Service Data Objects

André Tost
Senior Technical Staff Member
IBM Software Group
atost@us.ibm.com
Agenda

- SDO Definition/Goals
- Architecture
- Topologies
- Detailed features
  - DataObject API
    - Dynamic
    - Static
  - DataGraph
  - Others
- Example Scenario
- Outlook
- Demo
### Definition

**Service Data Objects (SDO)**

"Service data objects is a specification for a programming model that unifies data programming across data source types, provides robust support for common application patterns, and enables applications, tools, and frameworks to more easily query, view, bind, update, and introspect data."

*Next Generation Data Programming: Service Data Objects*, Beatty, Brodsky, Nally, Patel

#### Key messages:
1. Programming model specification
2. Unifies data programming across disparate data sources
3. Enables standard application development patterns
4. Enables tools and frameworks to be built to the consistent data model

#### Joint specification(s) published by IBM and BEA
- Version 1.0 published in November 2003
- Version 2.0 published in June 2005

#### Submitted to Java Community Process as JSR 235
Summary of Goals

1. Provide unified and consistent data access to heterogeneous data sources
   - Simplified programming model for the application programmer
   - Enable tools and frameworks to work consistently across heterogeneous data sources
   - Result: Dramatic simplification of the J2EE Programming Model

2. Provide robust programming model support for several J2EE best practice application patterns
   -Disconnected programming model
   - Custom data access layers based on common design patterns

3. Provide first class support for XML Schema, XML InfoSet, and XML data sources
   - XML/Java bindings (JAXB like capability)
   - JAX-RPC objects
# Data and Metadata APIs

<table>
<thead>
<tr>
<th>Model</th>
<th>Model</th>
<th>API</th>
<th>Data Source</th>
<th>MetaData API</th>
<th>Query Language</th>
</tr>
</thead>
<tbody>
<tr>
<td>SDO</td>
<td></td>
<td>Both</td>
<td>Any</td>
<td>SDO Metadata API, Java Introspection</td>
<td>Any</td>
</tr>
<tr>
<td>JDBC Rowset</td>
<td></td>
<td>Dynamic</td>
<td>Relational</td>
<td>Relational</td>
<td>SQL</td>
</tr>
<tr>
<td>JDBC Cached Rowset</td>
<td></td>
<td>Dynamic</td>
<td>Relational</td>
<td>Relational</td>
<td>SQL</td>
</tr>
<tr>
<td>Entity EJB</td>
<td></td>
<td>Static</td>
<td>Relational</td>
<td>Java Introspection</td>
<td>EJBQL</td>
</tr>
<tr>
<td>JDO</td>
<td></td>
<td>Static</td>
<td>Relational, Object</td>
<td>Java Introspection</td>
<td>JDOQL</td>
</tr>
<tr>
<td>JCA</td>
<td></td>
<td>Dynamic</td>
<td>Record-based</td>
<td>Undefined</td>
<td>Undefined</td>
</tr>
<tr>
<td>DOM and SAX</td>
<td></td>
<td>Dynamic</td>
<td>XML</td>
<td>XML InfoSet</td>
<td>XPath, XQuery</td>
</tr>
<tr>
<td>JAXB</td>
<td></td>
<td>Static</td>
<td>XML</td>
<td>Java Introspection</td>
<td>N/A</td>
</tr>
<tr>
<td>JAX-RPC</td>
<td></td>
<td>Static</td>
<td>XML</td>
<td>Java Introspection</td>
<td>N/A</td>
</tr>
</tbody>
</table>
Programming Model First Principles
“Triangle of Truth”
Triangle of Truth

Tomorrow...

