Table of Contents

Introduction to Object Automation

"*OA is a best practice development system that uses sophisticated design patterns to create complete complex systems in a consistent and precise way*."

OA is a software development system that creates applications using a visual modeling tool. This model driven approach allows for creating complex and powerful applications in a way that is organized and easy to maintain, where most of the code can be automatically built into the components and generated.

Motivation

The idea behind the development of OA has been to automate software development using a model driven approach. By using OABuilder, complex models can be designed and applications can be generated from these models, allowing the software designer to be more focused on the application requirements and less on the programming.

Reactive Models

Similar to how calculations on a spreadsheet are automatically updated, OA allows for this in the application model, so that the model is always in sync – even with other computers. This reactive modeling technology allows the model to be the application, and allows for other components to plug into it.

Disruptive Technology

Similar to when compilers replaced assemblers, model driven development (MDD) has had the potential to affect the software industry in the same way. This is not to just improve speed of development, but also quality – in a way that creates software that is at a whole new level, and can handle the complexity that solutions need.

The software industry has had CASE tools, RAD, UML, MDA, DSL, code generation for many years. These solutions make some complex things easy, and make some easy things too complex. The other problem is that they are shallow – don't handle the depths needed in real world applications. It is an investment (and risk) for developers to learn these tools. The other problem is that end products lack the quality and performance needed, not to mention issues with vendor tie-in.

The overall consensus is that these solutions are too hard to use, too rigid, and can't be used to produce high quality products. The end result is that "*modeling and design for most projects has not evolved"* in the way that other development technologies have evolved. When a new tool comes along into this space, there is skepticism and resistance - and rightfully so.

With OA, we don't *just* have a new tool, code generators, language, wiz-bang, etc. OA is a *new way* for development using "*reactive models*", and combining advanced MVC patterns, with object oriented programming. Using this technology, we have then built tools that contribute to better software solutions, and continue to expand automating software development.

Using OA, tools and code generation "*make sense"* and fill the missing and lagging evolution gap in the present development process.

The unique features of OA

Observable – *"the secret sauce"*

The main feature behind OA is the fact that object graphs are observable - "*reactive models*". This is a very powerful feature that is highly valued in application development. The downside to this approach has been that it usually would result in a heavy amount of overhead that would restrict its use to small amounts of objects.

OA has sophisticated techniques that overcome this overhead so that this is no longer a limiting factor.

Features

- Observable object and collection class that uses sophisticated techniques to *overcome* the overhead and complexity that has limited the use of the observable patterns.
- Reactive object oriented models.
- Expanded use of MVC pattern
- Model and Meta-data driven
- Reflective utilities for automating reusability functionality.
- Combines Reactive, Concurrent, and Parallel programming with Object Oriented.
- Property paths for working with Object Graphs
- Auto Messaging for real-time, distributed, multi-tiered, synchronized applications
- Caching – local, distributed, synchronized, "acting as one" system.
- DataSource independent, with support for any persistent layer.
- Object binding UI components for Desktop and Web Apps.
- Tools driven architecture – easy to create and expand RAD, frameworks, reusable functionality, automation to other APIs.

The OA Software Development System includes a single library, and a visual design tool "**OABuilder**", which include application generation tools, project templates, to help create complex systems using a consistent "*cookie cutter*" approach. OABuilder was built using the OA libraries and in turn offers a high productivity modeling environment.

OA includes the "*building blocks*" for creating software applications, with enterprise level requirements. This is an *A to Z* solution, for building scalable and distributed systems. OA handles the "*Heavy Lifting*" in a reusable way, so that focus is on the application and less on the tools, and technology.

OA uses a Reactive Model/Meta-data and MVC approach, where the OAModel defines and includes information about the application data structures and how they work together. This model then works as an observable object graph that creates the equivalent of a software *circuit board*.

Some of the priorities with OA are:

- Must be easy not to use – *eliminate the 80/20 rule*
- Easy to isolate any part – *testing, performance, bug tracking*
- Advanced observable patterns – *wired relationships and functionality*
- Finds and uses best practices – *always improving*
- Reusability – *everything works with the model*
- design patterns – *the laws of software development*
- Separate the heavy technical lifting – *layers of technology*
- Focus – *application developers need to focus on end product, less on technology*
- Perfect enough – *always improving through an reiterative process*
- Simplify – through using reusable functionality.

MDA + MVC + BPM + Reactive + Reflective + OOUI + Generators + Templates

Reactive Model Driven Architecture – defines the data and object model, and how it works together.
Model/View/Control – wires/binds the model to the view and user presentation components.
Business Process Management – validations, processing, batching, security, visibility, requirements.
Reflection API – introspection routines for automating how things work together.
Custom Code Generation – transforming models into code.
Template – reusable code, auto customized for model.
Object Oriented User Interface (OOUI) – user interface interacts explicitly with objects that represents entities in the application domain – *object bound components*.

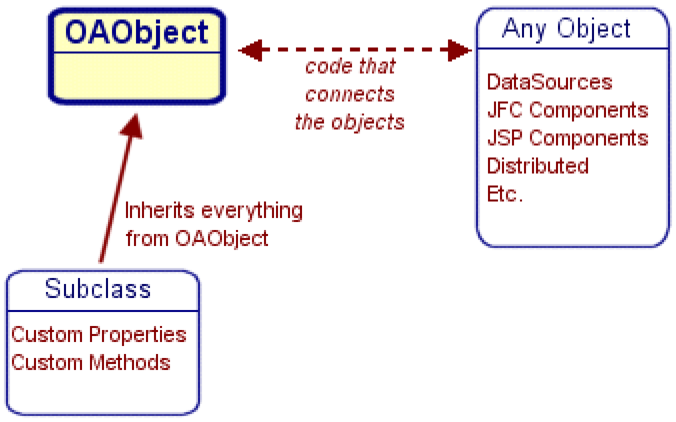
All of this combined allows for creating full applications in a consistent and flexible way. Our experience is that over 90% of the programming is automated. A big advantage is that this is a system for creating reusable functionality and tools.

Reactive Model Core

An OAModel is the data structures for an application.

At the core, OA is based around a two Java classes for data models, *"OAObject"* and a "*Hub*" collection class. These classes are the data and business access layer.

The OAObject class is extended to include custom properties and methods required to build custom data objects. This is used to form DAO type objects.



OAObjects automatically work with other OAObjects, forming relationships between the Objects. These relationships, which are called references, between OAObjects are automatically managed: One-One, One-Many, Many-Many, Self Referring Recursive, Cascading, and more.

OAObjects automatically work with, and are independent of, any DataSource. All queries are based on the properties of your OAObjects – the Object Model. The database structure is well formed and normalized, without any special requirements.

Support for JDBC Databases, XML, JSON, Rest, serialized file, data files, legacy systems, application servers, and more.

OAObjects automatically work with any Java Foundation Classes (JFC/Swing)
JTree, JTable, JList, JComboBox, JTextPane, JTextField, JButton, HTML word processor, Report writer, image editor, and more.

OAObjects automatically work with Web Pages / Java Server Pages (JSP)

Ajax enabled components similar to JFC components, pages, Forms, email, HTML element control, and more.

OAObjects automatically work with other Applications - Client, Server, distributed, synchronized

What you don't need

Even though you can use any of these, you will not need: Spring, Struts, Hibernate, IBatus, ORM xyz, apache project xyz, DSL language xyz, Cache engines, messaging systems, wizbang silver bullet, etc.

Dependency Injection (DI) type libraries have features that OA handles naturally by using annotations, and some core reflective libraries. The fact that your OA model objects are observable makes it easy to have your application objects *reactive* without using a DI type solution.

As for messaging and distributed communications, OA includes built-in functionality that makes your applications automatically distributed and in-sync on a real time bases. Methods can be made to be distributed so that a method called on one system is executed on another system.

Chapter 1 Reactive Model

Overview

OAObject is a Java *"base class"* that is an observable data access object, automating and eliminating most of the programming required to get objects to *work* together within an application. By being observable, interactions can be automated.

data, data, data

The core to any application is the "data" and the *"rules"* that manage the data. In code this is also the "*Object Graph*".

While designing custom software, most of the time is spent defining the data and how it will be used.

