Integrating Data & Services: Products & Challenges at BEA

Michael Carey
BEA Systems, Inc.
www.bea.com
Roadmap

- Background
- Data in the SOA World
- Declarative Data Services
- BEA AquaLogic Data Services Platform
  - Service Modeling
  - Read Services
  - Update Services
- Relevant Open Problems
- Conclusion
Roadmap

- Background
- Data in the SOA World
- Declarative Data Services
- BEA AquaLogic Data Services Platform
  - Service Modeling
  - Read Services
  - Update Services
- Relevant Open Problems
- Conclusion
Evolution of Database Systems

Files

CODASYL/IMS

Relational

Early DBMS Technologies
- Records and pointers
- Large, carefully tuned data access programs that have dependencies on physical access paths, indexes, etc.

Relational DB Systems
- Declarative approach
- Tables and views bring data independence
- Details left to system
- Designed to simplify data-centric application development

Manual Coding
- Byte or record streams
- Majority of application development effort goes toward building and maintaining data access logic
Relational Application Development

Department

\[ \text{dno}\ast, \text{name} \]

\[ 1 \text{ WorksIn} \]

N

Employee

\[ \text{eno}\ast, \text{name}, \text{salary} \]

Department

<table>
<thead>
<tr>
<th>dno</th>
<th>name</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>Toy</td>
</tr>
<tr>
<td>20</td>
<td>Shoe</td>
</tr>
</tbody>
</table>

Employee

<table>
<thead>
<tr>
<th>eno</th>
<th>name</th>
<th>salary</th>
<th>dept</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Lou</td>
<td>10000000</td>
<td>10</td>
</tr>
<tr>
<td>7</td>
<td>Laura</td>
<td>150000</td>
<td>20</td>
</tr>
<tr>
<td>22</td>
<td>Mike</td>
<td>80000</td>
<td>20</td>
</tr>
</tbody>
</table>

\[
\text{stmt} = \text{dbconn.prepareStatement (}
\quad \text{"select E.name, E.salary, D.no}
\quad \text{from Employee E, Department D}
\quad \text{where E.salary < 100000}
\quad \text{and D.name = ?}
\quad \text{and E.dept = D.dno"}
\quad \text{);}\
\]

...
Data Is *Everywhere* Now

- Perhaps relational databases made things too easy?
  - Departmental vs. inter-galactic centralized databases

- Databases come in many flavors
  - Relational: Oracle, DB2(s), SQL Server, MySQL, …
  - Hangers-on: IMS, IDMS, VSAM, …

- Not all important data is SQL-accessible
  - Packaged apps: SAP, PeopleSoft, Siebel, Oracle, SalesForce, …
  - Custom “homegrown” apps
  - Files of various shapes and sizes
  - And the list goes on…
Pain for Application Developers (Again)

- No one “single view of X” for any X
  - What data do I have about X?
  - How do I stitch together the info I need?
  - What else is X related to?

- Heterogeneity in multiple dimensions
  - \textit{Model heterogeneity}: disparate data models or data formats
  - \textit{API heterogeneity}: disparate data access and/or update APIs
  - \textit{Schema heterogeneity}: disparate representations for same info

- No reuse of developed artifacts
  - Different access criteria & returned data $\rightarrow$ different stitching plans
  - How would anyone even begin to find views/artifacts (w/o a model)?
Roadmap

- Background
- Data in the SOA World
- Declarative Data Services
- BEA AquaLogic Data Services Platform
  - Service Modeling
  - Read Services
  - Update Services
- Relevant Open Problems
- Conclusion
SOA To The Rescue?

- **Service-Oriented Architecture (SOA)**
  - Loosely-coupled interfaces (e.g., Web service contracts)
  - Each subsystem is a component with a service API
  - Create new assets by integrating & composing your existing assets!

- **We’re closer to dealing with heterogeneity**
  - Services all have XML Web service foundations
  - Custom logic hidden (e.g., data access and/or integration)

- **Fine …. but what about my data…?**
  - What are my business entities and how are they interrelated?
  - How can I find and/or share them, and what can they do?
  - And what about queries...?

→ SOA *what?*
Evolution of SOA Data Access

Coding

Manual Coding
- Java or C Programming
- Majority of application development effort goes toward building and maintaining data access logic

EAI

- Workflows and messages
- Large complex workflows that are cumbersome to build and maintain

DSP

DSP: Data Services
- Declarative approach
- Same basic principles as RDBMS
- Details left to system
- Designed for data service automation
Roadmap

- Background
- Data in the SOA World
- Declarative Data Services
- BEA AquaLogic Data Services Platform
  - Service Modeling
  - Read Services
  - Update Services
- Relevant Open Problems
- Conclusion
Seminal Work: *MultiBase* (Early 1980’s)

- One of the first DDBMS projects to relax the homogeneity assumption (vs. Ingres*, R*, ...)
  - Computer Corporation of America (CCA), funded by DoD

