IBM Software Group

IBM® WebSphere® Application Server V6

Service Data Objects (SDO)

@business on demand.

Updated January 25, 2005

This presentation, will focus on providing an overview of Service Data Objects.

Goals

- Provide an overview of Service Data Objects (SDO)
- Highlight the benefits of using SDO
- Provide an overview of SDO Core APIs

The goals for this presentation are to provide an overview of Service Data Objects and to highlight the benefits of using SDO.

Agenda

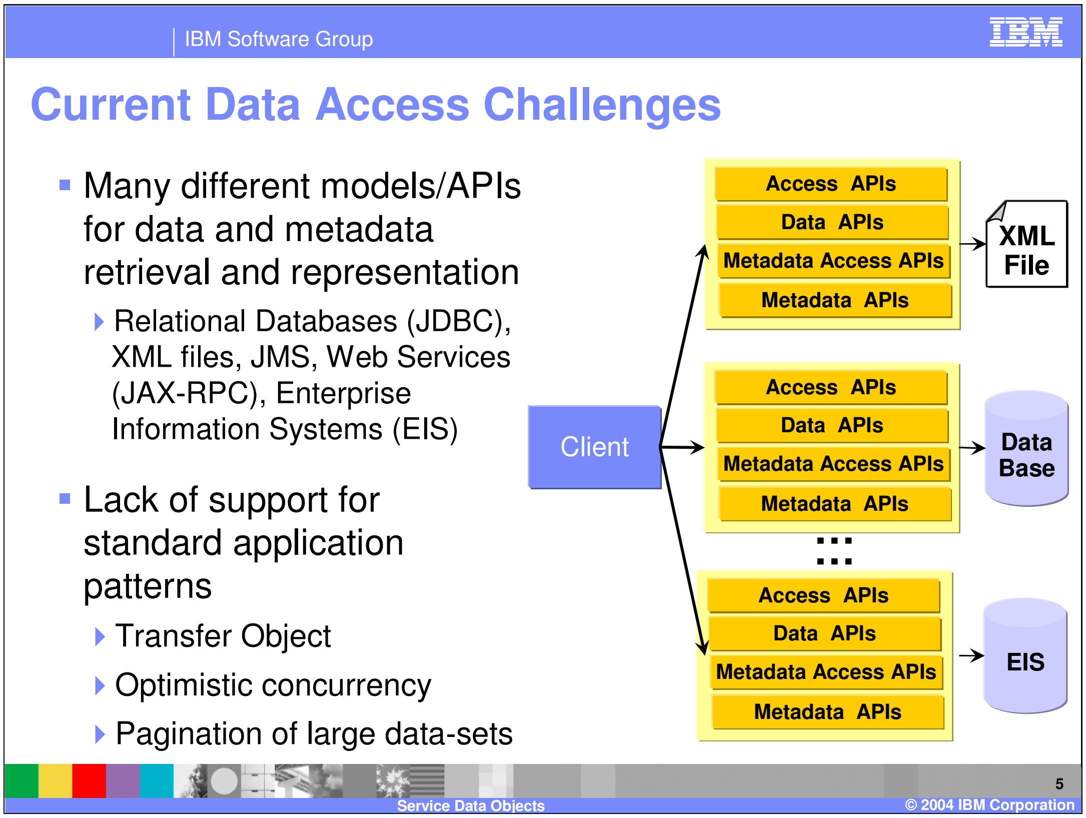
- Overview
  - Current problem with access to various data sources
  - Service Data Objects (SDO)
- SDO Architecture
- Summary and References

The agenda for this presentation is to start by looking at an overview of the motivation and goals of the SDO architecture. The second part of the presentation will focus on providing an overview of the SDO architecture.

Section

Overview

The next section will provide an overview of the SDO architecture.



The Java™ 2 Enterprise Edition (J2EE) platform and supporting JSRs provide a wide range of programming models and APIs. Because of this, J2EE developers are faced with a wide range of data model and access APIs to represent and retrieve data from a variety of backend data sources. This situation requires that developers become experts in many different data access technologies. Furthermore, not all of these technologies provide functionality that is rich enough to easily support many of the standard application patterns encountered in a typical J2EE application. Some examples of common application patterns that lack standard support are:

Transfer Object Pattern: This pattern is used to minimize the amount of network traffic that results from making several calls to remote methods for each attribute that is needed by a client. Instead of making several access calls, an object called a transfer object can be returned from a single remote call. This transfer object encapsulates the set of attributes needed by the client.