- Where (and how) does data come from (datasource)
- What are the processes and validation rules for managing the data
- Security – access to data and functions.
- How is the data related to other data
- How is the data presented to users
- How is the data stored
- What happens when data is modified
- How to manage relationships between data
- How is data kept updated real-time, batch mode, or eventually.
- Distributing data across servers, messaging, remote method calls, background processing, etc.

Once the information about the data is defined, programmers write the software. Most of software development is spent writing code that uses and manages the data. Programmers create classes that are used to store, validate and protect the data, and then write the code to store, retrieve, present to user, manages how objects work together and more. This *"code between the objects"* can be an enormous effort that requires a wide range of technical expertise.

The data for an application is then defined as a group of "classes". A class describes the data and commands (called methods). The concept for defining the data for a class is very similar to defining Columns for a Table used in a Database. The advantage of using a class is that you can build validation and methods that protect and work with the data. The other advantage is that the class is not dependent on a specific database or technology.

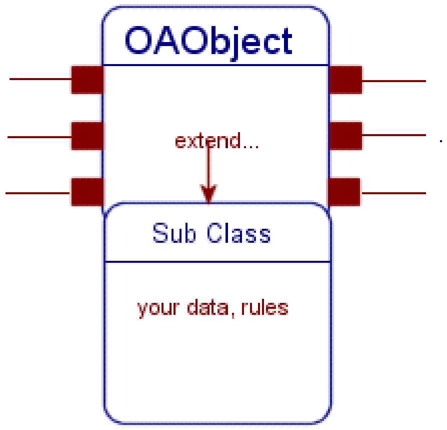
An Object is an instance of a Class, similar to how a Record is an instance of a Table. Just as a Table describes the data for a Record, a Class describes the structure of an Object. In a program, Objects are then created and used. A group of Classes and the type of relationships between them make up an *"Object Model"*. Just as a Database can be used to understand an application, it can be said that *"The Object Model describes the Application"*.

The classes in the Java Libraries were designed so that they could be easily used by any other class. To do this, a programmer must have the knowledge about the class and how it works. Programmers then write the code that connects Objects together. This *"code between the Objects"* can be time consuming and is error prone. Worse yet, there is not a consistent and "generic" way that programmers do this, increasing the time to manage code.

Created correctly, classes for an application can be designed and built so that they can work directly with any other classes and at the same time be independent of them. This creates a clear separation between objects, which allows for better managed software. Taken a step further, a base class can be created to offer this functionality, creating a *"reusable"* class that can work with other classes.

The problem with creating a base class that has all of this required functionality is that it would be too "bulky" and "complex", making it impractical to use. By making the OAObject observable using some special techniques, it is able to be light weight, flexible and scalable.

OAObject can be described as a base *"Universal"* DAO Class that is both light weight and simple to use. Applications can be built using reusable components. This allows developers the freedom to concentrate on the application, and less on the technology.



OAObject automatically works with:

- Other OAObjects, Hub collections to form powerful relationships between objects.
- Any DataSource – Independent of any Database, legacy system, application server, etc. – making it an object-oriented datastore.
- Any JFC/Swing component – the JFC Visual Components have been sub-classed to be bindable, so that they can easily work with any OAObject.
- Any JSP/Web Page – Java Server Pages, Ajax UI components, HTML Forms have all been automated to work directly with OAObjects.
- Other Applications and Computers – OAObjects communicate to a central object that can be set up to communicate with other applications and computers. Methods can be set up to be distributed, so that the actually implementation is on another computer, but treated as a local method call. These calls can be async to handle higher throughput.

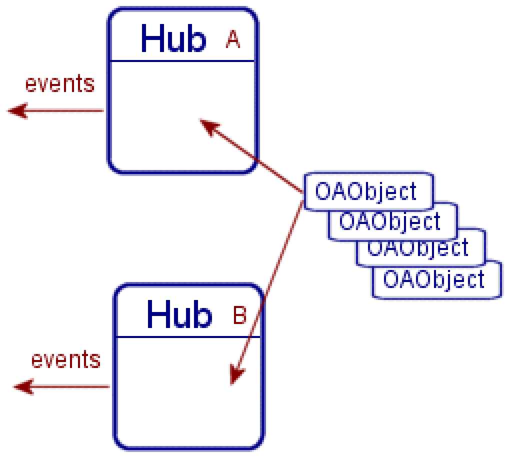
Built-In Functionality
This is a summary of what is included in OAObject. More complete definitions that describe the *"how's and whys"* is included in other chapters and in the API documentation.

- Annotations to expand meta data information.
- Reflective functionality automates binding, messaging and object relationships.
- **Objec**t Key – property values that make this object unique.
- Reference Information – how objects are related to other object. All references use the actual objects and not the key (or foreign key value). References types include one-one, one-many, many-many, recursive self references, owned and un-owned references, and more.
- Manages reference objects when working with database/datasource. "Moves" objects when changes are made to a reference property.
- Store miscellaneous data
- Initialization during creation
- Null Values – to know if a primitive property value is null
- Knows which Hub Collections (next section) object is a member of – allowing it to expand the observability functionality.
- Handles events for object – ex: property changes, before/after events, collection events, etc. All of the events use a single event class.
- Knows if objects is "new", or "changed"
- Cascading rules. Cancel, Save, Delete can be cascaded to reference objects.
- Allows for canceling changes
- Save and Delete methods, with cascading
- Calculated Properties – properties that rely on other properties or objects for their value.
- Serialization Support – to file/stream, other applications, compression, defining what gets serialized and how.
- XML, JSON, Rest, Java binary – reading and writing
- Locking, Transaction management
- Concurrency – handle issues like keeping multiple servers in sync, thread safe changes, deadlock detection and recovery.
- Zero admin data access. Backups, DB DDL updates, detecting data corruption and recovery.

Hub Collection - *many* OAObjects

Everything in programming is either an Object or a group (collection) of Objects. The "Hub Collection" is observable collection class used for storing OAObjects. The Hub is similar to using both the ArrayList and HashMap Collection Classes.

Like OAObject, Hub Collections are *"Observable"*, meaning that other objects can be notified when changes are done to a Hub or with any object within the Hub. A unique feature of OA is that the events are delegated to the Hubs that contain it, eliminating most of the overhead responsible for handling listeners and event notification.



For Applications, Hubs can be configured to form *relationships*. The Hubs can then be directly used by other components, using object binding. These relationships are based on the relationships of the actual OAObject classes, and can automatically be kept in-sync.

Relationship Types:

- Master/Detail – where the active object in a "Master" Hub determines the list of Objects in another "Detail" Hub.
- Linking – so that when an active object is changed, then the property that it is linked to will automatically be updated.
- Sharing – where more than one Hub can be using/sharing the same data, but for different purposes. This is usually used when binding, etc.
- Recursive – a self referencing relationship. Where a Class has a parent and many children of the same Class.
- Filtering – Hub Collection can filter the objects in the collection. Any changes automatically update the collection, so that it never has to be refreshed.
- Merging – where a collection is populated and kept in sync based on a property path.

Other

- Sequence properties – Hubs can automatically update a property in an object as to its position in the collection. This allows for ordering in cases where there is not a property to sort off of.
- Querying – Hubs can select/query objects from a DataSource, using an Object Query Language based on the OAObjects.
- Methods for sorting and finding objects.

DataSource Independent

OA includes a single Class that defines how OAObjects can work with a DataSource. A DataSource includes any form of data storage: database, xml, legacy system, application server, Object/Relational Mapping tools, etc. This Class, named OADataSource, allows for selecting/querying, reading, writing OAObjects to and from any datasource.

All queries are based on the structure of the OAObjects and return data as *real* objects and not resultsets. Since OAObjects are independent of the DataSource, it is easy to create or change the DataSource at anytime. OABuilder includes a generator for creating SQL Scripts and a Mapping Object for creating and updating a Database so that it matches an Object Model.

OA Binding

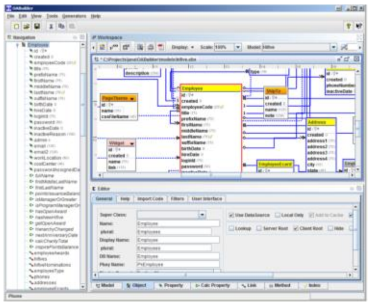
Since OAObject and Hub are observable, and OA includes libraries for meta-data and reflection that allow it to be bound to other components. This can be used for UI type components, for notify/messaging systems, etc.