Composition
BPEL

Data
Service Data Objects

Invocation
Service Components
SDO Roles and Dependencies

Business programmer role

Portable
System

Provider role

Business application

SDO APIs ← Service APIs

SDO Impl ← Service Impl

Middleware

dependency

System programmer role
Existing J2EE Architecture

- Data Access APIs
- Data APIs
- Metadata Access APIs
- Metadata APIs
- Client
- RDB
- EIS
- XML/XML Schema
SDO Architecture

Mediator Pattern

Client → SDO

- Data Access APIs
  - Data APIs
  - Metadata Access APIs
  - Metadata APIs

EIS

RDB

XML/XML Schema
Service Data Object Runtime Architecture

Service Data Objects

Metadata Model

Data Model

Pluggable Data Mediators

Data Access APIs

Data APIs

Metadata Access APIs

Metadata APIs

Client

RDB

EIS

XML/XML Schema
Data Mediator Services

- Responsibilities
  - Query data source
  - Creating graphs of data containing data objects
  - Looks to see if concurrency control was violated
  - Applies data graph changes back to the data source
Topology 1

- Client talks directly to an in-process mediator
Client talks to a service, in this case a stateless session bean.

The stateless session bean(s) talk to the mediator to access the data source.

In this case, the mediator goes remotely to a data source.
Topology 3

- Client talks to a service, in this case a stateless session bean
- The stateless session bean(s) talk to the mediator to access the data source
  (Note: Although the persistence form changes dramatically from Topology 2, the
  source code in the session bean(s) changes very little – the Mediator is acting as
  a data facade)
- In this case, the mediator is mediating access to Entity EJBs
- The WebSphere Entity EJB Container manages the Object/Relational mapping of
  the Entity to the data source
Topology 4

- Client talks to a JAX-RPC based web service passing a DataGraph/DataObjects.
- The web service does its processing and returns DataGraph/DataObjects back to the client.
- It is quite possible that the black box looks like Topology 2 or 3 above, where the web service is assuming the same role as the session bean.
Topology 5

- Mediator potentially running as a stored procedure
- Returns DataGraph/DataObjects back to the client
- Takes DataGraph/DataObjects as parameters
SDO features

- Dynamic Data API
- Static Data API
- Rich Data Objects
- Change Summary
- Navigation through graphs of data
- Metadata
- Relationship integrity
- XML and XML schema support
SDO Design and Runtime Components

1. SDO Core
   - Data Objects
   - Data Graphs
   - Introspection APIs

2. SDO Data Mediator Services
   - Query back end data source
   - Create data graphs
   - Manage optimistic concurrency

3. SDO Tools
   - Code generators
   - Metamodel converters
   - Schema converters
   - Data modeling tools
   - Schema modeling tools

4. SDO Enabled Runtimes and Frameworks
   - Data binding to UI
SDO Core UML Model
DataObjects (Page 1 of 2)

- Next generation JavaBeans
- Purpose is as a source independent data container
  - Primitives (Java/XML Schema like primitive types)
  - References to other data objects
- Not a provider of business logic methods

Examples:
- XML Schema
  DataObject would represent a complex type, with attributes being represented as primitives, and child complex type elements represented as references
- Relational Database
  DataObject might represent a row of data
Metadata introspection capabilities
- Enables access to types, relationships, and constraints
- Metadata can be generated from XML Schema, Java interfaces, XMI, et al

Dynamic interface or you can generate a statically-typed interface from metadata

Rich relationship integrity management
- Supports 1:1, 1:n, and n:m relationships
- Auto-manages inverse relationships
- Supports containment and reference semantics

Event management facilities

XML friendly: supports XML Schema for metadata and XML as a data source, and supports XPath expressions to get/set values (looking at XQuery)
DataObject

- get(Property)
- set(Property)
- isSet(Property)
- unset(Property)
- create(Property)
- delete()
DataObject Properties

- Properties by String, int, Property, XPath
  - `get("address")`
  - `get(1)`
  - `get(address)`
  - `get("address/zip")`
DataObject Typed Accessors

- `getXXX(propety)`. XXX is:
  - primitives: `int`, `float`, `boolean`, `byte[]`, ...
  - `String`
  - `BigDecimal`, `BigInteger`
  - `Date`
  - List for multi-valued properties
  - converts between primitives and Objects
  - converts between data types
    - `getInt("width")` of 5.123 returns 5
Dynamic Data Object APIs

