Service Data Objects For Java Specification

Version 2.1.0, November 2006

Authors

Matthew Adams Xcalia

Cezar Andrei BEA Systems, Inc.

Ron Barack SAP AG

Henning Blohm SAP AG

Christophe Boutard Xcalia

Stephen Brodsky IBM Corporation

Frank Budinsky IBM Corporation

Stefan Bünnig SAP AG

Michael Carey BEA Systems, Inc.

Blaise Doughan Oracle Corporation

Andy Grove Rogue Wave Software

Omar Halaseh Oracle Corporation

Larry Harris Oracle Corporation

Ulf von Mersewsky SAP AG

Shawn Moe IBM Corporation

Martin Nally IBM Corporation

Radu Preotiuc-Pietro BEA Systems, Inc.

Mike Rowley BEA Systems, Inc.

Eric Samson Xcalia

James Taylor BEA Systems, Inc.

Arnaud Thiefaine Xcalia

Copyright Notice

.

License

Status of this Document

This specification may change before final release and you are cautioned against relying on the content of this specification. The authors are currently soliciting your contributions and suggestions. Licenses are available for the purposes of feedback and (optionally) for implementation.

Table of Contents

Contents

1 Introduction

Service Data Objects (SDO) is a data programming **architecture** and an **API**.
The main purpose of SDO is to simplify data programming, so that developers can focus on business logic instead of the underlying technology.
SDO simplifies data programming by:

- unifying data programming across data source types
- providing support for common application patterns
- enabling applications, tools and frameworks to more easily query, view, bind, update, and introspect data.

For a high-level overview of SDO, see the white paper titled "Next-Generation Data Programming: Service Data Objects" [**?**] .

1.1 Key Concepts

The key concepts in the SDO architecture are the **Data Object**, the **data graph** and the **Data Access Services (DAS)**.
A Data Object holds a set of named properties, each of which contains either a simple data-type value or a reference to another Data Object. The Data Object API provides a dynamic data API for manipulating these properties.
The data graph provides an envelope for Data Objects, and is the normal unit of transport between components. Data graphs can track changes made to the graph of Data Objects. Changes include inserting Data Objects, deleting Data Objects and modifying Data Object property values.
Usually, data graphs are constructed from one of the following:

- Data sources
  - such as XML files, Enterprise Java^{TM} Beans (EJBs), XML databases and relational databases.
- Services
  - such as Web services, Java Connector Architecture (JCA) Resource Adapters and Java Message Service (JMS) messages.

Components that can populate data graphs from data sources and commit changes to data graphs back to the data source are called Data Access Services (DAS). The DAS architecture and APIs are outside the scope of this specification.

1.2 Requirements

The scope of the SDO specification includes the following requirements:

1. **Dynamic Data API.** Data Objects often have typed Java interfaces. However, sometimes it is either impossible or undesirable to create Java interfaces to represent the Data Objects. One common reason for this is when the data being transferred is defined by the output of a query. Examples would be:

2. A relational query against a relational persistence store.

3. An EJBQL queries against an EJB entity bean domain model.

4. Web services.

5. XML queries against an XML source.

6. When deployment of generated code is not practical.

   In these situations, it is necessary to use a dynamic store and associated API. SDO has the ability to represent Data Objects through a standard dynamic data API.

1. **Support for Static Data API.** In cases where metadata is known at development time (for example, the XML Schema definition or the SQL relational schema is known), SDO supports code-generating interfaces for Data Objects. When static data APIs are used, the dynamic data APIs are still available. SDO enables static data API code generation from a variety of metamodels, including:

2. Popular XML schema languages.

3. Relational database schemas with queries known at the time of code generation.

4. Web services, when the message is specified by an XML schema.

5. JCA connectors.

6. JMS message formats.

7. UML models

   While code-generation rules for static data APIs is outside the scope of this core SDO specification, it is the intent that SDO supports code-generated approaches for Data Objects.