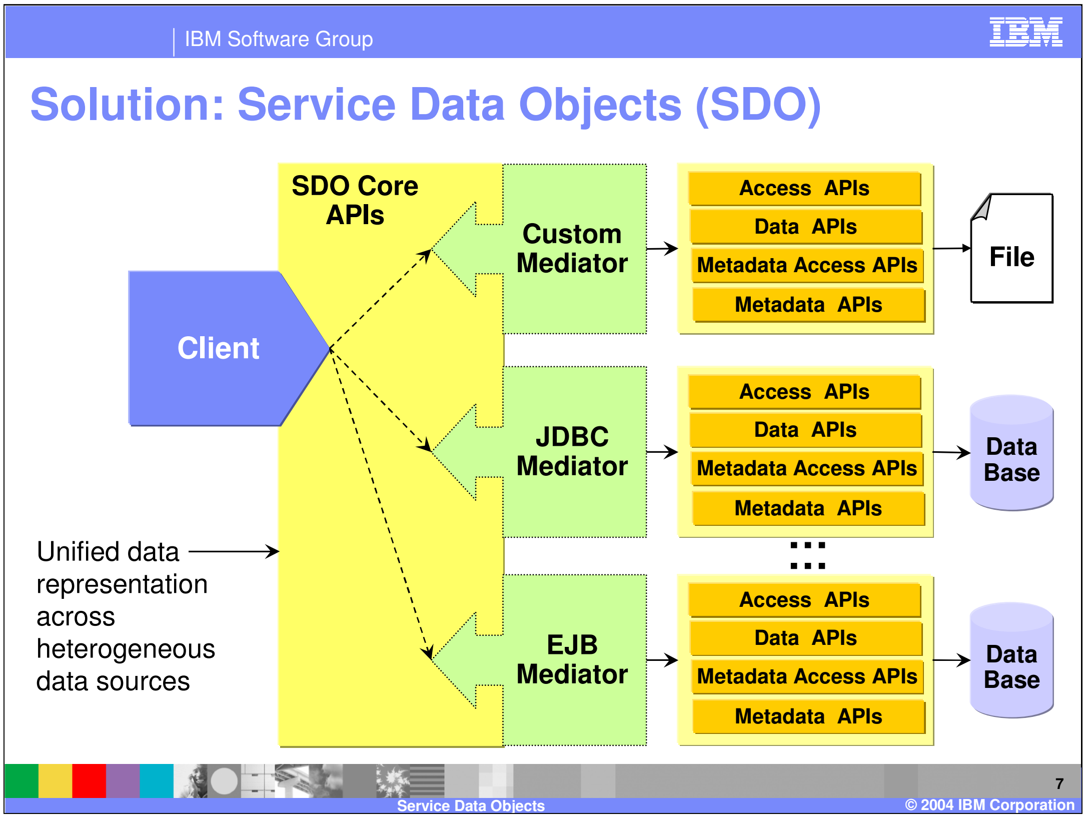
Optimistic Concurrency: With a disconnected programming model (such as a typical web-based application), it is not uncommon to employ an optimistic concurrency approach. The primary assumption with optimistic concurrency control is that data conflicts are infrequent.

Pagination of large datasets: Anyone who has developed an application that queries a back end data store for information knows that an important problem that needs to be addressed is how to page through a large set of data. Almost every application that queries a data store needs to know how to page through the data (particularly when data is displayed on a GUI) when the result set of the query is large.

Consequences of Current Data Access Challenges

- Programmers focus on learning technologies, rather than solving business problems
- Programmers do a lot of low-level coding
- Tools do not offer an easy development experience for J2EE application developers

The fact that there is a wide variety of APIs and data models used to access data in a J2EE application creates a situation where programmers spend a fair amount of time learning data access technologies rather than focusing on solving the business problem at hand. Combined with this the fact that there is a lack of standardized support for common application patterns, application developers typically need to do a fair amount of low level programming to develop their business applications. It should not be surprising that these same data access challenges also make it difficult for tool providers to build development environments that make it easy to build J2EE applications.



SDO was developed to address many of the current data access challenges. The primary goal of the SDO architecture is to make it easier for application and tools developers to create, view, update, and delete data that is stored in a variety of backend data stores. As has already been discussed, one of the reasons this is currently a challenge is that there are a wide variety of APIs and data models that are commonly used for J2EE application development. The SDO architecture addresses this problem by providing uniform data access and representation across a wide variety of data sources as well as support for many common application patterns that are encountered in J2EE application development. The intention is to decrease the amount of low level code developers need to write in order to create an application, and instead focus on solving the business problem.

The SDO architecture can be thought of to have two primary parts:

(1) The core set of SDO APIs that define the unified data representation model

(2) Data mediator services (DMS) used to access the data

The picture shown on this slide is intended to emphasize that both the client code and the mediator code need to know about the SDO core APIs. Basically, the core APIs are used by the client and the mediator to represent data in a uniform manner. The client will use this core API irrespective of the mediator that is being used.

NOTE: The current proposal for the SDO architecture (JSR 235) only addresses the APIs that are used to uniformly represent the data. At this time a mediator specification is not included. For this reason the client still has to work with a very small set of Mediator APIs. Fortunately the data source-specific code for the mediator needed consists of steps needed to initialize a particular mediator.

SDO: Design Points

- Unified data access and representation across heterogeneous data stores
- SDO supports
  - Dynamic and static (strongly typed) data APIs
  - Disconnected programming model
  - Introspection of data
  - Change history for data modifications
  - Relationship integrity

As discussed on the previous slide, SDO is intended to provide unified data access and representation across heterogeneous data sources. However, there are also several other significant SDO design points to mention.

Static and Dynamic Data APIs – One of the SDO requirements is that it support both dynamic and static (typed) data APIs. A dynamic API would typically be needed when the structure of the data is not known until runtime. An example of a dynamic data API is:

```
dataObject.set("LASTNAME", "Smith");
```

On the other hand, static data APIs can be easier to work with and represent a familiar programming model for most application developers. The following is an example of this:

```
public interface Person {
  public String getLastName();
  public void setLastName(String name);
  …
}
```

In the client code, the Person object would be used in the following manner:

```
Person p = …
p.setLastName("Smith");
```

The core SDO APIs provide the dynamic data API. Support for static data APIs comes in the form of code generation tools. However, you should note that the code generation specification is not currently in the scope of the SDO proposal.