Building Applications using OAObjects

This is a brief overview of the steps for creating software using the OAObject Class.

Create the Object Model
Using OAObject as the Base Class, add properties and methods to new Classes. ViaOA has a visual designer tool named ***"OABuilder"***. OABuilder can be used to design and generate Objects/Object Models, along with Database DDL script, server, client applications, and web applications.

oa.builder



Creating DataSource
If using a SQL Database, OABuilder can be used to generate a ddl script to create a relational database and a mapping object named "DataSource.java", which is the **ORM** related information that allows a relational database to be object oriented. This is the only object needed to allow your Objects to automatically work with a database. Other OADataSource Classes are also available with OA, to allow for creating distributed DB access, legacy access, etc.

GUI Development
Create Server and Client Applications using the Java Foundation Classes that now automatically work with OAObjects. The first step is to configure your OAObjects and Hubs into a model (mvc pattern), and then bind with UI components. OABuilder also generates these models, and the server and client applications.

Create Web Application using Java Server Pages and the OA JSP components for building pages, forms with Ajax enabled components - similar to building a client GUI application.

Create as Client/Server Application. An application can be made to work across multiple computers, where Objects are kept "in-sync" on all computers. OABuilder will generate this.

Quick Application Overview

Description – shopping cart application that allows user to select a product from company Abc.
Requirements – user can select an item and have it shipped to them. Abc needs to get the order, produce the item, ship it and maintain customer communications.

What OA offers – using OABuilder to design the model, with the following information as an example: admin user, customer, items, item pricing, orders, order items, order status.

OABuilder will generate the Java DAO (OAObject) classes,ORM layer, Database scheme DDL, server with embedded webserver, client Admin UI application with search/filters/reports, deployment webpage, and the Model classes.

The web application can then be built using the Model classes to bind with any HTML element using the OAJSP ajax components. Use calculated properties to maintain open orders, OAMail to send personalized emails, attachments (pdf, html).

The generated Admin client is used to manage all aspects of the application.

Agile?

Agile development is about the flexibility to make changes with the confidence that it will not break the existing application.
With OA, making design changes and adjustments is easy, and encouraged. Once application development is started, more ideas get generated that result in a better end product. There is little cost in refactoring, and making the "backend" changes are pretty much all produced by OABuilder.

Chapter 2 OAObject Details

Overview

Please see the online javadoc for OA at www.viaoa.com, under "*Products"* menu item.

OAObject has built-in functionality that makes it easy to bind (*interface/plug-in)* to any other Class. Once another Class is set up to work with OAObject, then any Subclass of OAObject will automatically work with the interfaced Classes.

The OA library includes reflective functions that make it easy to create reusable functionality for any OAObject/Hub – which then allows it to work with any OAModel.

Concepts and Functionality

This section will describe the concepts and functionality built into OAObject.

Object Graph
An object graph is a way to group objects together, based on collections of objects and the references to other objects and collections.
Hub Collections
A Hub Collection is a group of objects. OAObject works directly with the Hub so that the Hub can help manage the objects. An OAObject can be a member of more than one Hub.

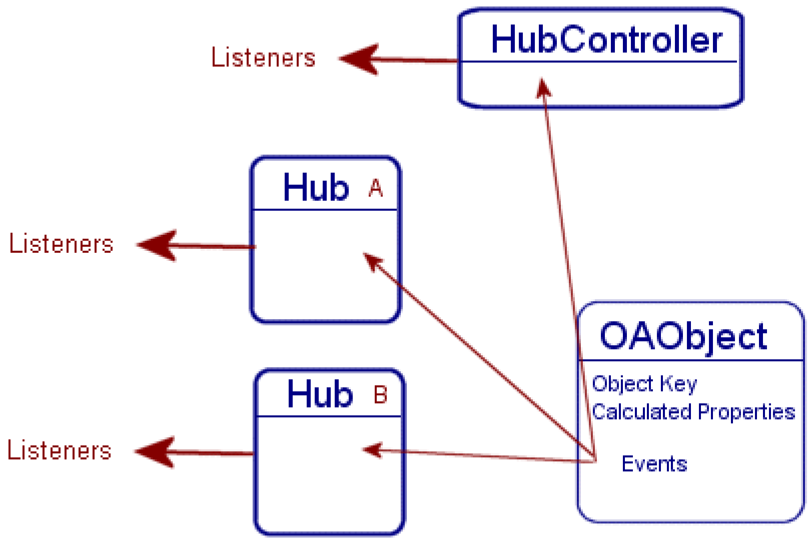
Cache
OA includes a single static object named OACacheDelegate, which is used to track all OAObjects and offers methods to find any object that has been created. It is also used to make sure that an object has not already been created and that an object is unique from other objects in its class.

Object Key
OAObject allows classes to define which property/properties are unique for the objects in the Class. OAObject manages changes to these properties to make sure that the values are valid and unique.

Object Information (meta-data)
OAObject Classes have information describing information about a Class. This includes the properties for the Object Key, information about references to other objects, and calculated properties.

Reflective access
OA includes a strong reflection library that uses the OAObject metadata to reusable methods for accesses and working with model objects.

Events
OAObject manages all aspects of the properties for a subclass. As changes are made to an object and its properties, OAObject will send events to notify other objects of changes.



Property Paths
A property path is a String value that refers to a specific object or property from an object. An example, “manager.department.name”, could be used from an Employee object to get the department name that the employee belongs to. Property Paths can be used to represent any property value.

Calculated Properties
A calculated property is a property within a class that is based on a calculation or formula, where its value is not stored. A calculated property can automatically be recalculated whenever any of the data that it uses is changed. Information about properties (using property paths) that a Calculated Property uses can be defined in the Object Information.

Reference Objects
A reference object is an object that is directly related to an object, creating a relationship. A reference could be a single object or a collection of objects.

Type of Relationships
This is a listing of the types of relationships that a Class can have with another Class. This information is built into the object information. Relationships between objects are “two-way”, meaning that both objects are related to each other.

- One-One relationship
- One-Many relationship
- Many-Many relationship
- Recursive – this is where an object can have many children objects of the same class and each of these children can themselves have children, recursively.
- An Owned relationship is one where the children cannot exist without the parent (owner) and all are treated as a single unit.
- Cascading Rules – a relationship can have rules that describe how related objects are affected when an object is cancelled, saved or deleted.

Managing Relationships
OAObject manages the relationships between objects. OAObject is responsible for retrieving and populating reference objects and for managing changes. An OAObject subclass does not have to have any code to handle retrieving or storing reference objects, OAObject does it completely. If a reference property is changed, then OAObject manages the change so that other objects are updated correctly.   For example, if an Department has many Employees, and an Employee has one Department: if an Employee's Department is changed, then the Employee object is removed from the original Department collection and added to the new assigned Department collection. This also works when an Employee is added to a different Departments Employee collection – the Employee's Department property is changed to the newly assigned Department.

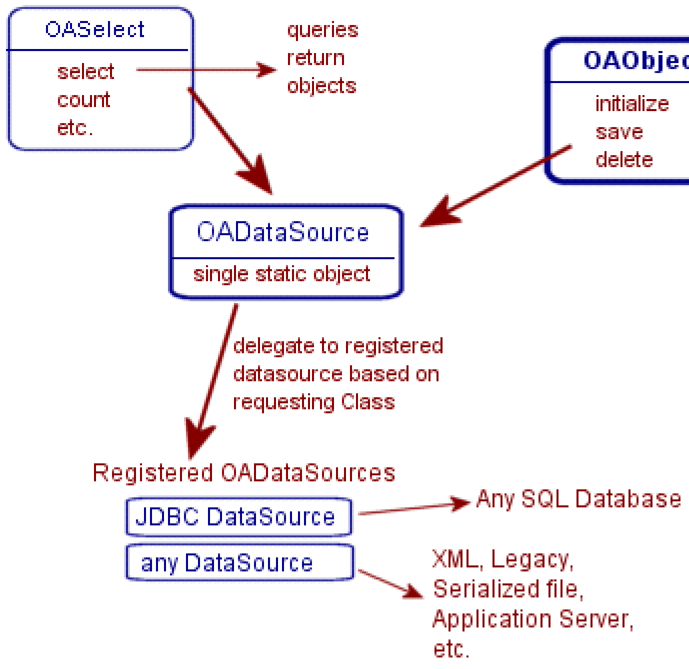
Sequence Properties
OAObject subclasses can define a property that is used to keep the position of the object within a collection. This property will automatically be updated as its position in the collection is changed.