1. **Complex Data Objects.** It is common to have to deal with "complex" or "compound" Data Objects. This is the case where the Data Object is the root of a tree, or even a graph of objects. An example of a tree would be a Data Object for an Order that has references to other Data Objects for the Line Items. If each of the Line Items had a reference to a Data Object for Product Descriptions, the set of objects would form a graph. When dealing with compound data objects, the change history is significantly harder to implement because inserts, deletes, adds, removes and re-orderings have to be tracked, as well as simple changes. Service Data Objects support arbitrary graphs of Data Objects with full change summaries.

1. **Change Summary.** It is a common pattern for a client to receive a Data Object from another program component, make updates to the Data Object, and then pass the modified Data Object back to the other program component. To support this scenario, it is often important for the program component receiving the modified Data Object to know what modifications were made. In simple cases, knowing whether or not the Data Object was modified can be enough. For other cases, it can be necessary (or at least desirable) to know which properties were modified. Some standard optimistic collision detection algorithms require knowledge not only of which columns changed, but what the previous values were. Service Data Objects support full change summary.

1. **Navigation through graphs of data.** SDO provides navigation capabilities on the dynamic data API. All Data Objects are reachable by breadth-first or depth-first traversals, or by using a subset of XPath 1.0 expressions.

1. **Metadata.** Many applications are coded with built-in knowledge of the shape of the data being returned. These applications know which methods to call or fields to access on the Data Objects they use. However, in order to enable development of generic or framework code that works with Data Objects, it is important to be able to introspect on Data Object metadata, which exposes the data model for the Data Objects. As Java reflection does not return sufficient information, SDO provides APIs for metadata. SDO metadata may be derived from:

2. XML Schema

3. EMOF (Essential Meta Object Facility)

4. Java

5. Relational databases

6. Other structured representations.

1. **Validation and Constraints.**

2. Supports validation of the standard set of constraints captured in the metadata. The metadata captures common constraints expressible in XML Schema and relational models (for example, occurrence constraints).

3. Provides an extensibility mechanism for adding custom constraints and validation.

1. **Relationship integrity.**

2. An important special case of constraints is the ability to define relationships between objects and to enforce the integrity of those constraints, including cardinality, ownership semantics and inverses. For example, consider the case where an employee has a relationship to its department and a department inversely has a list of its employees. If an employee's department identifier is changed then the employee should be removed, automatically, from the original department's list. Also, the employee should be added to the list of employees for the new department. Data Object relationships use regular Java objects as opposed to primary and foreign keys with external relationships.

3. Support for containment tree integrity is also important.

NOTE the following areas are out of scope:

1. **Complete metamodel and metadata API.** SDO includes a minimal metadata access API for use by Data Object client programmers. The intention is to provide a very simple client view of the model. For more complete metadata access, SDO may be used in conjunction with common metamodels and schema languages, such as XML Schema [**?**] and the EMOF compliance point from the MOF2 specification [**?**] . Java annotations in JSR 175 may be a future source of metadata.

2. **Data Access Service (DAS) specification.** Service Data Objects can be used in conjunction with "data accessors". Data accessors can populate data graphs with Data Objects from back-end data sources, and then apply changes to a data graph back to a data source. A data access service framework is out of scope but will be included in a future Data Access Service specification .

1.3 Organization of this Document

This specification is organized as follows:

- **Architecture**: Describes the overall SDO system.
- **Java API**: Defines and describes the Java API for SDO.
- **Generating Java from XML Schemas**: Shows how Java is generated from XML Schemas (XSD).

- **Java Interface Specification**: Defines how Java interfaces are generated and used.
- **Java Serialization of DataObjects**: Defines how to serialize DataObjects.
- **SDO Model for Types and Properties**: Shows the SDO Type and Property in model form.
- **Standard SDO Types**: Defines and describes the Standard SDO Types.
- **XML Schema to SDO Mapping**: Defines and describes how XML Schema declarations (XSD) are mapped to SDO Types and Properties.
- **Generation of XSD from SDO Type and Property**: Describes how to generate XSDs from SDO Types and Properties.
- **XPath Expression for DataObjects**: Defines an augmented subset of XPath that can be used with SDO for traversing through Data Objects.
- **Examples**: Provides a set of examples showing how SDO is used.
- **DataType Conversion Tables**: Shows the set of defined datatype conversions.