Disconnected Programming Model – A typical web based application uses a disconnected data access pattern. For example, consider a client request to access a particular web-based GUI. When a web client accesses a page that contains dynamic content, the follow scenario is not uncommon: (1) Client requests to view a form of data, (2) Servlet or JSP queries/requests data from a data source (Start new transaction, read data, end transaction), and then renders page data with the information, (3) Client updates data on the form and submits updates to backend, and (4) Servlet or JSP updates data source based on data submitted (Start new transaction, update data, end transaction). The SDO model has been designed to support this disconnected data access model. Typically with this type of data access pattern an optimistic concurrency approach is utilized.

SDO provides a number of features that are built into the core APIs that are included to address several common data access scenarios. First, SDO provides APIs for metadata so that when an application or framework is working with dynamic Data Objects it is possible to use introspection to determine the shape of the data. Also, in order to support a optimistic concurrency control, the core SDO APIs include change history information that includes the new and old value for a data item that has been changed. Finally, in a practical application there are usually relationships between objects, and it is important that any data access and representation API be able to support integrity of such relationships when objects are deleted (for example). As an example of relationship integrity, consider a data model where an Employee has a reference to a Department, and the Department has a reference to each Employee included in the department. If an employee changes departments, then the data objects need to reflect this change. The Employee should refer to the new Department and be included in the new Department list, and removed from the old Department list.

SDO: Design Points (Continued)

- Designed to integrate well in a tool environment
  - Data objects defined/configured using wizards and views
  - Tools can integrate utilities for generating static data access APIs
- Not intended to replace other data access technologies
- SDO proposal was published jointly by BEA and IBM as JSR 235

One of the important goals of the SDO architecture is to design a data access programming model that makes it easier for tools (IDEs) to allow rapid application development capabilities. The design points mention thus far, including unified data access/representation, dynamic APIs, and data introspection, all contribute to making SDO a tool-enabled technology.

It is important to note that SDO is not intended to replace other types of data access technologies. Rather, SDO is intended to be a layer on top of these technologies to help provide a uniform way to look at a wide range of heterogeneous data.

The initial draft of the SDO proposal has been submitted as JSR 235 as joint work between BEA and IBM. It is the intent that as SDO matures it will become the industry direction.

IBM Software Group

SDO Comparison with Other Technologies

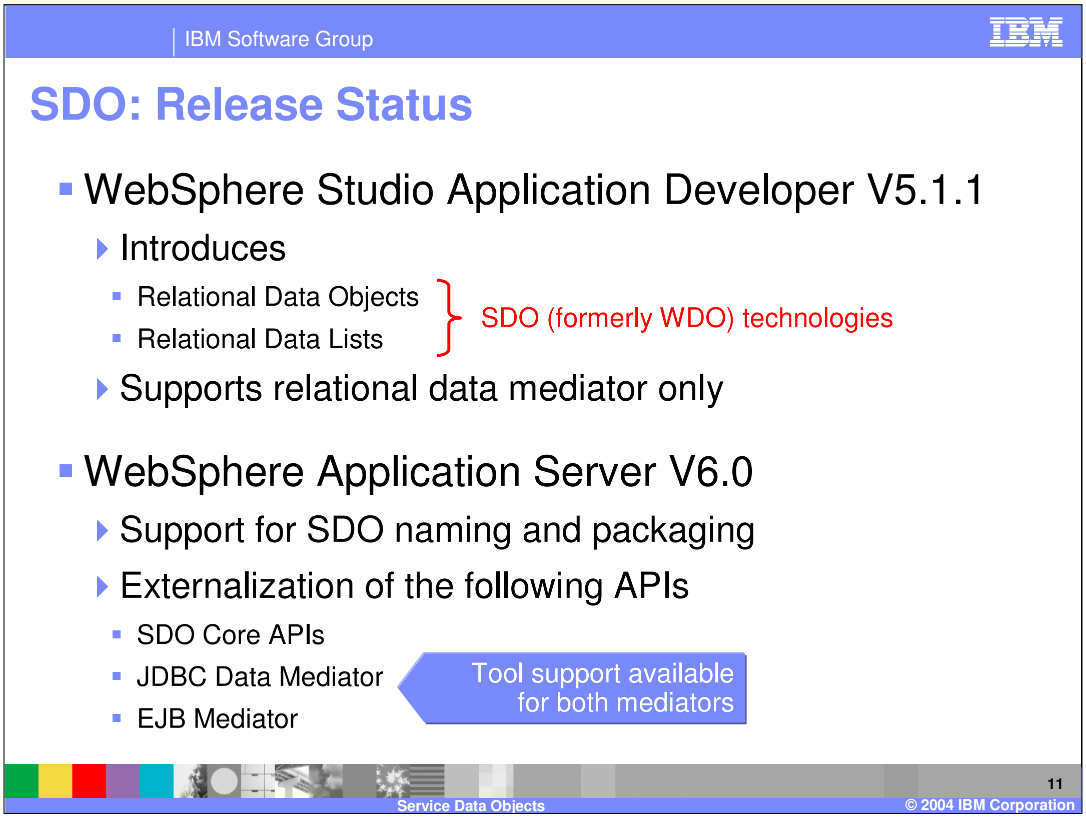
Technology	Model	API	Data Source	Metadata API	Query Language
JDBC RowSet	Connected	Dynamic	Relational	Relational	SQL
JDBC CachedRowSet	Disconnected	Dynamic	Relational	Relational	SQL
Entity EJB	Connected	Static	Relational	Java introspection	EJBQL
JDO	Connected	Static	Relational, Object	Java introspection	JDOQL
JCA	Disconnected	Dynamic	Record-based	Undefined	Undefined
DOM and SAX	N/A	Dynamic	XML	XML infoset	XPath, XQuery
JAXB	N/A	Static	XML	Java introspection	N/A
JAX-RPC	N/A	Static	XML	Java introspection	N/A
SDO	Disconnected	Both	Any	SDO Metadata API,Java Introspection	Any

