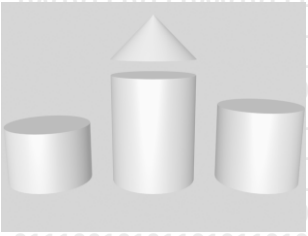


# ODABA$^{NG}$

## Technical Overview


**run Software-Werkstatt GmbH**
**Weigandufer 45**
**12059 Berlin**
Tel: +49 (30) 609 853 44
e-mail: run@run-software.com
web: www.run-software.com

Berlin, October 2012

# Content

# Introduction

**Features**

ODABA2 is an object-oriented database management system (ODBMS). It supports many of the standard features for object-oriented databases, different interfaces and provides a number of extension features.

Conceptually, ODABA is one of the most enhanced ODBMS in the world. Technically it supports several platforms and a number of special features

Models

ODABA is an enhanced $P_2$ model implementation, i.e. it supports types, relationships and collections, but also set relations and classifications, multiple inheritances and many other advanced features. ODABA supports the data model, the functional and the dynamic model.

Interfaces

You may access ODABA via C++ or COM-Interfaces. ODABA supports XML. It provides a script interface OSI with embedded query language (OSI).

Platforms

ODABA runs on all Windows platforms, LINUX and SUN Solaris. You may run ODABA in a local environment as well as in Client/Server environment. ODABA provides PIF (platform independent format) databases, which can be moved simply by file transfer between windows and SUN platforms.

Data storage

ODABA provides different types of data storage. Usually, data is stored in an ODABA database, which is the most efficient way. Since the ODABA database format is very specific and can be read by ODABA tools, only, ODABA supports XML and relational database[1] data storage. Using XML or ORACLE or MS SQL Server, you may access the ODABA database using XML tools or RDBMS tools. In this case, the performance is not as good, since relational databases or XML are not as optimized for accessing complex linked data structures as the ODABA database is.

Transactions

ODABA supports not only nested transactions, but also short, long and very long transactions, which is a persistent transaction mechanism that provides work area features.

[1] Relational database storage for ORACLE and MS SQL Server will be available in July 2007

**Conditions**

ODABA is an “open source” product. You might download one of the world’s best database systems free of charge at any time from the www.run-software.com web site as long as you are using it for your own purpose. When you are going to develop software, which you are selling to other companies, you need to buy a developer license for an appropriate number of users. Source code is available on demand.

WEB Forum

You can register in the WEB forum, which provides you with extensive documentation, news groups, discussion groups and more.

Support

You may get on-line support as well as extended support, courses or in-house support according to agreed conditions.

Membership

You can sign up a membership with a yearly membership fee, which is used for further development. Depending on the fee you may influence the direction and priorities for further developments

**Documentation**

All documentation is available in the WEB forum. This includes thousands of pages for concepts, access functions, tools and other areas, on-line browsers and research papers.

**Tools**

ODABA provides a number of command-line tools as well as GUI tools for modeling and documentation.

**ODABA2-philosophy**

The ODABA-philosophy is based on the assumption that human language is a well-proved model, which delivers the most important concepts for developing a database system. The second base for ODABA is the Unified Database Theory, a mathematical theory that allows verifying the different concepts and constructs.

Because ODABA2 is based on human language and takes its ideas from this source, you will find several useful concepts and facilities not available within other ODBMS. Even though ODABA becomes a very complex system it is still an intuitive system and easy to handle because of its analogies to human language.

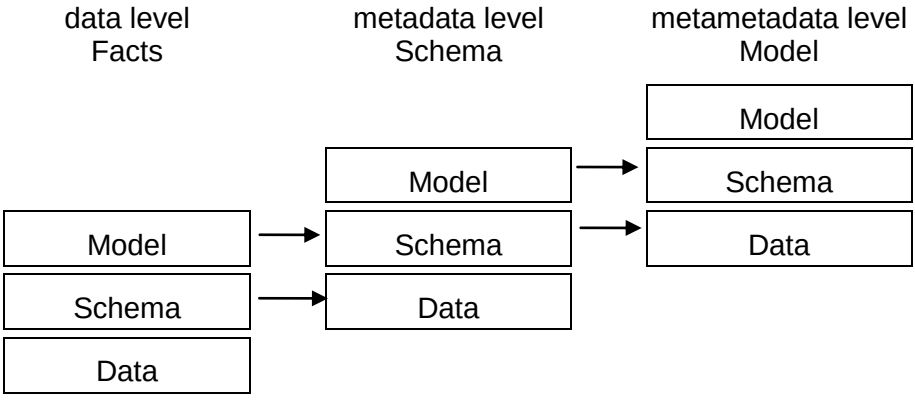
## ODABA2 - Recursive data model

*Facts are reflected on different levels that are often called data and metadata level. In general however, these two levels are not sufficient. ODABA2 is based on a recursive three level hierarchy.*

**Data level** On this level facts of the real world are reflected as states of an observed objects, i.e. an image of the object is created. The way the image looks like is described on the schema level.

