The OA Object Graph

The core of OA applications is the object graph, which represents the software application data, relationships, business rules, calculations, and metadata. Using objects in a graph is similar to using records in a database, with additional features to make it easier and more automated to work in a distributed environment.

> The concepts are language independent. ViaOA supplies a Java implementation with the open source github project oa-core, along with other libraries for desktop (oa-jfc) and web (oa-web). ViaOA also offers a visual modeling tool “OABuilder” that is used to design and generate full stack and distributed applications.

The motivation behind OA was to automate software development around the object graph, so that more time could be focused on what is being created and less on actual coding.

The idea behind OA, inspired by object databases, was to create an object graph that would be like using a relational database, with features that come with object oriented, observable, reactive, functional, memory caching, and distributed programming.

Like a database – be able to store and retrieve object data. Be able to query, based on objects structure. Be able to assign unique values. Be able to use object references instead of keys.

Relational – objects that are linked together, to create various types of relationships, patterns and rules.

Like object oriented programming – besides the data that can be stored, there is calculated data, methods, and meta-data that can further expand how the data is used and handled.

Navigation – access the object graph using property paths.

Observable – so that the data can be event based.

Reactive – so that data events can trigger other code to be executed. Allows for real time results.

Functional – code that can process through an object graph, using property paths, finder and filter processing. Also allows for real time updates, so that filtered results are always up to date.

Memory caching – act as if all objects are loaded in memory.

Distributed – methods can be called on one server and invoked on another server(s).

In Sync – allows objects to exist in more than one place at one time, but always staying in sync across servers.

Binding - Master/Detail, linking, sharing, filtering, and more. A natural MVC solution for creating UI components and frameworks.

Terminology

Observable object and collection classes – changes to a graph are event based. Property change events, and collection events are handled in a way that allows scaling.

Reactive – since objects and collections are observable, then changes can be done as a reaction to an event. This is how the objects stay in sync with each other, how calculated data is updated, and how objects are updated across servers. Any code can listen and create a reaction to an event. Code can run in the same thread, another thread, thread pool, queued, and/or processed on another server

Property Paths – a dot '.' separated string that defines how to navigate from an object/collection to get to properties or other data. Includes functional syntax that allows for filtering. Used for object queries, filters, binding, functional navigation, etc.

Relationships – how objects are related to other objects. These relationships will automatically stay in sync to form forward and reverse relationships.

Relationship Types - one, many, many-many, recursive, hierarchical. Also includes meta data to further define how the relationship can be used.

Navigating object graph – using a root object or collection, and property paths (support for chaining), to find, filter, merge, process/visit.

Meta Data – further define properties, calculations, triggers, security, access and relationships.

Calculated – data that is based on other data, that can also define any dependencies that would require a recalculation.

Dependent property path – used to define dependent data, so that changes can trigger an event to recalculated. Similar to how a spreadsheet formula works.

Reflective – functionality that allows for accessing any data in an object graph, and to convert data types.

Persistence – allows objects to be created, updated, deleted, retrieved and queried from any type of persistent storage – relational database, interface/API, end point, legacy, etc. Includes support for JDBC, object cache, distributed and custom datasources.

Cache –keep objects unique (no duplications), identifiers, soft references, querying. Options to define object life span, max sizes, clearing, marking as dirty, etc. Allows programming to a*ct-as-if* the full object graph is in memory.

Lazy – “acts as-if” object graph is fully loaded , only load what is needed, but take advantage when objects/data is loaded from I/O bound, by also loading other “sibling” data that can be predicted. Objects define preloading, caching, persistence source of data.

Realtime – based on reactive nature of the object graph, changes can be done at the time of the event, or handled based various types of change handling patterns: flagged, put in queue, batched for time based processing, throttling, etc.

Context – the user, process, system, etc that is currently running a command/thread.

Transactional – allows a thread of activity to be managed within transaction boundaries, and locking.

Security and access – determined by context and any specific model info, used to determine access rights visibility, etc.

Query – uses queries based on object graph and property paths. Will convert to SQL for relational database queries.

Filter – filter an object graph property path, returning the results. Filters and filter chains can be used to find data and for creating real time filtered results.

Merging – collect all objects in an object graph using a property path. Results always stay updated.

Grouping – take a collection of objects and have a new collection by group. Results always stay updated.

Left Join – using a collection and have them grouped based on all of the objects from a group by collection. Results always stay updated.

Aggregating – allows summarizing data into other objects, and having data always updated.

Remoting – methods can be called on one system and automatically sent, invoked and returned by another system. Allows communication independent solutions. Supports sync, async, broadcast, targeted, etc.

In Sync – objects in object graph automatically stay in sync across servers. Uses object/collection observable and remoting.

Object callbacks using Edit Query – fine grained control for objects in object graph to interface with other code. This is used for access, UI controls, APIs, etc.

Serialization – allows for marshel/unmarshel in any format: binary, text, third party, etc. Objects have fine grain control over how and what is serialized, which allows for performant, low bandwidth solutions for remoting, REST, webservices, API, end points, XML, CSV, YAML, JSON, etc.

Delta change control – objects know what was changed, and how to communicate with other systems that are kept in sync

Uniqueness – assigns keys (guid, ID props) and interact with datasources. Used as references when communicating with other systems.

Services – allows for dividing and separating an application into specific function services.

Communication Stack – implementation/solutions for Sockets, Ajax, UDP, Multicast, third party, open source, that automatically allows objects to be in-sync across servers and allow remote method calls between servers.

Recursive relationship – an object can have children, recursively. Includes options to define depth.

Hierarchy relationship – define a relationship based on a (1+) hierarchy of property paths to find the first available value.

Reflection – methods to generically access any information from an object, collection, object graph.

Conversion – conversion utilities to be able to convert from one Java type to another.

Java Serialization – manages how and what data is stored, and what is sent on the wire. Is able to be efficient (small) and performant. Improves generic Java serialization by using custom read/write/resolve and special use of transient.

Serialization – includes support reading/writing xml, json, csv, yaml, and custom.

Desktop UI – complete JFC/Swing component set that automatically works with objects in object graph. Custom renderers, editors. Templating for mouse over, reports, rendering. Styled editor, image editor, report writer, grid/table, and many more.

Web UI – component set that is able to interact with web pages, allowing for any type of dynamic behavior between web pages and object graph. Includes dynamic template engine.

Context – used to define user/API/system identification within the object graph. Implemented uses thread local.

Security and Access – objects define dynamic permissions based on object graph data, and context. Fine grain support.

OA Object Graph Implementation

Definition - "*objects and their references to objects*".
Includes methods to automatically load references, so that an object graph "acts as-if" it is fully loaded.

OAObject - base object class.

- Flags for isChanged, isNew, isDeleted.
- Change events: before/after property change
- object identifier – 1 or more key properties, GUID.
- Internal map to store references to other object(s) – reference/key, soft references.
- Hubs – list of hubs that this object is a member of, so that OAObject events will be sent to Hub.
- Annotations for datasource mapping, references, key
- Edit Query callback – allows outside code to call internal methods for additional fine grain control, like verifying changes, access permissions, etc.

Hub - observable collection class

- Allows adding and removing Hub Listeners, to get collection and object events.
- Allows creating relationships between Hubs, to form master/detail, sharing, filtering, copying, merging, grouping, etc.