10

Service Data Objects

This chart was taken from a white paper entitled "Next-Generation Data Programming: Service Data Objects". Refer to this document for a complete discussion on the comparison between SDO and these technologies. This white paper can be found at the following location: http://www.ibm.com/developerworks/library/j-commonj-sdowmt/

SDO is not intended to replace these data access technologies. At first glance it may seem as though SDO is a competing technology, while in fact in most cases SDO is a complimentary technology to the various types of data access, and could be integrated with these technologies. For each of these technologies an SDO data mediator could be created to retrieve the data, and the DataObject APIs could be used to represent the data in a uniform manner. With this scenario, an application developer would not need to understand the data access APIs for all of the technologies listed on this slide. Instead, they would only need to know the SDO APIs, and a small set of mediator specific APIs.



SDO technology was first introduced in version 5.1.1 of WebSphere Studio Application Developer. In this release the SDO technology was referred to as WDO, and was a Beta technology. In version 5.1.1 SDO was introduced in conjunction with JavaServer Faces (JSF) technology and JSP development in the form of two data components that utilized an SDO enabled relational (JDBC) mediator. Relational Data Objects are similar to rows in a database table, but can represent a different schema than the tables they are "bound" to. Relational Data Lists provide support to display multiple Relational Data Objects.

In version 6 of WebSphere Application Server the supporting SDO libraries will be provided with the runtime environment and will support the new naming and packaging of SDO. There will also be an externalization of the JDBC Data Mediator APIs as well as EJB Mediator APIs. IBM Rational Application Developer V6 also provides tool support for both the JDBC Mediator and EJB Mediator SDO technologies.

SDO: Roles

Role	Skills
Application Developer	▪ Knowledgeable in Java and XSD technologies ▪ Uses SDO interfaces (in particular DataObject) ▪ Uses static (generated) SDO APIs ▪ Program to specific data mediator services
Tool / Framework Provider	▪Uses SDO APIs and various data mediators ▪Expert in Java programming ▪Familiar with EMF programming model
Data Mediator Service Provider	▪ Expert in Java programming ▪ Understands the SDO APIs ▪ Skilled in a particular data access technology

The audience of developers interested in SDO technology spans several types of development roles.

Application developers make up the majority of the SDO audience. Application developers are looking for an easy way to access data without knowing the low level API details needed to access the data from a variety of data sources. Application developers will typically use tools and integrated development environments to develop applications.

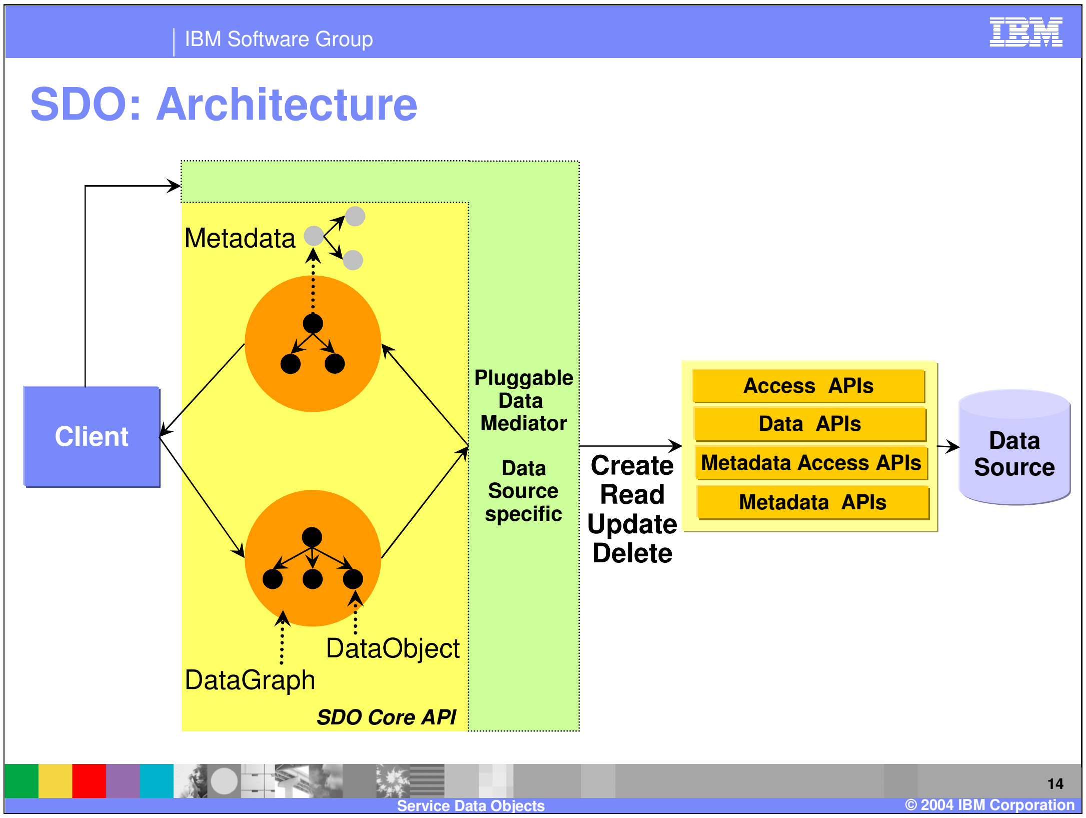
Developers that work on providing tools are also interested in SDO technologies as a way to bring Rapid Application Development capabilities to J2EE development. Developers working on these projects typically need to be familiar with the EMF (Eclipse Model Framework) programming model, as well as experts in java programming and familiar with SDO and a particular mediator API.