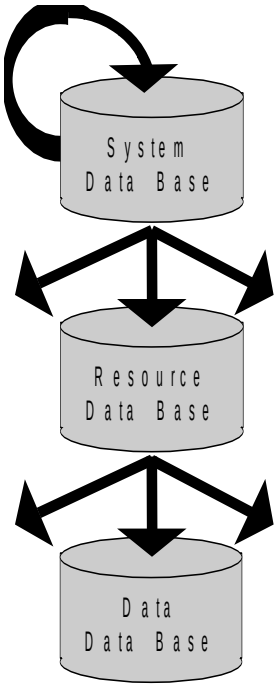
**Metadata level** On the metadata level the schemes for presenting facts and data are based on are defined (e.g. as a set of attributes or rules). For these definitions special objects on the schema level are created. Data stored at this level are called metadata

**Model level** The way reflecting schema objects is defined on the model level, i.e. the model level is the schema level for the schema level and the data at this level could be called metametadata.



## ODABA2 – Model levels and database

*Data is stored on all levels in the same way in one ore more databases, however, representing data on different levels has some effect on the relationships between these databases.*




The System Database reflects the model level, i.e. it contains the schema definition of an ODABA2 schema. This way it defines the definition of structure definitions and other objects (rules, descriptions etc.) on the schema level.

The resource Database contains data at the schema level (definitions of application structures, methods, windows, documents etc.). This way it acts as a dictionary as well as a repository.

The Data Database contains the application data, i.e. the facts observed in the real world.

Each of these data bases is considered (in terms of an OODBS) to be an independent object-oriented database which can be processed with the same means and tools, i.e. accessing data on the data level requires the same methods as accessing data on the schema or model level. However accessing an ODABA2 database always requires a dictionary.

**Dictionary**

Each database in the hierarchy shown above is the dictionary for its direct subordinated database(s). Even for the system database on the top of the hierarchy there is a dictionary. However the dictionary for the system database is completely identical with the system database, i.e. the system database is its own dictionary. In this sense the ODABA2 model is a recursive model.

# Database Concepts

*The concepts of the ODABA database are based on a detailed analysis of the human language. The human language has been analyzed mainly according to its main principles of reflecting real world facts, i.e. how human language is ordering facts, describing behavior and causalities.*

**Basic principles**

ODABA2 is based on basic principles of the human language model, i.e. it tries to reflect the way, knowledge and data is expressed in human language.

Types

Human language reflects facts by categorizing them to types (such as person, employee...). The type principle is the base for representing facts and their behavior within the $P_1$ database. Moreover, ODABA supports conceptual collections and extended set relationships typical for $P_2$ presentations

Operations

ODABA supports expressing behavior as operations. Besides standard operations provided by the system developers can add operations on different levels.

Causalities

The causal principle describes the dependency between causes and consequences, e.g. as reactions on events. This allows defining processes as well as event-triggered actions.

Time

The temporal principle reflects the facts and their relationships including their histories to present the changes of facts.

These principles can be found within the different sub models of the object-oriented model. The type principle you can find within the **object model** and the **functional model**. Causal relationships are reflected within the causal or **dynamic model**. The temporal principle, however, has not got a separate model. In ODABA it is part of the object model.

**Recursive models**

The ODABA-Model is a recursive model, i.e. the representation of data is the same on data and schema level. Thus, it is possible to use the same methods for accessing data on each level, i.e. the data for a person can be accessed the same way as a structure definition for e.g. Person. This makes it very simple to add model extensions to the system, e.g. for supporting the implementation of methods in Visual Basic classes.

# Object Model – Types and Properties

**Data *Types***

Types describe the way facts are represented. Types can be elementary or complex. Complex types contain properties that are associated with a type again.

- *Scalar types for elementary types (system types)*
  **CHAR**, **INT**, **UINT**, **REAL**, **LOGICAL**, **DATE**, **TIME**, **DATETIME** and **MEMO** are supported.
- *Enumerations for simple and hierarchical classifications*
  Enumerations can be defined for describing the categories of a classification.
- *Structures for complex types*
  Structures can be defined for complex types. Structures can be specialized to classes to reflect the common behavior of objects of a type.

**Keys**

Keys are used for identifying instances in a certain context. Any number of keys can be defined for a structure. A key may consist of several key components, which refer to attributes (transient or not) or collections. Key components can be marked as case sensitive.