String getString(String path);
String getString(int fieldIndex);
void setString(String path, String value);
void setString(int fieldIndex, String value);

int getInt(String path);
int getInt(int fieldIndex);
void setInt(String path, int value);
void setInt(int fieldIndex, int value);

Object getObject(String path);
Object getObject(int fieldIndex);
void setObject(String path, Object value);
void setObject(int fieldIndex, Object value);

DataObject getDataObject(String path);
DataObject getDataObject(int fieldIndex);
void setDataObject(String path, DataObject row);
void setDataObject(int fieldIndex, DataObject row);

List getList(String path);
List getList(int fieldIndex);

And so on for all the “standard” types
DataObject Tree

- `getDataGraph()`
- `getContainer()`
  - `employee.getContainer()` returns department
- `getContainmentProperty()`
  - `employee.getContainmentProperty()` returns employees
- DataObjects form a tree with containment
- A DO can have one containing DataObject
- If a DO is added to one containment it is removed from any previous one
- Containment property is how the container holds this DO.
import java.util.Date;

public interface Book extends DataObject {
    ...
    Date getPublicationDate();
    void setPublicationDate(Date value);

    String getTitle();
    void setTitle(String value);

    int getPages();
    void setPages(int value);
}
Example 1 – Updating DataObjects

```java
// create a new employee
DataObject newEmployee = department.createDataObject("employees");
newEmployee.set("name", "Al Smith");
newEmployee.set("SN", "0004");
newEmployee.setBoolean("manager", true);

// Set employeeOfTheMonth to be the new employee
company.set("employeeOfTheMonth", newEmployee);
```

```xml
<company name="MegaCorp" employeeOfTheMonth="0004">
  <departments name="Advanced Technologies" location="NY" number="123">
    <employees name="John Jones" SN="0001"/>
    <employees name="Jane Doe" SN="0003"/>
    <employees name="Al Smith" SN="0004" manager="true"/>
  </departments>
</company>
```
Example 2 – Accessing DataObjects

// Get an employee using an xpath-like expression
// starting from the company
DataObject employee =
    company.getDataObject("departments[number=123]/employees[SN=0002]");

// Or, an xpath-like expression can find the employee
// based on positions in lists:
DataObject employee = company.getDataObject("departments.0/employees.1");

// Or, use the API to go step by step to find the employee
// Get the list of departments starting from the company
List departments = company.getList("departments");
// Get the department at index 0 on the list
DataObject department = (DataObject) departments.get(0);
// Get the list of employees for the department
List employees = department.getList("employees");
// Get the employee at index 1 on the list
DataObject employeeFromList = (DataObject) employees.get(1);
DataObject create()

- `create(property)`
  - creates a new object and adds to the containment property
  - uses the type of the property
  - property is a container
  - `department.create("employees")`

- `create(property, type)`
  - creates a new object of the specified type
DataObject delete()

- Restores to the state at the time object was created
- Unsets all the properties
- Unsets properties of all the contained objects
- Removes from container
- Bidirectional properties unset, 1-way in unchanged, 1-way out unset
- Object may be used again, i.e. a "Reset"
DataObject isSet(), unset()

- **isSet(Property)**
  - true if the value is different than its default
    - get("year") returns 2004, default is 2000, isSet("year") true
  - if the property is a setting, true if set to any value (sequence)
    - `<letter><from/><to/><year>2000</year></letter>`
    - isSet("year") true for year as a setting

- **unset(Property)**
  - restores to default
  - isSet always false after unset
  - same state as when DO was created
    - set("year", 2004); unset("year"); get("year") is 2000
Sequence

- Orders values across properties
- Text that may be mixed with properties
- Unstructured business data
- XML data for mixed content or open content
  
  ```xml
  <letter><to/>Hello<name>John</name><from/>
  </letter>
  ```
Sequence API