Finally, in order for SDO to become more widely exploited, there is a need for developers to develop reusable data mediators for the various types of data sources used by application developers. These developers are typically skilled in a particular type of data access technology and will use these appropriate set of APIs (JDBC, for example) to create the data mediator service .

Section

Architecture

The next section will provide a detailed look at the SDO architecture.



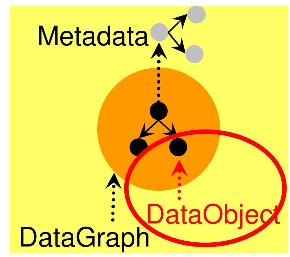
As discussed in the overview section, the SDO architecture is made up of two important pieces, the SDO core APIs and the data mediator service. The SDO Core API is made up of the following key interfaces: DataGraph, DataObject, and introspection APIs (See Type and Property Interfaces). The SDO Core APIs are included in the commonj.sdo package, and the object model diagram is included in the appendix of this presentation.

The pluggable mediator is currently a data source specific interface that the client uses to facilitate creating, deleting, viewing, and updating backend data. The client and mediator use the model defined by SDO core API to pass data back and forth. As shown in this illustration the client need not know the low level details of accessing data for a particular type of data source. All that is needed is the SDO Core APIs and a small set of mediator-specific methods needed to fetch, create, delete, and update back end data.

The specific components that make up the architecture will be discussed in the following slides.

SDO: DataObject

- Data is held in a disconnected, source-independent format defined by the DataObject interface
- Data access APIs are
  - Dynamic (generic)
  - Static and strongly typed (generated)
- Includes a reference to metadata
- Holds primitive data or multi-valued fields
- Supports relationship integrity
- Supports getting/setting values in DataObject graph using an XPATH expression

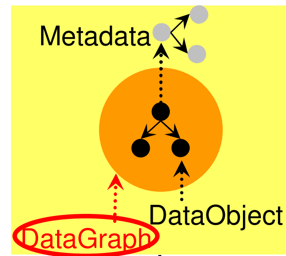


The most fundamental component in the SDO architecture is the DataObject. It is used to store structured data in a source independent and disconnected manner. DataObjects may hold primitive or multi-valued fields (including other DataObjects). Through the DataObject interface, data is held as a set of properties that can be accessed by the property index, name, or a Property object. In the case of using a property's name (string), there is support for passing a string that represents the XPATH location path to get or set a particular DataObject. There are a number of accessor methods for several common primitive types and common Java objects (like java.util.Date). The accessor methods defined for the DataObject interface make up the dynamic API. A DataObject also includes a reference to the corresponding metadata that is accessed through the getType() method. This method returns a commonj.sdo.Type object that contains a list of commonj.sdo.Property objects for the DataObject.

As mentioned previously the DataObject API supports both dynamic and static APIs. The dynamic APIs are automatically provided with the DataObject interface. In the case of static DataObject APIs, utilities are required to provide the code generation for these classes.

SDO: DataGraph

- Represents a basic unit of data transfer between client and data store
- Encapsulates set of DataObjects
  - Contains a root DataObject
  - References metadata from DataObjects
  - All other DataObjects are reachable by traversing the references from the root DataObject



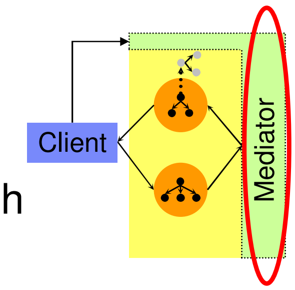
- Contains change information
  - Indicates which DataObjects have been added, updated, or deleted

The DataGraph interface is another important component in the SDO core API. A DataGraph object is the basic unit of transfer between a client and a mediator (or another component). The DataGraph contains a root DataObject. All other Data objects that are included in the DataGraph can be reached by traversing the references from the root DataObject. Metadata is also available via references from DataObjects included in the graph.

All other DataObjects are reachable by traversing the references from the root DataObject. DataGraphs are responsible for including change information that is provided to enable mediators to uphold an optimistic concurrency control. The reason for this is that DataGraphs are independent of any connection to the data source. The change information is included to communicate to the data mediator which DataObjects have been updated or deleted from the datagraph.