**Property types**

- *Attributes (transient or persistent)*
  Attributes may have literal or complex types. The lifetime of an attribute depends on the instance the attribute belongs to.
- *References (transient or persistent)*
  References are addressing dependent instances. A reference refers to an instance or to a collection.
- *Relationships*
  Relationships reflect links between independent instances. Relationships are bi-directional.
- *Base Structures*
  Base Structures define generalizations for a structure. ODABA2 supports multiple inheritance as well as multiple derivation.

**Instance Identity**

Instance identities are created for structure instances as well as for collection instances. The instance identity in ODABA2 is a 64-bit number. Hence, the number of instances in an ODABA2 database is practically unlimited.

# Object Model – Collections

*Sets of instances are represented as collections in ODABA. Collections in ODABA are always reference collections, except array attributes. Collections may refer to instances with the same type (typed), to instances having the same base type (weak typed) or to instances of any type (untyped).*

**Extents**

Extents are global controlled collections, which provide an entry point to the data in a universe.

Hierarchies

Extent hierarchies can describe persistent subset relations as well as unions and intersects.

**Collections**

Collections may appear as independent global collections within the scope of an active object or as local collections of a structure instance.

Owning

An owning collection owns its instances, i.e. removing an instance from such a collection means to delete the instance. ODABA ensures that each instance belongs to exactly one owning collection (or reference).

Controlled

For controlled collections, ODABA guarantees the consistency of related instances and indexes as well as the consistence of defined subset relations.

Indexes

Ore more persistent or transient indexes can be defined for each collection. An index is based on a key defined for the type of the collection. Indexes support ascending and descending orders.

- *Unique indexes*
  An index can be defined as unique index, i.e. the key values in the index must be unique.
- *Identity index*
  The identity index is a special index that is using the instance identity as Key.
- *Temporary indexes*
  Indexes can be stored as persistent indexes or created as transient indexes on demand.
- *Consistency verification*
  ODABA2 ensures the consistence of the indexes for controlled collections (extents, reference collections, relationships with inverse reference), whenever key components are modified.

# Object Model – Enhanced Concepts

*Besides the standard concepts for ODBMS the ODABA data model supports some enhanced concepts. These concepts are concerning with the representation of rather complex contexts as functional dependencies and complex objects.*

| | |
|---|---|
| **Views** | A view structure defines the intentional (structure) and the extensional aspect of a certain user view, i.e. the properties and the amount of instances to be shown in the view. |
| Structure | A view structure defines the intentional (structure) aspect of a certain user view. It is based on one or more persistent structures and defines a number of derived properties (attributes, references and relationships). |
| Extent | A view can apply on a set of view paths that refer to collections according to the type defined in the view. |
| Operation | A view is based on a set operation as UNION, PRODUCT, MINUS, INTERSECT, which defines the way of combining the passed collections. |
| **Project** | A project is the technical reflection of the view of a subject, which refers to a number of extents as basic concepts of the project. A database may support several projects that support cross-links. Projects may form hierarchies. |
| Module | A project may consist of a set of modules, where usually each module represents a executable unit. Modules may form hierarchies. |
| Namespace | A module consists of one or more namespaces, which may again form hierarchies. |
| Extents | Each namespace defines an own domain for storing instances and collections, i.e. has its own set of extents. |
| **Set relations** | ODABA2 supports consistent set relations as an integral part of the object model. |
| Extents | Set relations can be defined for extents as logical consistency rules. This includes subset/superset relations, union and intersect relations. |
| Relationships | Subset relations can be defined between relationships, which are linked through property paths. |

**Versions**

Time plays an important rule in many applications. ODABA supports an independent (orthogonal) time dimension in the database in different ways.

Instance versions

Instance Versions can be created to keep the previous instance state. However, when creating instance versions no versions for indexes are created. Thus the history of an instance can be seen but the index search for older versions is not supported.

Database versions

Different versions can be created for a database. When creating a new database version, modifications on instances will create new instance versions automatically. Moreover, indexes and relationships of the older versions are duplicated before being modified in the newer version. A roll back to an earlier version can recreate an earlier state of the database.

Schema Versions

When considering the complex relationships in an ODABA repository it becomes obvious that it does not make sense to create versions of a single structure. By creating project versions, which are universe versions for the dictionary, the complete state of a repository including the structure definitions, actions, method definitions, forms and many other repository resources, are preserved.

Thus, ODABA is not only providing on-line schema evolution but supporting version control as well.

**Generic Attributes**

Generic attributes are attributes that might be defined with different conceptual versions for an instance (e.g. languages). This allows presenting an instance according to a selected type for the generic attribute

Indexes

Indexes for keys referring to a generic attribute are generic indexes. A generic index generates a set of indexes for each type of the generic attribute (e.g. language).

