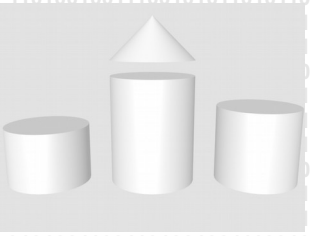

ODABA NG

ODABA Database and tools

Summary

ODABA is a terminology-oriented database system, which might be considered as extension of an object-oriented database system. Adding terminology orientation includes supporting terminology-based analysis, documentation support during all development phases and final document presentation.

From a technical point of view, ODABA fulfills technical standard requirements as locking and transaction mechanisms, several client/server model etc. But ODABA supports even more features, as usual object-oriented databases do (aggregation schema, extensive data exchange features, run-time type extensions, various client server models, various versioning strategies, support for different data storage formats etc.).

Besides technological diversity, ODABA provides terminology-based technologies and development tools for analysis, modeling and design, implementation and test. The ODABA GUI framework supports object-oriented GUI design based on defined database model.

ODABA completely transforms the terminology model into an object-oriented database model, which, again, might be transformed into a relational database schema or into an XML schema without loss of data model information. This allows storing or mirroring data also in relational databases or even in an XML file.

The following topics provide a short overview about ODABA features. More detailed documentation is contained in **User s Guide** and **Reference documentation**.

run Software-Werkstatt GmbH
Winckelmannstrasse 61
12487 Berlin

Tel: +49 (30) 609 853 44
e-mail: run@run-software.com
web: www.run-software.com

Berlin, June 2018

Table of Contents

1 ODABA database features

This chapter considers database features available without using the GUI framework and tools provided with the Object Development Environment (ODE).

ODABA provides many enhanced database features on one hand, and, on the other hand, tries to comply to standards as far as possible. Here, standard features and main extensions (differences to other database management systems) are listed.

The theoretical base for ODABA is Unified Database Theory, which defines abstract database model levels for different types of databases. ODABA combines the flexibility of key/value stores (P_0) with the simplicity of relational databases (P_1), the complexity of object-oriented databases (P_2) and analytical features of data warehouse technologies (P_3).

1.1 ODABA and Unified Database Theory

The ODABA database management system supports different database model levels as being described in Unified Database Theory. Similar levels have been encountered in Terminology Model, i.e. when reflecting subject area knowledge in terms of human language. Hence, ODABA not only unifies different database model levels, but also harmonizes human and technical languages.

Key/value stores

P_0 database models are typically provided as key/value stores. ODABA supports P_0 databases by means of extension properties.

Entity-relationship models

Relational databases (entity-relationship models) are considered to be P_1 database models, which store data in well-structured instances that are collected in tables. P_1 requirements are also fulfilled by object-oriented databases, which may store data in well-structured object instances collected in extents. This way, ODABA also supports P_1 requirements.

Object-oriented database models

Besides an instance schema, P_2 database models allow storing any number of data collections (e.g. children of a person), which is typical for object-oriented databases. This also includes inverse relationships and inheritance, which also are considered as relationships. ODABA supports the requirements on object-oriented databases and is, hence, also a P_2 database model.

Aggregation schema

P_3 database features are typically implemented as aggregation schema (e.g. in data warehouse tools). P_3 database models support set relation schemata in terms of subset/super set relations and (hierarchical) classification schemata. ODABA supports set hierarchies (subset/super set relations) and classification schemata. Thus, ODABA fulfills requirements for P_3 database models.

1.2 Meta-model

The dictionary or resource database containing the application database schema and other application resources is an ordinary ODABA database based on a kind of meta-model. The meta-model describing the resource database covers following areas:

- data model - containing type definitions for defining data types
- documentation model - containing type definitions for documentation objects
- administration model - containing basic administration type definitions (user, rights etc.)
- functional model - containing type definitions for implementation classes, options and external resources.