- getValue(index)
- getProperty(index)
- setValue(index, value)

- add(property, value)
- add(index, property, value)
- remove(index)
- move(index, index)
DataGraph

- Is an “envelope” object
- Contains a single DataObject
- References the schema for the DataObjects
- Records change summary information accessible by mediators to provide optimistic concurrency control semantics
- Flows as an XML Message (e.g. Datagraph.xsd)
- Four key components to the message on the wire:
  1. Schema
  2. Data
  3. Data change summary
  4. Faults

Data Mediator Service is responsible for filling DataGraph with DataObjects from data source, updating data source from DataObject changes
DataGraph API

- `DataObject getRootObject()`
- `DataObject createRootObject()`

- `ChangeSummary getChangeSummary()`

- `Type getType(String uri, String typeName)`
ChangeSummary

- Return the original values of the DataGraph
- Advice to services:
  - Old values allow optimistic concurrency
  - Services only update changed items
  - If an old value was not set, treat as unknown
ChangeSummary API

- `beginLogging()`, `endLogging()`, `isLogging()`
- `getDataGraph()`
- `List getChangedObjects()`
- `isCreated(DataObject)`, `isDeleted(DataObject)`
- Considering: `getOldContainer()`, `getOldContainmentProperty()`
ChangeSummary API 2

- List /*Setting*/ getOldValues(DataObject)
- Setting
  - getOldValue()
  - getProperty()
  - isSet()
ChangeSummary Example

- `get("firstName")` is "Adam"
- `get("lastName")` is null (default)
- `set("firstName", "Charlie")`
- `set("lastName", "Smith")`
- ChangeHistory settings:
  - `property=firstName, oldValue=Adam, isSet=true`
  - `property=lastName, oldValue=null, isSet=false`
Metadata

- Type and Property provide simple access to advanced metadata
- Metadata faithful to original source
  - XSD, EMOF, Relational, Java, C, COBOL, ...
- Metadata driven system software needs access to the original source
Type API

- getName()
- getInstanceClass()
- isInstance(Object)
- getProperties()
- getProperty(String propertyName)
- String getURI()
Property

- Properties have types
- 1, N, 1-1, 1-N, N-M multiplicity
- Unidirectional and bidirectional
- Containment semantics
- Sequence semantics
- Defaults
- In progress: read-only *aka* immutable
Property API

- getName()
- getType()
- getContainingType()
- isContainment()
- isMany()
Sample Scenario – Viewing/Updating an Insurance Policy
Traditional Best Practices using Connected/Disconnected Data

1. Click URL to view insurance policy

2. Make request to retrieve data required by view
   Pass any parameters as data objects

3. Initiate transaction, security check
   Interrogate EJB entity model
   Perform transactional processing
   Create and fill disconnected data objects
   Return disconnected data objects

4. Format data objects
   Scheme: JSP Tags, Scripting, or a data object aware formatting framework

5. Modify the policy (thus mutating the data objects)
   Submit changes

6. Pass data objects and deltas back to the transactional server

7. Update EJB entity model with deltas

Client
HTML

Controller/View
Servlet/JSP

Business Logic
Session Beans

Model
Entity Beans

Persistence
RDB
Disconnected Programming Model

- Insurance policy view/update scenario demonstrates a traditional web based disconnected model:
  1. Client makes policy view request
  2. Controller requests policy data from transactional server
  3. Server starts transaction
  4. Server retrieves data from transactional resource
  5. Server copies data into non-transactional disconnected data objects
  6. Server commits transaction
  7. Controller combines data objects with render objects (widgets, tags, et al) to produce client view
  8. Client updates non-transactional disconnected data objects, submits changes
  9. Controller delegates changes to the transactional server
  10. Server starts transaction
  11. Server validates data concurrency semantics (see OCC)
  12. Server persists changes back to the transactional resource
  13. Server commits transaction

- Data is “checked out” of the data store for some “period” of time (but not locked)
  - Possibility exists that the data might become stale
  - Possibility exists that someone else might change the same data elsewhere