Property Types
Properties for an OAObject subclass can use any type of variable, including primitive types.

Working with DataSources
An OADataSource is an interface that describes methods and services for accessing and storing data. This includes Relational Databases (using JDBC), XML, json, serialized files, application servers, etc. **OASelect** can be used to select OAObjects using a query language based on the Object Model structure, Property Paths and returns results as OAObjects.

OAObject is independent of a DataSource. Applications can be built without a DataSource. DataSources can be added or changed anytime without changing the application. OADataSources are created and registered and then managed by a single static OADataSource Object. OAObject calls on this OADataSource to do various functions like initializing properties, saving, deleting.

OAObject reference properties are based on the actual object. Some DataSources use the concept of a "foreign key" to store the key to a reference. OADataSources that use fkeys store the value of the key in OAObject properties and use them to retrieve/create the reference object when the reference object is requested.



Initialization
An OAObjects properties are all initialized when the object is created (constructed). OADataSource also performs any initialization, which could allow for auto generated object ID properties.

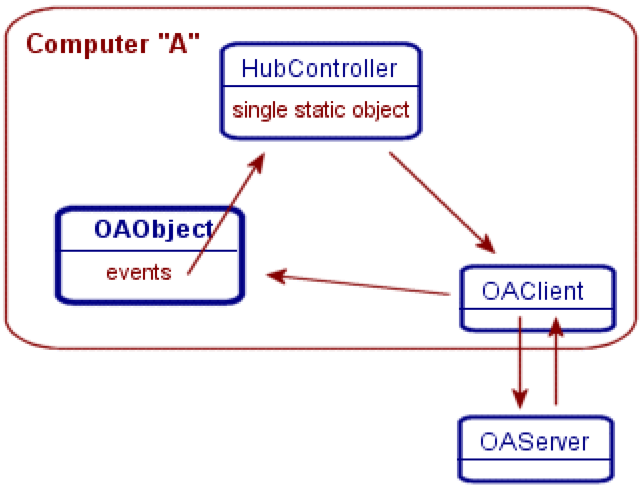
Null Values
OAObject has methods to mange properties that have a "Null" value. This is needed so that primitive data types can have a way to represent that their value as a Null.

Changes
Since OAObject manages changes to the properties of an object, it is able to manage changes that have been done to these properties.

Cascading
OAObject allows for cascading to other reference objects. OAObject makes sure that objects are only "visited" once when this is being done.

Serialization
OAObject handles serialization of objects so that it is as efficient as possible. OAObjects can automatically work with other computers and applications and uses object serialization to transfer objects. OAObject uses "*smart*" serialization so that as little data is sent as possible. Extra support is added to OAObject to allow for controlling how and what is serialized. This is a key part of how OA can use messaging to keep all computers in-sync. OA has features to provide very light weight object serialization that allows it to scale where many computes can act like "one system".

Client/Server capabilities
OAObjects can automatically work with an application running on other computers. To set this up, all that is needed is to have one of the applications create an OAServer object and for the other applications to create an OAClient object. Once the OAClient is created, OAObject automatically works with the OAServer and all applications are kept in sync with each other. To set up this type of Client/Server environment, a separate server is not required. Any one of the computers can run the OAServer Object. The OAServer object is responsible for managing and keeping all OAClients synchronized. The objects on each client will always have the same value, but the applications can each be performing different functions.



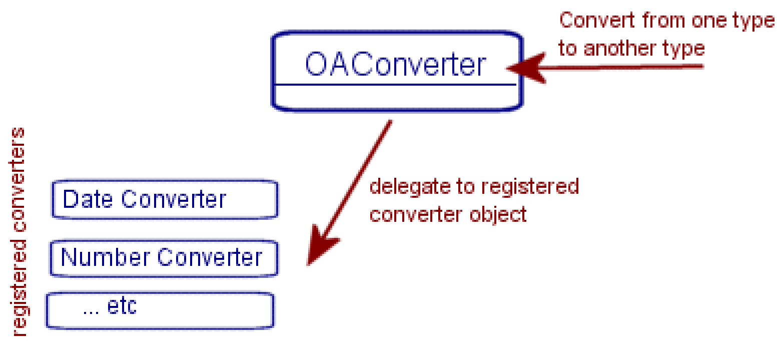
XML Support
OAObject manages reading and writing XML and makes sure that an object is only completely outputted once. When the object is referenced again in the same output, it will only output the object key. *Please refer to* OAXMLReader and OAXMLWriter for more information.

Objects and references can be controlled to know the boundaries of the data that is being serialized.

JSON Support
Similar to XML, OAObject manages reading and writing JSON.

Miscellaneous Data
OAObject allows for storing temporary data in the form of name/value pairs. This is useful in applications where you might need to store short term data specific to the instance of the running program.

Data Type Conversions
Since it necessary to convert property values to/from of variable types (esp. Strings), OAObject has built-in methods for handling this. A separate class "OAConverter" is called to handle this. OAConverter is a single static object that allows custom converters to register themselves to handle converting to/from one value to another type. Also included is the formatting to use for conversion.


OAConverter
Convert from one type to another type
delegate to registered converter object
registered converters
Date Converter
Number Converter
... etc

"Hooks"
OAObject has methods that can be "hooked" or replaced so that custom features can be added.

OAObject Methods

This section will cover some of the methods that are in OAObject.

OAObject Constructor
Adds object to OAObjectCacheDelegate
Calls OADataSource to initialize object properties

createNewObject
Method that can be used to create a new object. If an OAClient is being used, the object will be created on the server computer.

isLoading, setLoading
Used to know if an objects properties are being loaded from another source (ex: datasource, xml file, etc.).

isServer
Used to know if this is a server or client application.

put/get/removeProperty
Methods to store and retrieve any property based on the name of the property. Property name is case insensitive. If the property name is an actual property defined in the object, then it's

get/setMethod will be called, otherwise the objects miscellaneous properties will be used. These methods are overloaded to provide various ways for calling them.

getPropertyNames
Returns a String array of property names

getPropertyAsString
returns a property value converted to a String. The format for the output can also be supplied.

isNew
OAObject "knows" if an Object is "new", meaning that it has just been created and has not been saved.

Is/setEditable
Flag that can be set to know if object can be changed. Note: this is only a flag for other code to use and is not enforced by OAObject.

isMethodEnabled
returns true by default. Note: this is only a flag for other code to use and is not enforced by OAObject.

getChangedProperties
returns String array of all property names that have been changed since the object was created or last saved. Another form of the method accepts a parameter to tell it what type of changes to return: this object only, cascaded reference objects, owned objects, etc.

isChanged
flag to know if the object has been changed since it was created or last saved.

copyInto
copy this object and all of its non-key properties into another object. Note: this will not copy any reference objects of type "many" (references that return a Hub Collection).

firePropertyChange
Used when a property is changed to a new value. OAObject will send the event to all of the Hubs that it is a member of, and to the OAObjectCacheDelegate, which will then send the event to listeners that have been registered – including other systems in a multi-system application.

getObjectKey
An Object Key is used to uniquely identify the object so that it is unique within the Class. OAObject uses and manages an object OAObjectKey to do this.

getPropertyNames
Returns a String array of property names for this object. Does not include Misc Properties.

is/set/removeNull
Used to set a property's value to Null. Primitive data types do not have a way to know if their value is Null, so this is a way to flag them as null.

getHub
Used for reference types of "many" that return a Hub Collection. This method uses information about the reference and will create a Hub Collection for the correct Class. It will then call OADataSource to get the objects and populate the Hub with those objects.

An example
```
    public Hub<Address> getAddresses() {
        if (hubAddresses == null) {
            hubAddresses = (Hub<Address>) getHub(PROPERTY_Addresses);
        }
        return hubAddresses;
    }
```

Notice that there is no code to specifically communicate with the datasource. OAObject.getHub will work with OADataSource

getObject
Used for reference types of "one" that return an object. This method uses information about the reference and will then call OADataSource to get the object.