ODABA is, conceptually, terminology-oriented, i.e. is supports terminology based data modeling (Terminus) and allows creating database models from defined terminology models. In order to provide proper documentation, all development resources are linked with a documentation topic, which is typically maintained while developing an application.

1.3 Database model

ODABA supports all features typical for object-oriented database models as (multiple) inheritance, attributes, references/relationships, inverse relationships etc. ODABA provides several useful model extensions.

All database entries (instances and indexes) contain a 32 byte entry descriptor, which contains version and schema version information, modification count and data base position. ODABA supports 64-bit identities (internal entry number), which supports up to 9 223 372 036 854 775 808 database entries, which means for the moment practically: no limit.

Transient properties

Persistent object types may contain transient properties (attributes or references), which are evaluated when accessing the properties. Transient properties may be defined as simple attributes (like age calculated from birth date), but may also refer to complex data type or collections resulting from complex operations.

Generic attributes

Generic attributes are those, which allow storing any number of values which are presented depending on the current environment. Typically, language depending information are stored as generic attributes.

By means of generic attributes ODABA supports multilingual features (generic attributes) in documentation topics that allow providing documentation in any number of languages.

Extension attributes

ODABA allows extending particular object instances by attribute extensions at runtime, i.e. different instances of a given type may get different extension attributes (one person gets a 'comment' attribute, another 'last journey').

Hierarchical enumerations

ODABA supports hierarchical enumerations (classifications). In contrast to ordinary enumerations, beside value (code) and name ODABA enumerators provide a number of additional information:

- label - language depending name
- title - short description
- description - detailed description
- type - related data type (class)
- condition - constraint applying to object instances of the class (data type) defined by the enumerator

Set relations

Set relations allow defining subset/super set relationships. Extent collection may form set hierarchies of any level. Relationship collections may refer to one super set.

Shared inheritance

Shared inheritance allows two or more instances inheriting from the same base instance (e.g. two employee instances may inherit from/refer to the same person instance, when the person is employed more than once.

Weak-typed collections

Usually, collections (references, relationships, extents) are (strong) typed, i.e. all instances referenced in the collection belong to the same type. Weak-typed collections as supported in ODABA, only require that all instances in the collection are based on a common type (e.g. a weak-typed person collection may contain person, employee and student instances, which all inherit from person). Consequently, ODABA also supports untyped (VOID) collections, which may contain instances of any type.

Instance ownership

The ODABA database model is defined in a way, that for each instance exactly one owner property is defined. This makes it much easier to decide, whether an instance must be removed, only from a collection or deleted completely.

Besides, other options are available for controlling the automatic instance deletion behavior.

Copy model

When copying instances between or within databases, problems may arise, especially, when using deep copy option (i.e. copying an instance and all its referenced instances recursively). Usually, default copy rules are sufficient to avoid unlimited recursions by defining primary and secondary relationships for inverse relationships. Complex database models may, however, cause copy conflicts and require a specific copy model, which defines the sequence of extents to be copies and copy options for relationships.

1.4 Versioning

Different versioning strategies allow freezing the state of single object instances or a complete database. Several versioning strategies may be combined:

- version scope - defines the scope for consistent version numbers (database, object space, owner or instance)
- managed - provides a hierarchy of major and minor versions, where major versions define consistent version states and may be assigned to time stamps.
- synchronized - version numbers are synchronized representing a temporal order of changes

Moreover, online schema evolution allows extending a database schema without reorganizing running databases. Schema version uses managed versioning for the resource database, i.e. creating new schema versions will freeze the current schema version, which allows upgrading object instances at run-time.

1.5 Scalable servers

ODABA may run with or without server - no different for the application. Platform independent database format also allows using databases on different hardware platforms and running applications in heterogeneous environments.

Local applications

ODABA supports access to databases stored on a local computer. There is practically no difference between file server and local applications. In order to support parallel applications on a local machine, ODABA distinguish between shared and exclusive database access, which is just a run-time option.