SDO: Data Mediator Service

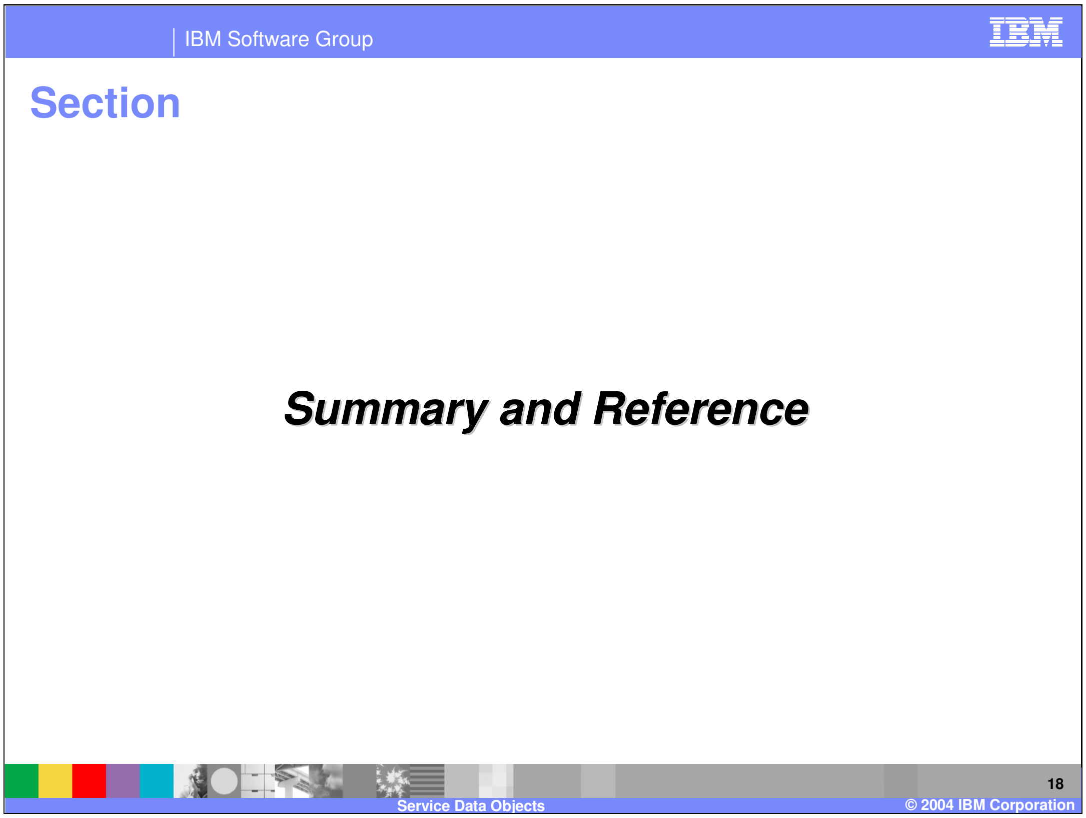
- Created for specific data store
  - JDBC Mediator, EJB Mediator, etc.
- Responsible for populating DataGraph
- Queries and updates the data store
  - Implements an optimistic concurrency strategy for updates
- Stateless with respect to the DataGraph



The data mediator is the component in the SDO architecture that is responsible for facilitating data access between the client and a particular back end data source. It is expected that there will be a mediator for a wide range of data source types. A data mediator is created with backend specific metadata, but the basic interaction between any mediator and a client is similar.

The data mediator is responsible for communicating with the data source and performing updates and queries. During this process the mediator is responsible for creating and populating a DataGraph with DataObjects that reflect the result of a particular query. Likewise, a mediator may also provide the ability to create an empty DataGraph that may be used to insert new DataObejcts by a client that will then be passed back to the mediator to create this data in the data source.

It is important to note that the data mediator is stateless with respect to the DataGraph. Since the DataGraph includes the change information, this is enough for the mediator to employ an optimistic concurrency control strategy in order to determine if a concurrency violation has occurred.



The next section will provide a summary and references for this SDO Overview presentation.