An example
```
    public Location getLocation() {
        if (location == null) {
            location = (Location) getObject(PROPERTY_Location);
        }
        return location;
    }
```

recurse
method that can be used to access all reference objects that are "connected" to it.

save
saves this object and all objects that are defined to cascade with it. It first calls the getCantSaveMessage method to verify that the object is allowed to be saved. Before actually saving, it will call the beforeSave method, then cascadeSave. When it is being saved, OAObject knows the order to save and how to handle any conflicts. The method onSave is then called to do the actually saving of the object. It will call OADataSource to save the data and will then set the object as not being new and set it as not being changed. It will then call the afterSave method.

Notifies listeners with before/on/afterSave events.

delete
deletes this object and all objects that are defined to cascade with it. It first calls the getCantDeleteMessage method to verify that the object is allowed to be deleted. Before actually deleting, it will call the beforeDelete method, then cascadeDelete. The method onDelete is then called to do the actually deleting of the object. It will call OADataSource to delete the data, remove object from all Hubs, and will then set the object as being new. It will then call the afterDelete method.

Objects that have a reference to the deleted object will have the reference set to null.

Notifies listeners with before/on/afterDelete events.

getCantDeleteMessage
Method that will return an error message if an object cannot be deleted. This method will call all reference objects that are defined to be cascaded.

beforeSave, onSave, afterSave
beforeCancel, onCancel, afterCancel
beforeDelete, onDelete, afterDelete
Methods that provide "hooks" for subclasses to add specific code. All of the "before" & "after" methods do nothing by default. The "on" methods are where the actual procedure is performed.

lock, isLocked, unlock, getLock
Used to flag an object as being locked. OAObject uses static methods in a class named OALock, that handles the actual locking of the object. Note that by locking an object, it does not restrict access to the object. It is used as a flag for applications to use.

isRequried
method used to find out if a particular property is required.

find
Using a property path from this object, find the first object that has a matching value.

findAll
Using a property path from this object, find all of the objects that have a matching value.

Chapter 3 Hub Collection

Overview
This chapter describes in detail the functionality of the Hub Collection Class. A Hub Collection is used to manage a collection of OAObjects.

Introduction
This section will describe the functionality built into the Hub Collection Class.

Collection Class
Hub has same methods as both an ArrayList and HashMap, making it easy to learn and powerful to use.

Observable
Just as OAObject is, the Hub Class is observable, meaning that events are sent to listeners whenever changes are made to the Hub or any of the contained Objects. The HubListener and HubEvent Classes are used for handling listeners and events.

Navigational
Hubs have methods that allow it to be navigated. This is primarily used when using Hubs with GUI components. To do this, Hubs have an *Active Object*. An active object is a reference to the object in the collection that currently has the focus. GUI components that work with one object at a time can then *know* which object to display/edit.

Model/View/Control (MVC)
Hubs act as the Model when developing UI applications. Hubs can be configured so that changes made to one Hub automatically change other Objects or Hubs.

GUI components that use the Objects/Hubs are then controlled by the Objects/Hubs and will directly work together.

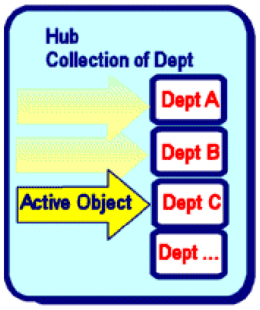
Hub Internals
The *secret* to how the Hub works is that inside the Hub there are three objects that are used to connect Hubs together to form relationships.

- Data – An ArrayList and HashMap are used to store the objects. This can be shared/used by other Hubs.
- Unique info – information that is unique to a single Hub. Ex: the registered listeners. This is not shared/used by other Hubs.
- Active info – keeps track of the object within the Hub that has the current focus. This can be shared/used by other Hubs.

By allowing Hubs to share the same data, it is easy to use the same data for multiple purposes, like multiple views.

Master Object
A master object is the object that a Hub belongs to. An example is that a Department object that has a Hub of Employee objects will have the Department object as the master of the Employee Hub. The objects in this Employee Hub “belong” to this Department. If an Employee object is added/inserted to this Hub, then its Department reference will be set to the Department object that is the master of the Hub.

Active Object – An active object is a navigational feature that is used to know which object in the collection has the current "focus". It does not restrict accessing other objects in the Hub, but is used by GUI applications, etc to know which object should be displayed.

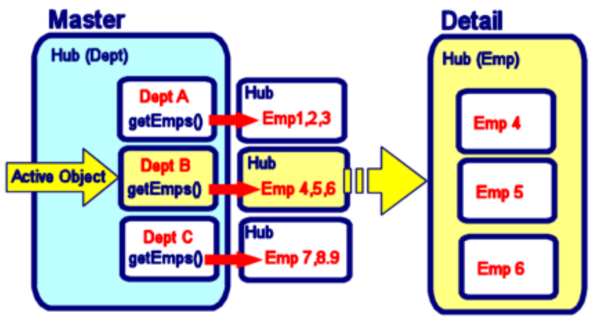


Wiring
Hubs can be configured to work together so that changes to one can automatically update other Hubs or Objects. This includes Sharing, Master/Detail, Linking, Filtering – all described next. All of these relationships are based on the structure of the OAObjects.

Sharing
Navigational feature where Hubs can be created that use the same Data from another *Master* Hub. Shared Hubs are useful in GUI applications where the same collection of objects is needed for different things. Example: A JTable that uses a Hub of Departments display and maintain all of the Departments, and a shared Hub could be used in a drop down list to select the Department for an Employee. Both Hubs would be using the same Data, but for different purposes.

Master/Detail
Navigational feature used to synchronize two Hubs so that they work together automatically based on changing the Active Object in the Master Hub.

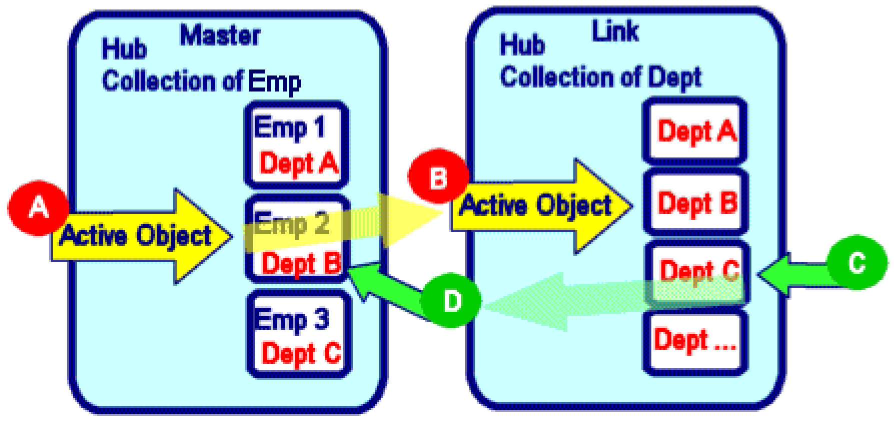
A Hub can be created that is a detail of another Hub. This detail Hub will automatically be populated with objects from the active object of the Master Hub. Example: if a Department has many Employees, then a Hub of Department objects can be used to create a Detail Hub of Employee Objects. The Detail Hub will automatically be populated with the Employee objects from the active object in the Hub of Department Objects.



Using the diagram, the Detail Hub is populated with the Employee Objects from the Department that is the Active Object in the Master Hub. Actually, the Detail Hub is not really *populated*, but rather it uses the same Data that the Dept B Employee Hub is using. If an Employee Object is added to the Active Department's Employee Hub, the Detail Hub would also contain this Employee. A Detail Hub never has its "own" Objects, it is always *sharing* the Data from the Active Object in the Master Hub.

In this example, a UI Component (ex: Table) could be setup to list the Department Objects and another UI Component (ex: Table) could list the Employee Objects of the Department that is selected. If another Department is selected, then the Table listing the Employees will show that Departments Employee Objects.

Linking
Navigational feature where a Hub (Master Hub) can be linked to a reference property of the active object in another Hub (Linked Hub).



Example:
The Master Hub on the left is a collection of Employee Objects. The Link Hub on the right has Department objects.
if the **(A)** Active Object in the Master Hub is changed to "Emp 2", the **(B)** Active Object in the Link Hub is changed to "Dept B", since that is the Dept for "Emp 2".
If the **(C)** Active Object in Link Hub is changed to "Dept C", the **(D)** Dept for the Active Object in the Master Hub will be set to "Dept C".