- The disconnect data scenario applies to several other enterprise architecture use models:
  - Offline mode (Lotus Notes replication semantics, PDA synchronization, et al)
  - B2B SCM/PRM – Company A obtains data from company B, modifies, returns updated data
Optimistic Concurrency Control (OCC)

Popular strategy

- Scenario:
  My insurance agent and I are updating my insurance policy concurrently

- The optimistic concurrency control strategy supports this use model
  - Increased concurrency
  - Increased throughput

- Collision detection strategy
  - Disconnected data objects maintain primary key, old value, and new value
  - When disconnected data object changes are written to the database, the old value is first checked to make sure it is still the same, before the new value is applied
  - If the old value in the DataObject differs from the current value in the transactional data store, an error is thrown to the application, otherwise the update is completed

- Note: there are multiple strategies for managing OCC
Several Best Practice Patterns and Strategies

- Separation of concerns between data fetch and data render logic
- Session façade pattern with EJB Entity CMP to represent business model
- Actions aggregated into a single transaction providing ACID reads/updates
- Transfer objects used between view/controller and the transactional tier
- Application of data render objects to transfer objects for efficient and reusable user interface programming
- Server transaction and connection resource utilization minimized
- Data Object change summary used to optimize transactional update operation
- Optimistic concurrency pattern leveraged to enable high transaction throughput

Note: Items affected by SDO are in the Blue font
Applying SDO to the Insurance View/Update Scenario

- Transfer objects used between view/controller and the transactional tier
  - The transfer objects could use the rich DataGraph/DataObject data types
  - The view/controller could communicate directly with an Entity CMP mediator in order to obtain the disconnected data. If processed needed to occur in addition to the data fetch – the session bean could use the Entity CMP mediator.

- Application of data render objects to transfer objects for efficient and reusable user interface programming
  - SDO enables frameworks like JSF to bind their widget set to a single unified data representation format
  - Since SDO is likely to enable constrain frameworks, those can be leveraged by the user interface componentry to provide client side cascading delete semantics, field uniqueness, read-only semantics, et al

- Disconnected and OCC programming
  - The change set automatically maintained by the DataGraph enables the data mediators to provide optimistic concurrency control for the application programmer
  - Maintaining change information on the client enables the application to only send back the change sets to the server optimizing network bandwidth usage

- How much of the model would have to change if the data source changed from Relational to IMS (or vice versa)?
  - Leveraging the mediator pattern and the unified DataGraph/DataObject representation scheme minimizes the ripple effect resulting from either an data access API or data representation change

- Although not mentioned above, SDO provides several application server caching opportunities to further enhance application performance and scalability
Outlook – SDO 2

- Published June 2005
- Purely additions to SDO 1.0
- No deprecations or changes to existing function
- XML/XSD support
- Java code generation
- SDO Model
- Helpers (factory, copy, equality, type, ...)
- More methods on existing interfaces
- More ways to have a change summary
- Basic/open/sequenced DataObjects
Eclipse SDO Implementation

1. Download & install a Java RTE/SDK v1.4

2. Download the Eclipse SDK v3.1 (Eclipse platform, Java development tools, Plug-in development environment)
   2. Select “downloads”
   3. Select “Eclipse SDK 3.1”

3. Download EMF v2.1.0
   2. Select “Downloads”
   3. Download “emf-sdo-xsd-SDK-2.1.0.zip”

4. Install Eclipse SDK v3.1, and EMF v2.1.0
   1. Unzip eclipse-SDK-3.1.0-win32.zip in /c:/eclipse31 (or wherever you install software)
   2. Unzip the EMF zip file into the directory where you installed Eclipse SDK v3.1
   3. Run eclipse.exe from the directory where you installed Eclipse SDK v3.1

5. Run the tutorial “Using the SDO DataGraph editor”
Resources

- **SDO 1 Specification**

- **SDO 2 Specification**

- **Article list**

- **Eclipse EMF SDO Implementation**