IBM Software Group | IBM

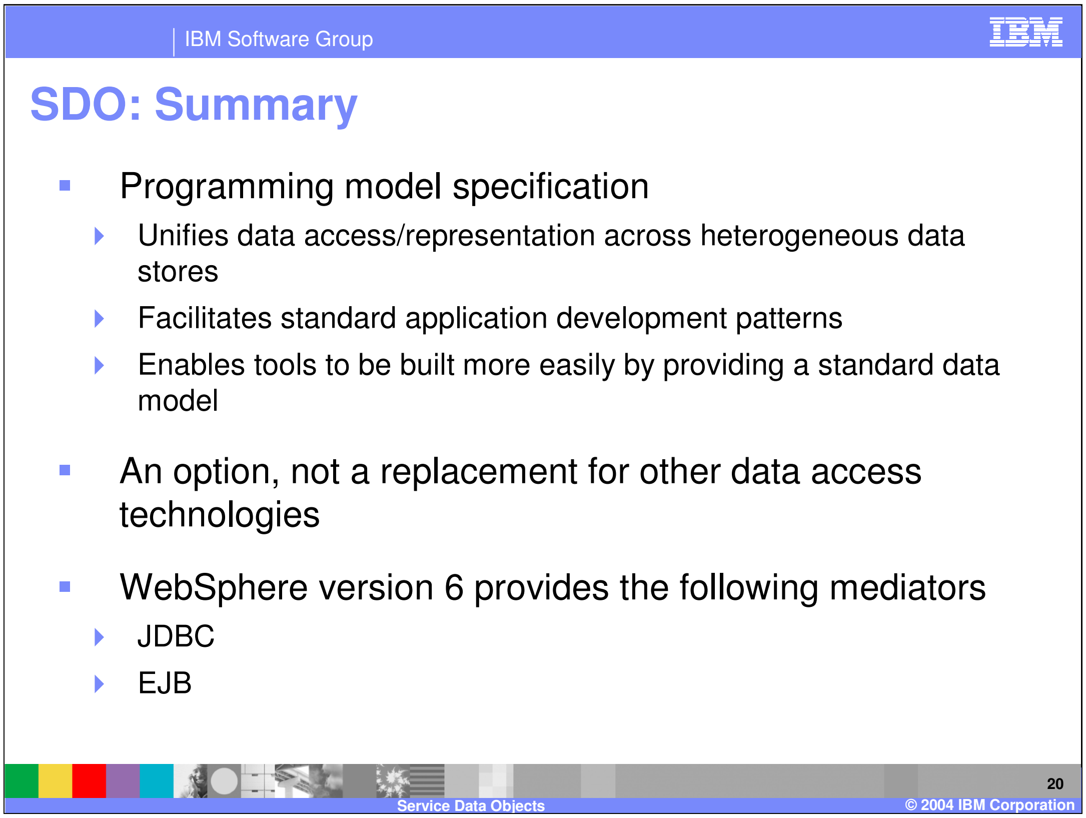
SDO: Usage Scenarios

- JDBC Mediator
  - Integrates well with JSF/JSP to provide J2EE RAD capabilities
- EJB Mediator
  - Transfer Object pattern for CMP EJB DMS
  - Integrates with current EJB and JSF tools

19

Service Data Objects

This slide lists several usage scenarios for the SDO technology. This presentation did not discuss the technical details regarding the JDBC and EJB mediators, however, this information has been provided in another presentation.



SDO is a new programming model specification that is aimed at providing a unified data representation across heterogeneous data stores. This technology facilitates standard application patterns, and enables tools to be built more easily by providing a standard data model.

It is important to note that SDO is not intended to be a replacement for other types of data access technologies, rather it is expected SDO will be used in conjunction with these technologies.

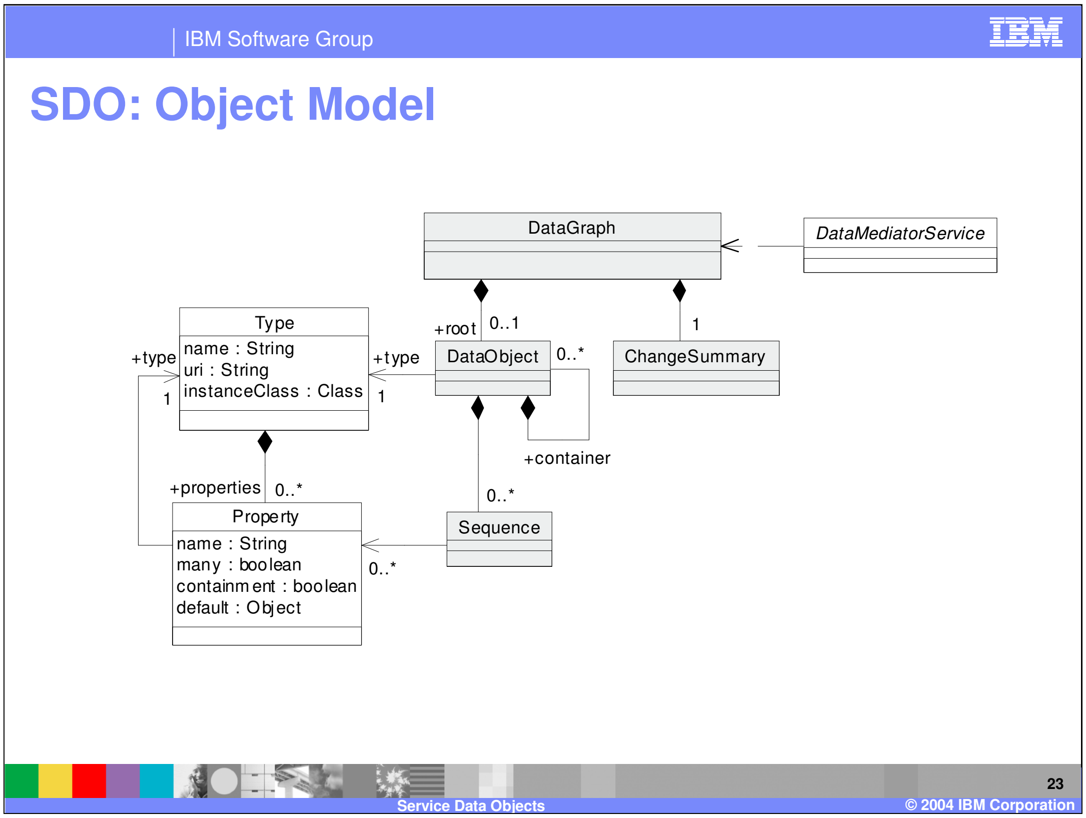
Although it was not discussed in this presentation, there are several important data mediators that are available with WebSphere Application Server V6. These mediators include the JDBC and EJB mediators. These mediators are discussed in another presentation.