This is a common setup when using ComboBoxes. In the above example, a form that displays an Employee would have a ComboBox that is a dropdown list of Departments to choose from. When the active object in the Employee Hub is changed, the Employees Department is retrieved and used to set the active object in the Department Hub – this will then display the correct Department in the ComboBox. When the user selects a different Department using the CombBox, the active object in the Department Hub is changed, which automatically changes the Employees Department.

Recursive Hubs
A recursive Hub is a Hub where each object has children objects of the same class. Each object has a method to get its “parent”. A ”Root Hub” is the top Hub where all of the objects in it do not have a parent (value is null). Hub and OAObject will automatically put objects in the correct Hub based on the parent value. If another Object owns the Hub, then all children under it will have a reference to the owner object.

Hub Filtering
A Hub can be created that filters objects from another Hub.

XML Support
Works directly with OAXMLReader/Writer to read/write XML.

JSON Support
Works directly with OAJSONReader/Writer to read/write JSON.

Serialization
Works with OAObject to handle serialization of objects to a stream.

Works directly with DataSource
The Hub Class has methods to directly select objects from a DataSource. Hubs are also set up to only *pre-fetch* a certain number of objects at a time, so that response is faster. Methods to get a total count of objects and to load all objects is also included.

Manages Objects
A Hub can automatically update a property in the objects that is the position of the object within the Hub. This can then be used as the sort property so that the order of the objects is maintained.

Hub Listener

The Hub Listener is used to "listen" to events from the Hub collection and any of the objects in the Hub. This is the single observable listener used for any OA object or collection.

The class includes all of the methods needed to know when a change has happened. Along with the Hub's active object, an application can have all of the information it needs through callbacks/events.

Methods

constructor
creates a new Hub to be used for containing Objects for a specific type of Class

cancel
removes any objects that have been added, adds objects that have been removed, calls cancel method for all objects.

getChanged
Method to know if any object has been changed or if an object has been added or removed.

getAdd/RemovedObjects
Returns a Vector of objects that have been added/removed.

copyInto
copies objects into an object array, or another Hub

setCacheSize
can be used to set the maximum number of objects that the Hub can contain. Once this limit is reached, unchanged objects are removed from the top of the Hub.

size/getSize
used to get the total number of objects that are currently in the Hub. If objects are being retrieved from a DataSource, then *getCount* should be used to get the total number objects that will be loaded.

getCount
If objects are being loaded from a DataSource, then this will return the total number of objects that will be loaded from the DataSource query. Otherwise, this method will return the total number of objects in the Hub.

Is/hasMoreData
If objects are being loaded from a DataSource, then this will return true if there are more objects that will be loaded.

get/setAllowNew
Flag to know if objects can be added to Hub. Hub will not allow objects to be added or inserted if this is false.

get/setAllowDelete
Flag to know if objects can be deleted. This is only a flag, Hub does not enforce it.

get/setAllowEdit
Flag to know if objects can be edited. This is only a flag, Hub does not enforce it.

get / contains
Methods used to retrieve an object by using the object or an object key value. Example: if the object key for a Class is an “int” then get(12) would return the object with key property equal to 12.

getObject / elementAt
get an object at a specific position within the collection – uses the internal ArrayList. Returns null if position is out of scope. This is a good method to use to loop through all of the objects in a Hub. Once this method returns a null value, then the loop can be exited.
Hub also supports the for next operation.

save/saveAll
Calls save on all changed/new objects in the Hub. If the any objects have been removed and are owned by a master object, then they will be deleted when the owner is saved.

delete/deleteAll
Calls delete on all objects in the Hub.

clone
Creates a new Hub with the same objects and settings.

loadAllData
If objects are being loaded from a DataSource, then this will load the remainder of the objects..

getActiveObject / getAO
navigational method. returns the current active object – object that has “focus”.

setActiveObject / setAO / setPos
navigational method. set the active object or the position of the active object.

getRootHub
if this a recursive Hub, then this will return the root Hub. This is the Hub where the objects do not have a parent.

getPos / indexOf
Find the position of a specific object within the Hub. The parameter can be an object or the value of the object key.

setAddHub
This can be used to have the active object automatically added to another Hub whenever it is changed. This is so that as a user selects objects in one Hub, they are automatically added to another Hub.

isOwned
Used in Master/Detail to know if the Master Hub object “owns” this Hub.

add / addElement
Add an Object to this Hub (to bottom of list). If the Hub has a Master Object, then the added object’s reference property will be changed to the master object.

swap

switch position of two objects in Hub.

move
Move an object from one position to another.

insert / insertElementAt
Insert an object at a specific position within the Hub. If position is greater the Hub size then object is added to Hub. If the Hub has a Master Object, then the added object's reference property will be changed to the master object.

remove / removeElement
remove an object from the Hub. This does not delete the object. If the Hub has a master object, then the reference property for the removed object is set to null.

remove / removeElementAt
remove the object at a specific position. This does not delete the object. If the Hub has a master object, then the reference property for the removed object is set to null.

setNullOnRemove
navigational method. if true, then set active object to null when active object is removed. Otherwise, the next object is made active.

removeAll / removeAllElemetns / clear
remove all objects from Hub. (see remove above)

createShared / createSharedHub
returns a Hub that has the same data as this Hub.

getDetail
returns a Hub that will act as a Detail Hub. This Detail Hub will automatically share the same Data as the Hub for the active object. Whenever the active object is change, the Detail Hub's Data is changed so that it is shared with the new active object's Detail Hub.

addHubListener
add a HubListener to a Hub. All OAObject and Hub events will be sent to listeners. This method can also be called to include a property path. If the property is a calculated property, then listeners will be set up for all of the properties that the calculated property uses, so that the property can be automatically refreshed/recalculated.

removeHubListener
Removes a listener from receiving HubEvents.

get/setSelectRequiredWhere
This method is used to add an additional where clause to any select that this Hub performs. This is useful for restricting all queries that use the Hub.

get/setSelectWhere
where clause to use for select to DataSource. Query uses property path(s) based on object structure.

get/setSelectOrder
order clause to use for select to DataSource. Query uses property path(s) based on object structure.

setAutoNumber
property in object that is updated to the value of the position of the object in the Hub.

setAutoMatch

Advanced method used to automatically create objects in a Hub with a reference property that is equal to objects in another Hub.

sort
can be used to sort objects by one or more properties/property paths or by supplying a Comparator object. Objects are kept sorted whenever changes are made. Objects that are added or changed are automatically put in the correct position.

cancelSort
cancels keeping objects sorted.

findFirst, findLast
sets active object to first/last object in Hub that has property/property path equal to property value to find. Set to null if not found.

findPrevious, findNext
sets active object to next/prev object in Hub that matches same as findFirst/findLast.

find
finds first object in Hub that has property/property path equal to a value. Set to null if not found. Does not set active object.

select
used to select objects from a DataSource. Query String uses property path(s) based on object structure.

selectPassthru
used to select using a query that is native to the DataSource.

setLinkHub
Links this Hub to a property in the active object of another Hub.

Chapter 4 OADataSource

Overview

This chapter describes the functionality of the OADataSource Class. OADataSource is an interface for querying, storing and updating OAObjects. The query language is based on the object structure, and is usually mapped to the datasource. OA includes a datasource for relational databases that use JDBC, and an object relational map (ORM) that can be created using OABuilder. Other ORM tools can be used.

Using OADataSource, programs can be built that are totally independent of the datasource, and can treat any datasource as object oriented.

Introduction

A common theme throughout the documentation on OA is how "*everything works with your objects*". This document describes how your objects can automatically work with any datasource/database.

A datasource is any form of service that provides persistent storage and retrieval of data. The most common datasource are databases that use SQL and have a JDBC driver. Other types of storage could be memory, distributed, flat files, XML, JSON, REST, streaming, application servers, web services, legacy systems, object-relational engines, etc.

The OADataSource Class offers the functionality to query a datasource based on the structure of the Object Model and not on the Datasource structure. Results from a query are returned in OAObjects instead of data or resultsets.

OADataSource has methods to save and delete Objects. The OAObject save and delete methods directly use OADataSource.

Applications can be created that are independent of the datasource. Objects are automatically retrieved, stored and updated without requiring specific code relating to the datasource/database.

Concepts and Functionality

Central Datasource control Object
OADataSource is an abstract class that has static methods used to offer services to OAObjects.