- Interesting foundation and technical contributions
  - Functional data model (i.e., “everything is a function”)
    - Prehistoric objects with identity
    - Functions model attribute access, relationship navigation
    - Model realized via the DAPLEX query language
  - Important technical achievements
    - Function-based model to normalize relational, network, and other models
    - Early results on federated query processing, sets/multisets, and more
    - This is why computer scientists need to study “history” – or not (😊)
# Declarative Integration via XML & XQuery

<table>
<thead>
<tr>
<th>Requirements</th>
<th>Standards</th>
</tr>
</thead>
<tbody>
<tr>
<td>A standard for data format and data interchange</td>
<td>XML</td>
</tr>
<tr>
<td>A standard for describing and modeling data</td>
<td>XML Schema</td>
</tr>
<tr>
<td>A standard for interfacing into applications</td>
<td>Web Services</td>
</tr>
<tr>
<td>A standard for querying both relational and non-relational data</td>
<td>XQuery</td>
</tr>
<tr>
<td>A standard Java programming model (read + write)</td>
<td>SDO (Service Data Objects)</td>
</tr>
<tr>
<td>A standard for publishing available services</td>
<td>Web Services</td>
</tr>
</tbody>
</table>
Roadmap

- Background
- Data in the SOA World
- Declarative Data Services
- BEA AquaLogic Data Services Platform
  - Service Modeling
  - Read Services
  - Update Services
- Relevant Open Problems
- Conclusion
Data Services in AquaLogic DSP

- Logical models capture data access and integration complexity **once**
- **Same** data model, programming model, and API for all enterprise data
AquaLogic DSP System Structure
Ex: Customer Profile Data Service

CUSTOMER, ORDER

CREDIT_CARD

getRating(…)

Customer Info

Order Info

Credit Card Info

Rating Info
Roadmap

- Background
- Data in the SOA World
- Declarative Data Services
- BEA AquaLogic Data Services Platform
  - Service Modeling
  - Read Services
  - Update Services
- Relevant Open Problems
- Conclusion
Service Modeling (Logical Services)
Service Modeling (Physical Services)
Data Service – Design Overview
Roadmap

- Background
- Data in the SOA World
- Declarative Data Services
- BEA AquaLogic Data Services Platform
  - Service Modeling
  - Read Services
  - Update Services
- Relevant Open Problems
- Conclusion
Graphical Service Creation
Data Service – “Get All” Read Method

(::pragma function ... kind="read" ...::)

declare function tns:getProfile() as element(ns0:PROFILE)*
{
  for $CUSTOMER in db1:CUSTOMER()
  return
  <tns:PROFILE>
    <CID>{ fn:data($CUSTOMER/CID) }</CID>
    <LAST_NAME>{ fn:data($CUSTOMER/LAST_NAME) }</LAST_NAME>
    <ORDERS>{ db1:getORDER($CUSTOMER) }</ORDERS>
    <CREDIT_CARDS>{
      db2:CREDIT_CARD()[CID eq $CUSTOMER/CID]
    }</CREDIT_CARDS>
    <RATING>{
      fn:data(ws1:getRating{
        <ns5:getRating>
          <ns5:lName>{ data($CUSTOMER/LAST_NAME) }</ns5:lName>
          <ns5:ssn>{ data($CUSTOMER/SSN) }</ns5:ssn>
        </ns5:getRating>
      }
    }
  </tns:PROFILE>
ALDSP Query Processing Overview

- Compile-time function composition
  - Similar to RDBMS view rewriting & unnesting
  - Facilitates pushdown, eliminates irrelevant data sources, …
    → *It’s what makes data services reusable!!*

- Joins and related operations
  - *Goal:* Let each RDBMS do what it does best → *maximize SQL pushdown*
  - Outerjoins, presorted grouping, sorting pushdown, function pushdown, …
  - Batched parameter-passing for streamed distributed join processing

- Runtime system
  - Streamable operators based on XML TokenIterator model

- Miscellaneous anti-latency weapons
  - `async(exp), failover(exp1,exp2), timeout(exp1,t,exp2)`
Query Processing, Example 1 (getProfile)
Query Processing, Example 2 (query getProfile)

```
declare namespace xsn="http://DemoSources/PROFILE";
for $p in xsn:getProfile()
  where $p/ID eq "CUSTOMER00000001"
  return $p
<$PROFILE>
  <$LAST_NAME>$p/LAST_NAME</LAST_NAME>
  <$RATING>$p/RATING</RATING>
</PROFILE>
```
Roadmap

- Background
- Data in the SOA World
- Declarative Data Services
- BEA AquaLogic Data Services Platform
  - Service Modeling
  - Read Services
  - Update Services
- Relevant Open Problems
- Conclusion
What About Updates?

