Apache Tuscany
RDB DAS

Kevin Williams

Luciano Resende

Agenda

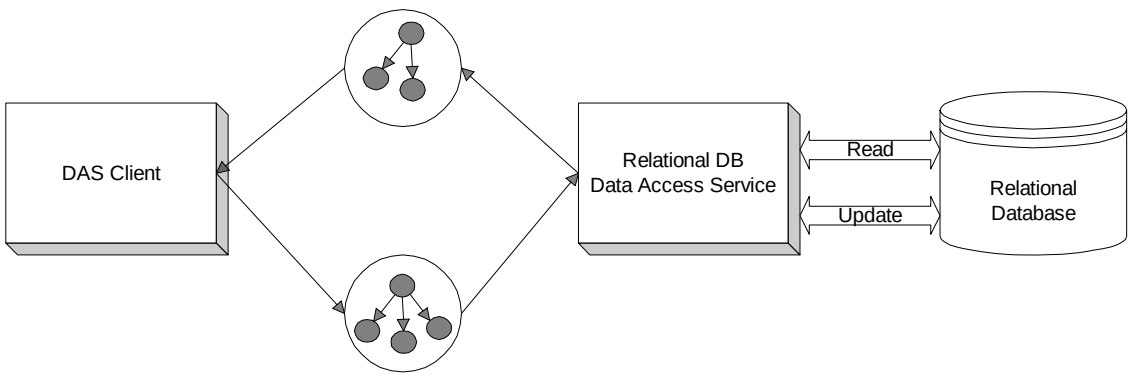
- **Overview**
- **Programming model**
- **Some Examples**
- **Where to get more**

Overview

- **SDO 2.1 Specification - DAS definition:**

  "Components that can populate data graphs from data sources and commit changes to datagraphs back to the data source are called Data Access Services"

- **Data Access Services (DAS) provides two fundamental capabilities:**
  - Results of a data source query returned as a graph of Data Objects
  - Graph modifications reflected back to data source as a series of Creates/Updates/Deletes



Overview

- **DAS provides a simple and intuitive interface**
  - Based on Command pattern
  - Most read interactions consist of acquiring a command and then executing it
  - Most write interactions consist of providing a modified graph (with change summary) to a DAS instance

- **Progressive programming model**
  - Simple tasks are simple
  - More complicated tasks are a little less simple

Overview – simplest read

```
// create a DAS instance with a given database connection
DAS das = DAS.FACTORY.createDAS(getConnection());

// create a Command instance using a given SQL query statement
Command readCustomer = das.createCommand("select * from CUSTOMER where ID = 1");

// execute the query, returning the root of the data graph representing the result set
DataObject root = readCustomer.executeQuery();

// obtain the first (and only) CUSTOMER data object from the root data object
DataObject cust = root.getDataObject("CUSTOMER[1]");
```

Overview - write

- **Pushing changes back to the DB can be equally straightforward (continuing previous example)**

```
cust.setString (“LASTNAME”, “Williams”);
das.applyChanges(root);
```

- **Note the lack of configuration (no side file) for these two examples**
  - Convention over configuration
    - SDO properties named “X” will map to column “X” on the data source repository
    - SDO properties named “ID” will be considered Primary Keys

Overview – config file

- **Configuration file can be used to organize related sets of commands and other configurable items**
- **Previous examples employing a config file**

```
DAS das = DAS.FACTORY.createDAS(getConfig(“customerConfig.xml"),
   getConnection());
Command read = das.getCommand(“read customer”);
DataObject root = read.executeQuery();
DataObject cust =  root.getDataObject(“CUSTOMER[1]”);
…
cust.setString (“LASTNAME”, “Williams”);
das.applyChanges(root);
```

- **SQL has been moved to a side-file file**

```
<Config xmlns="http:///org.apache.tuscany.das.rdb/config.xsd">
  <Command name="read customer" SQL="select * from CUSTOMER where
   ID = 10021" kind="Select"/>
</Config>
```

Programming Model - capabilities

- **DAS supports simple database interactions with a simple API but more complex scenarios are also supported**
  - Statically typed (generated) SDO Data Objects
  - Optimistic concurrency control
  - Generated database IDs
  - Stored procedures
  - External transaction participation
  - Simple name mapping (Table/Column -> SDO Type/property)
  - Column-type conversions
  - Paging
- **Future capabilities**
  - SCA integration

Programming Model - Example

3.6 Stored Procedure (Read)

This example demonstrates the ability to execute a stored procedure and return a graph of data objects.

3.6.1 Example

```
// create a DAS instance with a given database connection
DAS das = DAS.FACTORY.createDAS(getConnection());

// create a Command instance that specifies the "GETALLCOMPANIES" stored procedure
Command read = das.createCommand("{call GETALLCOMPANIES()}");

// execute the stored procedure, returning the root of the data graph representing the
//    result set
DataObject root = read.executeQuery();

// obtain the value of the ID column from the first company in the result
Int id = root.getInt("COMPANY[1]/ID");
```

Programming Model - Example

3.13 Stored Procedure OUT Parameters

This example demonstrates ability to execute a stored procedure and retrieve an OUT or IN/OUT parameter.

3.13.1 Example

```
// create a DAS instance with a given configuration and database connection
DAS das = DAS.FACTORY.createDAS(getConfig("StoredProcTest.xml"), getConnection());

// create Command instance
Command read = das.getCommand("getNamedCustomers");

// set input parameter
read.setParameter(1, "Williams");

// execute the query, returning the root of the data graph representing the result set
DataObject root = read.executeQuery();

// obtain value from output parameter
Integer customersRead = (Integer) read.getParameter(2);

// verify that expected information was obtained
assertEquals(4, customersRead.intValue());
assertEquals(customersRead.intValue(), root.getList("CUSTOMER").size());
```

Programming Model

3.7 Paging

This example demonstrates the ability to examine portions of a query result set in a disconnected fashion.

3.7.1 Example

```
// create a DAS instance with a given database connection
DAS das = DAS.FACTORY.createDAS(getConnection());

// create a Command instance that reads all customer records when executed
Command custCommand = das.createCommand("select * from CUSTOMER order by ID");

// create a Pager instance with a specified query and a page size of two records
Pager pager = DAS.FACTORY.createPager(custCommand, 2);

// get and work with first page
DataObject root = pager.next();
DataObject customer1 = root.getDataObject("CUSTOMER[1]");
DataObject customer2 = root.getDataObject("CUSTOMER[2]");

// get and work with the second page
root = pager.next();
customer1 = root.getDataObject("CUSTOMER[1]");
customer2 = root.getDataObject("CUSTOMER[2]");

// get and work with first page again
root = pager.previous();
customer1 = root.getDataObject("CUSTOMER[1]");
customer2 = root.getDataObject("CUSTOMER[2]");
```

Where to get more

- **Possibly more than you actually want to know about RDB-DAS**
  - Home page:
    - http://incubator.apache.org/tuscany/rdb-das-java.html