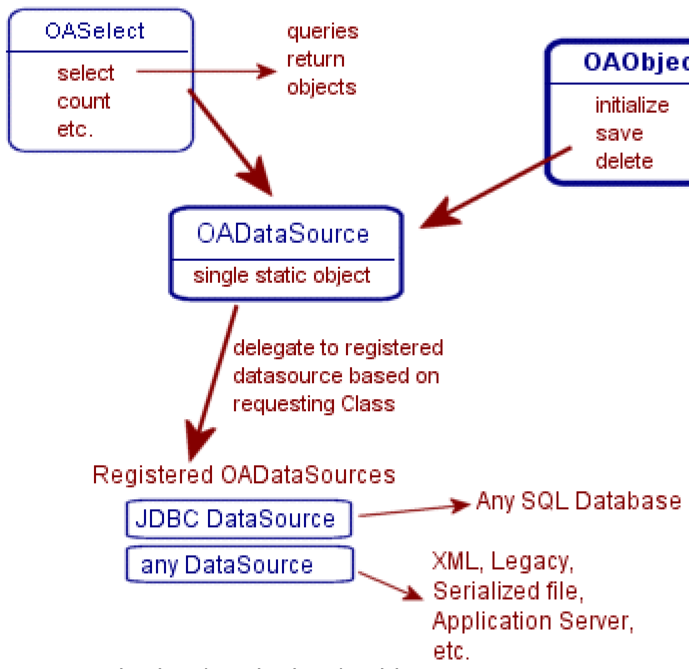
Registered DataSources
OADataSource subclasses are created and registered with the static OADataSource. When a datasource operation is requested, the static OADataSource finds the registered datasource that can handle a request.

Querying Objects
Queries are based on the structure of the Classes (object query language - OQL), using property paths, and not on the datasource structure/language. OADataSource also has methods that can be used to directly pass a datasource specific query. Queries return an iterator that is used for retrieving OAObjects one at a time.

Saving and Deleting OAObjects
The OAObject Class has built-in methods to initialize, save, and delete an object. OAObject works with OADataSource to perform these operations.

***OASelect** query object*
The OASelect Class can be used for creating datasource independent Object Queries. OASelect manages a query by working directly with a OADataSource. The Hub Collection methods for select use OASelect to perform its queries. Note: OASelect is a convenience object and is not required for querying an OADataSource (OADataSources can be used directly).

This diagram shows how other classes use OADataSource:



Auto numbering / assigning / guid
OADataSource can automatically create autonumber values for object properties. A GUID property value is a String value that uses a machine specific id concatenated with an autonumber value.

No Duplicates created
OADataSources work with OAObjectCacheController to make sure that a duplicate object is not created. If a query returns an object that has already been created, then the OADataSource will use the current object created. OADataSource has settings for controlling how the data is updated and allows for managing conflicts – when data stored in datasource has been changed outside of the running application.

Logging Activity
OADataSources have methods to be able to capture all activity to a stream/file.

Methods

getDataSources
static method that returns an array of all registered OADataSources

getDataSource
static method that returns an OADataSource for a specific class.

setGuid
seed value for creating GUID type property values. A sequential number (autonumber) is used with this guid seed to create unique ID values.

getObject
Using the Object Id value, this will create and return an object.

get/setAssignNumberOnCreate
if true, then autonumber properties are assigned a unique ID when the object is created. If false, then the ID is created before the object is saved.

isClassSupported
method to “know” if an OADataSource supports a particular class.

updateLinks / deleteLinks
called to save/delete many-many relationships. Some datasources (like databases) use a “link” table to manage many-many relationships.

insert
save a new object to datasource.

update
save/update an existing object to datasource.

delete
remove an object from datasource.

save
helper method that will call insert or update depending if the object is an OAObject and isNew = true

count
returns the number of objects that would be returned from a query. Query based on structure of object model, and not the structure of datasource.

countPassThru
Same as count, except that query is based on the structure of datasource. The query is “passed thru” to datasource without being converted from object query language to datasource language.

select
used to query datasource for objects. Returns an iterator that is used to return objects one at a time.

selectPassThru
Same as “select”, except that queries are based on structure of datasource and not OQL.

execute
Perform a datasource specific command.

initializeObject
Called by OAObject constructor to initialize an object.

enableLogFile, createDefaultLogFile, setLogFile, createLogFile
Methods to set the output stream that records database updates, inserts, deletes.

Implemented OADataSources

OADataSource is set up so that it can be extended to work with any “back-end” datasource/database.

An OADataSource has all of the methods for working with a datasource in an object oriented way.

We include OADataSources for JDBC, memory datasource, and a client datasource that will access the server using distributed method calls.

OADataSourceJDBC
OADataSource that uses JDBC to work with objects and a SQL database. OADataSourceJDBC uses an Object/Relational Map to work with databases.

Designer Tool Support
OABuilder (visual object designer/code generator) currently generates SQL Scripts and a Java mapping object (ORM). OABuilder can also generate a SQL script to update an existing database to match changes made to the Object Model.

Connection Pooling
Methods to allow for controlling database connections and statements.

Converts OQL to SQL
All queries based on Object Model (OQL) are converted to SQL.

Many-Many support
OADataSourceJDBC uses link tables to manage many-many object relationships

Foreign Key management
OADataSourceJDBC uses Object ID property to match with foreign keys.

OADataSourceClient
OADataSource used for Client/Server programs so all Client datasource requests will automatically use OADataSource running on the Server computer.

OADataSourceAuto
OADataSource that will only generate and assign autonumber properties. Does not directly support persistent storage of objects. This is usually used as a *fake* datasource when objects are stored in a binary or XML format.

OASelect Query Object

The OASelect is the “front-end” for creating a Query sent to an OADataSource. OASelect works directly with OADataSource methods to handle any query. The Hub Collection uses an OASelect object for the Hub’s select methods.

OA Object Relational Mapping (ORM)

OA allows for working with any “back end” database, the most common being relational databases with JDBC access. The OADataSourceJDBC is included in OA and allows for working with any JDBC driver. The OADataSource interface methods are all supported and allow for select, save, delete, which in turn will use JDBC to communicate with the database/s. The ORM layer is done using annotations and an OABuilder generated DataSource.java file.

The OAModel is not dependent on this ORM, and can use any other ORM or even call JDBC directly and use resultsets, etc. We have used Hibernate, IBATUS, TopLink, legacy, etc in other projects where the DataSource was kept total independent of the objects. For example: if using JDBC directly, you could create and populate an Employee object (OAObject subclass) directly, etc. With the OA ORM, you have a more natural object oriented solution, where you never directly have to write data access code.

Chapter 5 Utilities

OA includes helper methods and classes that work with Java to extend and include additional features.

The **OAString** class has static methods for working with Java Strings, parsing, formatting, etc.

The **OAFile** class has static methods for working with Files.

The **OAImage** works with Images, loading, saving to bytes, and other images types.

The **OADateTime, OADate, OATime** classes greatly simplify working with Date/Time values.

Conversions, formatting
Converting from one Java type to another is handled by a conversion utility named OAConverter. This includes converting from one type to another, formatting, math functions like rounding, etc.

Protocol Handlers – used with URLs, to be able to access information in objects as url resources. Example: a web page can have an image tag that references a image stored in an OAObject, and also include scaling information, etc. OA includes handlers to access OAObject properties, jar file resources, resources on disk/network.

HTML, CSS support – OA has parsers, tools, and tags that allow for dynamic data to be inserted into a page. The OAHtmlTextPane is a JFC component that can be used to edit/view/print the html. This is used for reports.

Java Serialization – handles object reference boundaries to create *smart* serialization. Support for custom callback to manage what and how references are included. Support for compression. This also handles concurrency issues that can happen as an object is being serialized and changes are actively happen that could affect the objects already serialized. When the receiving client unserializes the objects, the changes are also applied.

XML, JSON – support for reading and writing.

Chapter 6 Property Paths

A key feature within OA is the OAPropertyPath.

A property path is a dot separated value that represents how to get from one object to another object. For example “`orderItem.vendor.state`”, could be used from an Order instance to get the vendors state for the item that was ordered.

These property paths can go as far as needed, and OA will basically reflect on this to get from one end to the other. The OA reflection classes are internally optimized for this.

