100";

for ( Employee emp : runQuery(q))
    print( emp.getName()
        + emp.getManager().getName() );
```

Subtle dependency…

… must be maintained if code changes

AutoFetch (ECOOP 2006)

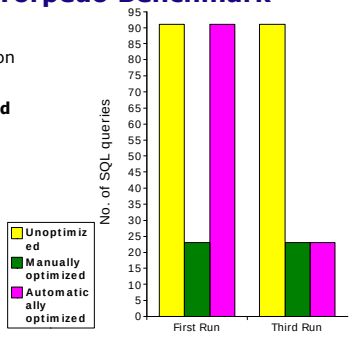
- **Prefetches objects based on past client behavior**
  - Classify queries as similar based on call stack
  - Collect statistics on way query results are used
  - Add prefetch specifications to similar queries
- **Prototype**
  - Extension of Hibernate
  - Could also be defined for DLINQ?

AutoFetch Torpedo Benchmark

Measures # of queries
Web auction application
17 use cases

AutoFetch is a fast as hand optimized

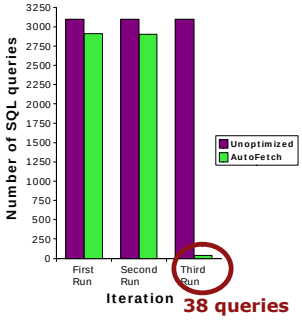
…with simpler code



AutoFetch OO7 Benchmark

- Measures: traversals, queries, and updates
  - Based on CAD applications
- AutoFetch reduces queries by **factor** of 100

T1 Traversal



38 queries

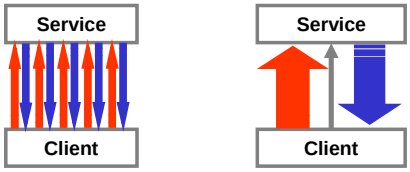
AutoFetch Discussion

- **Disadvantages:**
  - Does not optimize initial query executions
    - Related work: PrefetchGuide (cf Phil Bernstein)
- **Advantages:**
  - Best performance:
    - AutoFetch: 1 query
    - PrefetchGuide: at least 2 queries
  - Can prefetch arbitrary object graphs
  - More data for prediction

Web Services

Web Services

```
Container c;
c = root.Sub("papers").Sub("index.htm");
String s = c.Get().Text();
```



Code Becomes Data

- **Use Lazy Batched Futures (ICWS 2006)**

```
Request request = new Request();
File file1 = request.Sub("papers").Sub("index.htm") .Get();
File file2 = request.Sub("papers").Sub("picture.gif") .Get();
request.invoke();  // calls service, fills in file1 and file2
String s = file1.Text();
```

Results *must not* be needed before *invoke*

- **Web Service call includes both file requests**

```
<invokeRequest>
  <doc><name>papers</name>
    <item><name>index.htm</name><ID>1</ID></item>
    <item><name>picture.gif</name><ID>2</ID></item>
  </doc>
</invokeRequest>
```


Web Services

- **Programming model for Web Services**
  - Server publishes an "object model"
  - Client uses objects without per-method overhead
  - Similar to AppleScript
- **Better understanding of *Document-Oriented* style**
  - Document = description of work to be done
    - not just "data"
  - Server **interprets** the request
  - Request may also specify **prefetch**

Work in Progress

I Still Want It All

- **Write in object-oriented style**
  - No joins, funky query syntax, etc
- **Full optimization**
  - query optimization & prefetch

"The Holy Grail"
– Erik Meijer

```
foreach (Employee emp in DB.Employees() )
    if ( emp.Department.Name.startsWith(prefix)
         && emp.Salary >= limit )
      print( emp.Name + emp.Manager.Name );
```

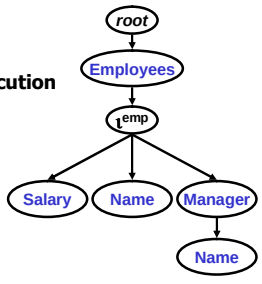

Query Extraction

1. **Path analysis by abstract interpretation**
   1. Basic paths
   2. Conditional paths
   3. Control vs. data dependence
2. **Query creation**
   - Condition promotion
3. **Program simplification**
   - Remove tests implied by query

Basic Path Analysis

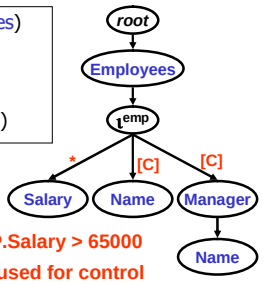
- **Abstract values by *paths***
  - ι is an iteration variable
- **Concretization = query execution**

```
foreach (emp in db.Employees)
  if (emp.Salary > 65000)
    print(emp.Name
      + emp.Manager.Name)
```



Conditional Path Analysis

```
foreach (emp in db.Employees)
  if (emp.Salary > 65000)
    print(emp.Name
      + emp.Manager.Name)
```

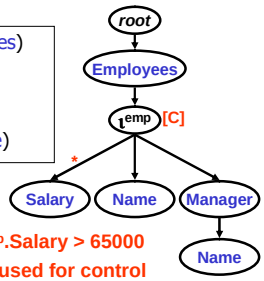


C = Employees.ι^{emp}.Salary > 65000

*** means path only used for control**

Condition Promotion

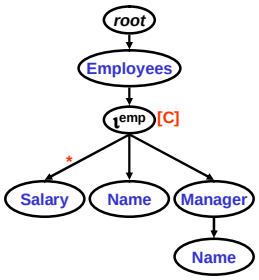
```
foreach (emp in db.Employees)
  if (emp.Salary > 65000)
    print(emp.Name
      + emp.Manager.Name)
```



C = Employees.ι^{emp}.Salary > 65000

* means path only used for control

Creating OQL Query




```
select struct (
  Name = emp.Name,
  Salary = emp.Salary,
  Manager = struct (
    Name = emp.Manager.Name))
from Employee as emp
where emp.Salary > 65000
```

C = Employees.ι^{emp}.Salary > 65000

* means path only used for control flow

Assemble and Simplify Program

```
foreach (emp in db.Employees)
  if (emp.Salary > 65000)
    print(emp.Name
      + emp.Manager.Name)
```

```
query =
  "select struct (
     Name = emp.Name,
     Salary = emp.Salary,
     Manager = Struct (
       Name = emp.Manager.Name))
    from Employee as emp
    where emp.Salary > 65000";
List result = session.createQuery(query);
for (Employee e : result.list())
  if (emp.Salary > 65000)
    print(e.Name +
          e.Manager.Name);
```


Issues with Query Extraction

- **Identifying where to extract queries**
  - May require sub-queries
- **Not yet handled**
  - Aggregation
  - Updates
- **Will it work in practice?**
  - Benchmarks are not good

Thoughts

- **AppleScript**
  - Action = Verb + Objects (as in English, not OO)
  - Sends works to server for execution
- **Distributed Objects versus Web Services**
  - PL challenge: batch operations into a single request
  - Not "what is possible" but "what is natural"
  - Latency is issue
- **AutoFetch**
  - Return all the data that is "needed"
  - By dynamic profiling of previous client behavior
  - Combined with DLINQ? Needs O/R mapping
- **Query Extraction**
  - Partition a program into a query and a residual program