File server

In order to run ODABA in a local network environment, ODABA applications do not need a database (object) server One may develop and use an ODABA application just referring to a database file on on a shared device in a local network using file server support. This also provides concurrent access to the database. It also means that database files might simply be moved from one environment to another.

ODABA object server

In order to support many users in a local network (LAN), an object server would perform better than simple file server applications. In order to run an application with an object server, the application's configuration file has to be changed slightly and an object server has to be setup. Usually, no changes in the application are necessary.

Replication server

The ODABA replication server allows running databases on an internet server. Thus, clients distributed all over the world may access the same ODABA database. In order to run an application with a replication server, the application's configuration file has to be changed slightly and a replication server has to be provided in the internet. Usually, no changes in the application are necessary.

Replication server access is transaction based and works on a local copy (replicate) of the master database. Replication server is the preferred access mode for distributed clients in the internet. Since reading data happens with local access speed, replication server access is fast as long as update load is low.

ODABA HTTP server

The ODABA HTTP server (`OHTTPServer`) is a mean of communication with an ODABA database via HTTP internet protocol. This technique differs from previous server technologies an is intended for supporting WEB application and App development. ODABA supports all request types and also provides HTTP client features for testing complex requests.

1.6 Database access

Database access works like accessing object instances in an object-oriented environment. The database appears like a huge storage area, which allows accessing data in different ways.

- typed access - allows referring to property and type names as being defined in the database model. Typed access is supported for OSI and C++.
- generic access - allows accessing data via property and value handle, which may form hierarchies. Generic access is supported for .net (C#), C++ and OSI.
- view access - allows accessing (usually derived) data via view definitions and aggregation schemata.

C++ and .NET APIs provide traditional access interfaces to the database, but there is also a scripting language (OSI), that provides direct access to database. OSI is, syntactically, similar to Java or C++ and nearly as powerful.

1.6.1 Access interfaces

ODABA supports several language interfaces in order to access database content in different ways. Access interfaces are provided for C++ (typed and generic access), C# (generic access) and ODABA Script Interface OSI (typed and generic access).

ODABA also supports mixed implementation classes (C# and OSI or C++ and OSI).

1.6.2 Access handles

Database access functionality is provided via handles on different access layers.

- Dictionary - The dictionary or resource database contains database model definitions, but also GUI design resources and application methods.
- Database - The data base contains application data. With one dictionary, one may run any number of databases.
- ObjectSpace - An object space is a well-defined area (scope) within the database, which may contain data for a specific application area (geographical area, client, ...). Object spaces may form hierarchies. The database is the root object space,
- Property - Properties refer to collections or instances and provide a wide area of access functionality.
- Value - is, typically, an elementary data item, but may also refer to complex attributes or arrays.

All access layers are supported by appropriate handle classes with the same name. Access handle are the base for generic data access, which is supported in C++, .net and OSI.

1.6.3 Typed access

Typed access allows accessing properties and values by name as being defined in the database model. OSI supports typed access by nature. In order to support typed access in C++, C++ header files may be generated using ODE development tools. .net does not support typed access.

1.6.4 Views and aggregation schemata

Similar to traditional databases, ODABA provides View definitions as well as ad-hoc views (SELECT statement). In contrast to traditional view definitions, elements of the SELECT statements (FROM, WHERE etc.) may also be referenced as operations in an operation path, which allows more flexible view definitions.

Since views support basic aggregation by the GROUP BY operation, in defining aggregation schemata has been introduced in ODABA just setting an aggregation option in the view definition. Complex aggregation schemata allow providing aggregated data on different aggregation levels.

1.7 Concurrency behavior

In order to avoid concurrency problems, ODABA provides internal and external locking and transaction features. Depending an handle access mode, ODABA forbids or permits concurrent instance update. An automatic update detection discovers and "repairs" conflicts in case of concurrent updates.