- So far we have focused on read services
  - Declaratively specified using XQuery
  - System selects efficient implementation

- Obviously need write services as well
  - Automation through lineage analysis of read services
  - Full automation possible for SQL-based data services
  - Update overrides required for Web services (non-SQL sources)

- What programming model for writes?
  - Disconnected model is highly desirable
  - Want flexible optimistic concurrency options
  - *Answer:* SDO from IBM, BEA, Oracle, SAP, and others
SDO API & Change Tracking

//Get SDO
CustomerDoc custSDO = CustomerDS.getCustomerById("007");

// Make changes to SDO
custSDO.setCustName("Mike");
custSDO.setEmail("mcarey@bea.com");

//Submit SDO
CustomerDS.submit(custSDO);
Update Decomposition

Update Framework

- XA and non-XA sources
- Automated change decomposition
- Automatic SQL generation for RDBMS
- Update “hooks” for business validations, replacement logic, or compensation logic (e.g., via a workflow)
Concurrency (RDBMS Sources)

- Based on optimistic concurrency control
  - Before values are compared to current database values
  - *Ex:* `update CUSTOMER set FIRST_NAME=?`  
    `where CUSTOMER_ID=? and FIRST_NAME=?`

- Comparison (consistency) options include
  - All updated fields
  - All read or updated fields
  - Designated field or fields (e.g., timestamp or version id)

- Benefits of this approach
  - Stateless and therefore scalable
  - Natural fit for Web apps and services
Roadmap

- Background
- Data in the SOA World
- Declarative Data Services
- BEA AquaLogic Data Services Platform
  - Service Modeling
  - Read Services
  - Update Services
- Relevant Open Problems
- Conclusion
Semantic Query Optimization
(The 1980’s Strike Again!)

Ex: Order management Web service API:

- oms:getOrderHistory(cid)
- oms:getOpenOrders(cid)
- oms:getRecentOrders(cid, startDate)

How might we define a single Customer/Order view that integrates information from this Web service as well as other sources?

```
declare function vns:getCustInfo() as element(vns:CustInfo)*
{
    for $CUST in db1:CUSTOMER() return
        <vns:CustInfo>
            <Id>{ fn:data($CUST/CID) }</Id>
            <Name>{ fn:data($CUST/LNAME) }</Name>
            { oms:getOrderHistory($CUST/CID) }
        </vns:CustInfo>
};
```
Semantic Query Optimization
(Continued)

Imagine a set of equivalence rules:

- \(\text{oms:getOpenOrders}(\text{cid}) == \text{oms:getOrderHistory}(\$\text{cid}) [\text{status eq ‘open’}]\)
- \(\text{oms:getRecentOrders}(\text{cid}, \text{start}D\text{ate}) == \text{oms:getOrderHistory}(\$\text{cid}) [\text{orderDate ge $start\text{Date}}]\)

Such rules could then enable query rewriting:

```
for $c$ in vns:getCustInfo(),
  $o$ in $c$/Orders/Order
where $c$/Id eq $cid$ and $o/status$ eq ‘open’
return $o$
```

```
for $CUST$ in db1:CUSTOMER(),
  $o$ in oms:getOpenOrders($CUST/CID)/Order
where $CUST/CID$ eq $cid$
return $o$
```
Dynamic Query Processing
(Over Services + Data)

Ex: Consider a data service query that gets information from a mixture of internal and external services:

```
for $cust in crm:GetRecentCustomers($sinceDate)
let $score1 := exper:getCreditScore($cust/SSN),
    $score2 := equi:getCreditScore($cust/SSN),
    $score := min($score1,$score2)
where $cust/Address/State eq 'Maine' and $score gt 675
return
  <TargetCustomer>
    <Name>{data($cust/NAME)}</Name>
    <Rating>{$score}</Rating>
  </TargetCustomer>
```

What’s the “best” way to evaluate such a query?

- Should the credit score inquiries be parallelized for a customer?
- Should the credit score inquiries be parallelized across customers?
- How selective are the various predicates and function arguments?
- How large (in various dimensions) are the intermediate results? (Etc.!)
Sagas and Compensation
(Approximating WS Transactions)

Web services don’t understand “CRUD”
- Enable definition of insert, delete, & update operations

Web services don’t speak 2PC
- Use sagas to provide recovery atomicity
- Enable registration of undo’s and did-I-do’s
- Use compensation to accomplish rollback

Failures may lead to indeterminate states
- Reality bites!
- Need to record such failures and provide APIs to enable app-level discovery & repair work
Semantically-Assisted Schema Integration (Or “Pick me, pick me!”)
Semantically-Assisted Schema Integration (Continued)
Roadmap

- Background
- Data in the SOA World
- Declarative Data Services
- BEA AquaLogic Data Services Platform
  - Service Modeling
  - Read Services
  - Update Services
- Relevant Open Problems
- Conclusion
In Summary

- **Challenges in the Brave New World**
  - From databases (then) to *data services* (now)

- **Simplify data service development**
  - Data-oriented modeling and thinking can contribute to SOA
  - XQuery + XML Schema → *declarative* data services
  - Java / WS APIs + SDO → automating updates as well as reads

- **Interesting open “systems” problems**
  - Semantic query optimization (revisited)
  - Dynamic query processing over data + services
  - Sagas and compensation for Web services
  - Semantically-assisted schema integration