References

- Developer Works:
  - http://www.ibm.com/developerworks/library/j-commonj-sdowmt/

Section

Appendix: Service Data Objects

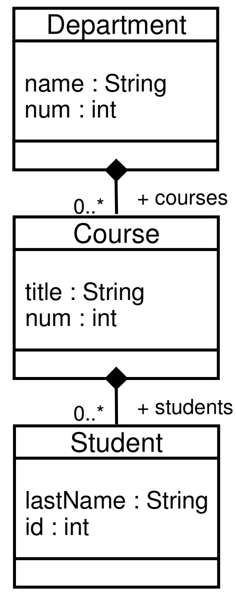


This diagram was taken from the Service Data Objects specification. This document, along with the details of the SDO Java APIs can be found at the following location: http://www.ibm.com/developerworks/library/j-commonj-sdowmt/

SDO Example

```
//Use Data Mediator to get data graph
//Then get root DataObject
DataObject root = dataGraph.getRootObject();
DataObject dept = root.getDataObject(“department”);

//Get the first course in the data graph
List courses = (DataObject) dept.getList(“courses”);
DataObject course = dept.get(0);

//Access a student in the first course
List students = course.getList(“students”);
DataObject student = (DataObject) students.get(1);

//Same as above code snippet, but using XPATH
String xpath = “courses.0/students.1”;
DataObject studentWithXPATH = dept.getDataObject(xpath);

//Another XPATH example
xpath = “courses[num=567]/students[id=555]”;
DataObject studentWithXPATH = dept.getDataObject(xpath);
```

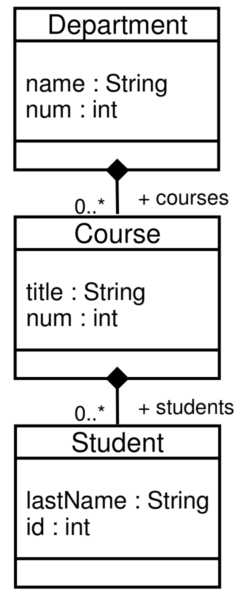
Model



SDO Example (Continued)

```
//Make data modifications
dept.setString("name", "Computer Science");

//Creating a new DataObject
DataObject newStudent =
           course.createDataObject("students");
newStudent.set("lastName", "Smith");
newStudent.set("id", "556");


//Deleting a DataObject from the DataGraph
student.delete();
```

Model



Trademarks, Copyrights, and Disclaimers

The following terms are trademarks or registered trademarks of International Business Machines Corporation in the United States, other countries, or both:

IBM	CICS	IMS	MQSeries	Tivoli
IBM(logo)	Cloudscape	Informix	OS/390	WebSphere
e(logo)business	DB2	iSeries	OS/400	xSeries
AIX	DB2 Universal Database	Lotus	pSeries	zSeries

Java and all Java-based trademarks are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

Microsoft, Windows, Windows NT, and the Windows logo are registered trademarks of Microsoft Corporation in the United States, other countries, or both.

Intel, ActionMedia, LANDesk, MMX, Pentium and ProShare are trademarks of Intel Corporation in the United States, other countries, or both.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Linux is a registered trademark of Linus Torvalds.

Other company, product and service names may be trademarks or service marks of others.

Product data has been reviewed for accuracy as of the date of initial publication. Product data is subject to change without notice. This document could include technical inaccuracies or typographical errors. IBM may make improvements and/or changes in the product(s) and/or program(s) described herein at any time without notice. Any statements regarding IBM's future direction and intent are subject to change or withdrawal without notice, and represent goals and objectives only. References in this document to IBM products, programs, or services does not imply that IBM intends to make such products, programs or services available in all countries in which IBM operates or does business. Any reference to an IBM Program Product in this document is not intended to state or imply that only that program product may be used. Any functionally equivalent program, that does not infringe IBM's intellectual property rights, may be used instead.

Information is provided "AS IS" without warranty of any kind. THE INFORMATION PROVIDED IN THIS DOCUMENT IS DISTRIBUTED "AS IS" WITHOUT ANY WARRANTY, EITHER EXPRESS OR IMPLIED. IBM EXPRESSLY DISCLAIMS ANY WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NONINFRINGEMENT. IBM shall have no responsibility to update this information. IBM products are warranted, if at all, according to the terms and conditions of the agreements (e.g., IBM Customer Agreement, Statement of Limited Warranty, International Program License Agreement, etc.) under which they are provided. Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products in connection with this publication and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. IBM makes no representations or warranties, express or implied, regarding non-IBM products and services.

The provision of the information contained herein is not intended to, and does not, grant any right or license under any IBM patents or copyrights. Inquiries regarding patent or copyright licenses should be made, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A.

Performance is based on measurements and projections using standard IBM benchmarks in a controlled environment. All customer examples described are presented as illustrations of how those customers have used IBM products and the results they may have achieved. The actual throughput or performance that any user will experience will vary depending upon considerations such as the amount of multiprogramming in the user's job stream, the I/O configuration, the storage configuration, and the workload processed. Therefore, no assurance can be given that an individual user will achieve throughput or performance improvements equivalent to the ratios stated here.

Note to U.S. Government Users - Documentation related to restricted rights-Use, duplication or disclosure is subject to restrictions set forth in GSA ADP Schedule Contract and IBM Corp.