1.7.1 Transactions

In order to make concurrent applications save, but also for improving performance (e.g. for maintenance processes), ODABA provides different transaction technologies:

- Internal transactions - very short transactions started while running any update function.
- Read transaction - started in order to block event generation (e.g. while iterating through a collection.
- Pool transaction - Medium transaction storing changes in an internal area
- File transaction - Long transaction storing changes in a file
- Work spaces - Very long transactions storing changes in hierarchical staging areas

In order to restore changes made after last backup, a recovery file may be written, which logs all transactions.

1.7.2 Implicit and explicit locking features

In order to avoid concurrent updates, instances and indexes can be locked (read or write lock) temporarily. Several internal and external lock features are provided:

- implicit locks - are very short locks (mainly for indexes/collections) when being updated. All resources updated within a transaction are also locked until the end of transaction. Also, instances are locked (write lock) implicitly, when being read in write mode.
- read locks - read locks may be called by applications in order to prevent a resource from being read or updated by other processes or transactions.
- write locks - allow preventing an instance or collection from being updated by other processes or threads
- persistent locks - allow marking instance in the database as being locked (write lock) permanently.
- key locks . special lock mechanism in order to reduce write lock time for collections in longer transactions.

There are some other types of locking global resources described in **User's Guide** and **Reference Documentation**.

1.8 Event handling

Database applications generate internal events usually indicating data modifications but also state changes in access handles (open, close, select instance etc.). Events may be received by event handlers (overloaded functions written in C++, C# or OSI), but may also be passed by process event handlers to other application components. Thus, database events are sent to GUI framework (when being used), in order to indicate state changes in data or access handles.

1.8.1 Database events

Events are generated in order to signal different kinds of state transition referring to database resource or access handle state. Most events are designed as before- and after-events. Before-events may be used to suppress the state change in process.

- Handle events - are generated, when opening or closing an access handle or when changing selected data in an access handle
- Resource events - are generated, when database resources change, i.e. while creating, updating, storing or deleting instances or while removing or inserting instances from/into collections. Resource events are also generated when the intended state transition failed.
- Server events - in order to synchronize states between client and server, the server sends events generated while processing a request back to the client, which also generates the events returned.

Handle and resource events are generated on instance and property level, i.e. events are also generated for elementary attributes.

1.8.2 Event handling

There are three different ways of handling database events:

- Overloaded event handler - In order to handle events, context classes have to be defined for corresponding database resource (database, object space, data type, property). Events generated call virtual event handler functions (dummy functions) that may be overloaded in the context class (OSI, C++, .net)
- Process event handler - In order to provide kind of generic event handling, process event handlers may be provided. Process event handler register themselves to a property handle in order to receive property and instance events.

1.9 Accessing external data

ODABA supports accessing external data via property and value handles, i.e. using the same functionality as for accessing database data. External data may be defined as normal data type with some restrictions caused by the external data format. ODABA also allows defining mapping schemata mapping database data types to external data types. Last but not least, ODABA supports data exchange with several external storage types.

File access

Property handle functionality may also be used for accessing external files (XML, JSON, OIF, EFDS and CSV). External file structures may be defined in the dictionary or in the external file (e.g. as header in a CSV file or as separate file description).

ODABA also provides a **`Directory`** class that allows accessing directories and files via property handles.

Object-relation mapper (ORM)

An ORM allows running ODABA applications based on relational databases. An entity-relationship model is created from the database schema, which may be loaded into an ORACLE, MS SQL Server or MySQL database. Data is stored in the relational database, but an OR mapping layer provides additional features, which are nor available in relational databases.

Running an ODABA application based on relational databases does not support extension features as versioning or extension properties.

Data exchange

ODABA provides import/export functions for storing/loading data from external files. File formats supported are CSV (no names, no hierarchies), ESDF (no names), XML, OIF and JSON.

ODABA also supports defining an exchange mapping schema in order to import/export data partially and with different names and hierarchies in external files.