Since the property path is a String, and is not “*compiler safe*”, there is a method in OAString to create a property path using static values built into the OAObjects, so that the compiler will catch any typos.

OA has various parsers and language tools to convert to other types, along with building trees that define and track changes related to the objects and information defined by the property path.

Property Paths are used in many areas and are a key concept when using Object Graphs and Object Binding.

This includes:

Language independent DataSource queries – the property paths will then be converted using ORM to then access the database/datasource using the datasource language (ex: SQL).

Merging – another key feature is being able to merge all of the data from a property path into a new Hub, that is then kept up to date. A HubMerger can be used that has an initial Hub, a property path that defines what data to get, and a second Hub that will be populated with this data. Internally a tree structure is created that will listen for any changes that affect the collection and then make adjustments. Since the results are in a Hub, it can be used as any other Hub – by UI components, for example, or a list of issues, etc.

Calculated properties – This is a read only property that can have other data that it uses to then get a result. By using property paths to define the data that is needed for the calculation, the calc property can internally listen for any changes and then send a change event when any of the data is changed.

Listeners – using property paths, and the fact that OA is observable, a listener can be notified when an event happens.

Filters – used to be able to update a list and determine if an object should be included in the list. If a change is made some/anywhere that affects this, then an event can be sent that will then have the filter update the Hub collection.

Chapter 7 Annotation and Reflective utilities

OA is a model driven solution that is able to use the features of Java, and extended with Annotations to build a rich level of metadata.

The OA reflective library has the functionality to take advantage of this information to create an observable framework. This is used internally throughout most of the components. It has been optimized to use caching and known patterns to offer high performance.

Chapter 8 Object Graphing

Extending on the observable and meta-data features built into OAObjects and Hubs, OA is about working with the concept of an Object Graph, where the details are handled *under the hood*.

This allows the persistence of the objects, transport, uniqueness and identity to be handled in a way that is object oriented, and makes everything work as if it is one object graph.

Serialization – serializing objects can have issues when there are references, resulting in much more data than expected. OA manages this so that the “boundaries” of the object are known and are managed.

Under the hood Example: A client application is started, connects to server, gets Hub of employee objects from server. The client UI allows the user to have access to any of the data in the system. As the user selects an employee to display the employee orders, the method employee.getOrders() is called. If the orders are not loaded, OAObject requests the property from the server. The server will then call employee.getOrders(). If the server does not have them loaded, then the OAObject will access the OADataSource to get them. They will then be on the server, and serialized and returned to the client. All of this happens automatically under the hood for all references. There are options to easily have the data preloaded, and have it sent to the client on initialization, or when the employee object is requested. This code is smart enough to know what the client has and what to send. For example, in some cases all that is needed is the object key, or only certain references and not all of them.

Chapter 9 OA JFC for rich *Swing* UI clients

The OAJFC components encapsulate the Java Foundation Classes (JFC), also known as "Swing", to make them bindable with OAObjects and Hubs. The Model/View/Component (MVC) pattern is extensively used throughout OA and this component set.

Introduction

Each of the JFC components has been extended to have an "OA" equivalent component. For example, OATextField is a subclass for JTextField that adds binding to any property in the object, or Hub that it is linked to. Whichever is the active object in the Hub is the object that the textfield will be bound to. Changing the active object, will have the textfield automatically display the value.

Concepts and Functionality

The concepts for the OAJFC bindings are similar to the days when data binding was all the rage. The main difference is that the binding is done with the object and object graph, and reflecting real time updates. For UI components, binding, editing and editors, custom rendering, security, etc is automated to work with the object model. Binding is done by associated an OAObject or Hub collection with a component, along with the property path to use. All components offer formatting and custom editors and renders, along with set properties for visibility, editable, colors, fonts, etc.

Also includes drag-and-drop, undo/redo, custom popup menus.

OAJFC Components set at a glance
The complete set of components includes the following, plus more:
TextField, TextArea, Password, CheckBox, ComboBoxes, DateComboBox, TreeComboBox, Buttons with built-in commands, Color chooser, Split button, Label, List, MenuItem, OutlookBar, Popup List/Tree, RadioButton, Scroll, Table, Tree, Dialogs,
Autocomplete functionality.

Table – sortable, option to save setup, freezing columns from hort scrolling,
Tree – supports recursive nodes

OAHtmlTextPane – wysiwyg styled HTML editor, printable functionality to allow for smart/controlled page breaks, headers, footers, Spellchecking, etc.

Printing – print preview, scaling, save to image, save to Pdf, etc.

OAImage Editor – allows common image editing, also included in OAHtmlTextPane for inserting and editing images. Images can be stored in an OAObject property.

OAReport – works with OAHtmlTextPane to display html reports that can be generated using a JSP-like tag language that uses OAObject/Hub as root objects and property paths to dynamically update the report. Smart page breaks, headers, footers, print preview, save as pdf, email support, etc. included.

Chapter 10 OA JSP for web development

Introduction

Set of ajax enabled components that allow binding web pages to application data – the object model.

Web pages can be built where the html tags are dynamically updated by the OAJSP components. This allows the pages created by web designers to automatically work with the "*back end*", without the application developer reworking the pages. There are components to handle a complete set of functionality, and the ability to extend and create new components. These components have callbacks that allow them to make changes to the html elements on a page.

The current component suite uses JQuery internally and does not require developers to code (or debug) and JQuery or Java script code – since the components send it internally. Developers use the components to be able to bind the back end data model to the html elements.

Concepts and Functionality

The OA JSP package includes the following components:
Button, CheckBox, Combo, Dialog, Form, Grid, Image, Link, List, Password, Radio, ServletImage, Table and columns and editors, table pager, TextArea, TextField
Autocomplete, popup dates, error and notify messages.

HtmlAttribute, HtmlElement – for controlling and customizes and html element. This is commonly used to set visibility, animations, or replacing portions of page with dynamic data from back end/model.

There are session related objects for Application, Session, and Form (page).

Includes support for multipart forms, used for uploading data.

There is backend support for generating images, download data, and pdf files.

Chapter 11 OABuilder model designer

Building the on functionality of OA, OABuilder is a visual development tool for modeling applications. This includes designing the objects, their relationships, properties, calculated properties and methods. This forms a rich set of meta-data that can be used to define and build most of an application.

This information is enough to be able to generate the DAO, ORM, Database, Model, Server, Client and Server UI applications, Reports, security. And since it was built using OA, it can capture model changes, and then generate updates that include database DDL scripts, and regenerate all of the application – while keeping any custom changes that were already made.

Code Generation

One of the strengths of the OA Model is the transition objects that it can create, that can be used by code generation functionality. A model can be transformed into a set of objects that can then be used to dynamically generate any type of program. Currently, OABuilder includes the following code generators:

OAObject class files – these are the DAO objects that are created for each object in the model, which are observable, and used to bind to UI, and automatically work with any datasource and communicated between clients and servers applications.
Search class files – these are non persistent objects that are used by UI applications for inputting query parameters used for searching.
Filter class files – these are non persistent objects that are used by UI applications for inputting filter related information – ex: a date range for a filter that limits the Employees to only show new hires.
Model class files – these are the model classes that can be used to bind with UI applications. The Hubs are *wired* together to create the various master/detail, linking, filtering and other relationships defined in the model.

JDBC DataSource – ORM used so that OA model objects automatically work with database.
SQL DDL Script – complete script used to build the database.
SQL Update Script – DDL script used to update and existing database with any model changes.
Test Data – (not completed) This will be used to generate sample data for testing.
Admin client – UI application that is used to manage all aspects of the data in the model. This uses a very sophisticated set of algorithms that takes the model metadata and converts a matching UI client. This is much more than a crud based application. It handles very complicated relationships and UI scenarios in a consistent an structured way.

Generating a complete project

The above code generation does not include the framework around the application. To do this in a manner that is flexible, we have a Template project that is used as the base, with areas that are dynamically adjusted based on the model.

The **OATemplate** project is a complete application framework *waiting to be told what to do*. Follows the Model/View/Controller (MVC) design pattern.

Server application – startup, UI, manages database, manages client connections and logins, web server, client application JavaWebStart deployment, Logging, Help, about.
Custom processes, remote objects so that methods called on server can be executed on the server.

Client application – client Admin application for accessing the complete object model. Includes login, help, searching, listings, reports, etc.