1.10 Database utilities

In order to support maintenance functions and other services, ODABA provides a number of utilities, for copying, checking and repairing databases, but also for loading schema definitions (ODL), running OSI scripts and investigating the database via a command line shell.

Maintenance tools

Maintenance tools are several tools for database backup, restore, copy, setup and check, but also for obtaining database statistics and database state information. Also maintenance tools include database repair utilities.

Language tools

Language tools provide ODABA specific language tools:

- ODL - Schema loader allows loading a database schema from an external ODL file
- OSI - OSI interpreter allows calling OSI functions from a command.
- OShell - Command line tool that allows browsing through a database and creating, updating or deleting data.
- OSI debugger - OSI provides a debug feature that allows debugging OSI scripts for command line and GUI tools

Database servers

Several servers may be called from command line or started as daemon/service.

2 ODABA GUI framework

The ODABA GUI framework is a technology based on Active Data Link (ADL), which passed database events to GUI controls in order to automatically fill data in GUI forms and lists. The GUI framework supports implementing GUI applications based on database model.

The ObjectCommander (GUI data browser) and is the GUI equivalent of the OS-hell command line tool. Moreover, several browser tools and components are pro-vided.

2.1 Application development tools

The GUI framework has been used providing a number of development tools supporting development phases from analysis (Terminus) via defining database model and implementing methods (ClassEditor), GUI designer for building GUI applications (Designer) up to documentation (Terminus) and test (TestBrowser).

Development resources are linked with documentation topics, that support documentation during application development. Later, document generation features may be used for creating documents from document topics. Also content of document topics may be displayed as detailed online help in GUI applications.

- Terminus - provides problem analysis functionality, but also document generation features for producing LibreOffice and HTML documents
- ClassEditor - provides database model builder and database model check functions. Also, it contains class implementation features inclusive check functionality for OSI functions, documentation features and code generator for C++ and C#, interface generator for generating OSI interface functions that allow calling C++ functions from OSI functions and others.
- Designer - supports object-oriented GUI design. Controls and forms are considered as methods of design classes. Data binding to class properties automatically controls filling and refreshing controls and forms.
- TestBrowser - ODABA provides a test framework that allows running automated tests in a command line environment. The TestBrowser provides a GUI application for defining and running tests.

The ODABA GUI framework also supports debugging GUI sessions after being executed. In order to record and replay GUI sessions, the Activity Log may be enabled. The activity log records all activities of one or more GU applications. For each action in a GUI application, a time stamp, user name, process ID, action type and name and action parameters are recorded.

2.2 Object commander

The ObjectCommander (GUI data browser) and is the GUI equivalent of the OS-hell command line tool. The Option browser supports defining any number of data sources (databases) and browse data for two data sources simultaneously similar to midnight or total commander.

Besides browsing data, instances or collections may be copied within or between databases and data may be created, updated or deleted. Also, one may run OS-hell commands or osi actions.

2.3 Browser applications

ODABA provides a number of Browser applications, which may be used for getting specific applications, but also as examples for ODABA GUI framework applications. Many of these applications ar scripted simply using OSI, so one may adapt applications to special needs.

- MessageBrowser -has been provided in order to provide an enhanced message view to the error log. Besides the error written to the error log the Message Browser displays explanatory text for each registered error. Depending to the configuration, the display refreshes continuously and old messages are removed from the message browser.
- OptionBrowser - has been provided for configuring application databases or examining application options. Besides using configuration or ini-files for setting options, options may also be defined in an option hierarchy in the application database.
- ActionBrowser - The action browser, which is also an integrated function in ClassEditor (menu item **Objects/Action Controls**), is used for defining and documenting actions to be executed in an application.
- DirectoryBrowser - is an example demonstrating how to access file system resources in an ODABA environment. Is shows directories and file content of a directory defined in the option `ROOT_PATH`. The file structure is imported into a (temporary) database and shows the directory trees and file content.