# Functional Model – Overview

*ODABA supports the function model (representation of behavior) as methods, which can be defined in different method classes. From each structure, several classes can be derived to represent the methods in different ways (e.g. as OSI-Expression or window template).*

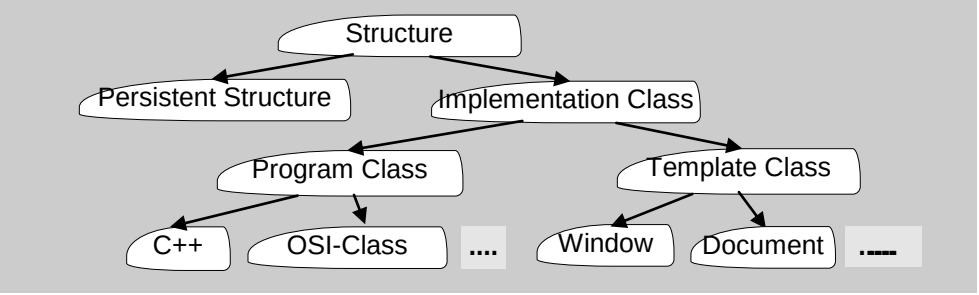
**Conceptual levels** Classes can be provided on different conceptual levels and on different levels of specialization.

System Classes: About 10 System classes provide the basic (generic) access functionality to the database. System classes provide generic access on the property level but also typed access by means of generated C++ classes.

Problem Classes: Type specific methods within problem classes, which might be implemented in different ways.

Context Classes: Context classes support implementing specific behavior in a certain context (e.g. in a collection. For a specific relationship). Different types of context classes can be implemented for a type.

Because there are different Class types for implementing methods, a user defined Structure may be derived to several Classes:




**Class Types** Classes can be implemented as program or implementation class.

Program Classes: ODABA supports a proprietary scripting language (ODABA OSI), for implementing methods as well as different programming languages. Besides C++ any language supporting COM (Visual Basic, Delphi etc) can implement user classes.

Template classes: Template classes are used for data presentation in windows forms, documents or HTML pages.

# Functional Model – System Classes

System Classes are provided as C++ classes or via COM. Thus, they can be referenced in any program environment that supports C++ or COM.

**Client/Server** — Client/Server classes are used to establish a connection between server and client. ODABA applications are scalable and may run locally, as well as in a client/server environment.

- Client — The client class provides functionality for running an ODABA application (as Connect).
- Server — The server class provides functionality for running a server and allows implementing an ODABA server in a specific environment.

**Database access** — Database access classes provide functions for accessing databases and repositories or dictionaries.

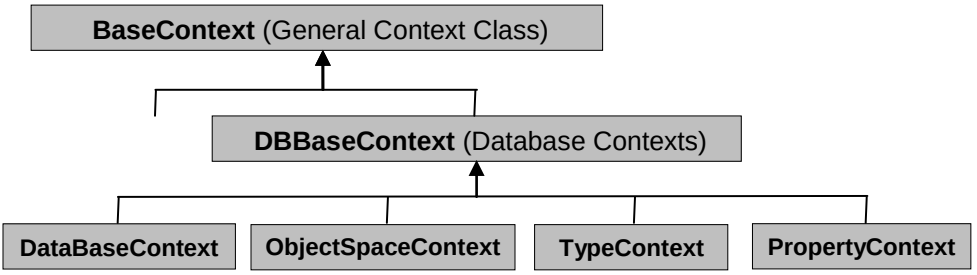
- DataSource — The data source handle allows opening a database as being parameterized in an ini-file or database catalog..
- Dictionary — This class provides the functionality for accessing structure and extent definitions. Because a Dictionary handle is a special database, it supports also all Database handle class functions, which allows accessing any resource defined in the repository.
- Database — The class provides functionality for accessing object spaces defined in the database. Since each ODABA database is a root object space, ObjectSpace handle functionality is inherited from Database..
- ObjectSpace — This class contains the functions for accessing (create, delete, get) subordinated object spaces in a database as well access functions to the extents of the object space.

**Property/Value** — The classes provide functions and operations for accessing and manipulating property instances and instance values. The property handle supports cursor functions while value handles support operations on value properties. The property and value handle classes supports more than 250 functions for

- Select, create, delete, copying instances
- Set operations (union, intersect, minus select)
- Creating and accessing views
- Value operations and type conversions
- Locking and transaction features

# Functional Model – Context Classes

*In many cases, the behavior of an instance depends on the environment or the context in which the instance is referenced. Within the scope of a database, the context for a property is defined by the structure containing the property, while the context for a structured instance is usually a collection (property) that refers to the instance. Context classes allow modifying standard behavior for a property or instance implemented in a problem class. ODABA supports different types of context classes.*




**Database Context** Database context classes allow defining special behavior of the database as user's login or authorization checks. Database context functions are called when opening or closing a database.

**Object Space Context** The object space context allows similar to the database context defining special access rules for a universe. Database context functions are called when opening or closing a universe.

**Structure Context** The structure context allows specializing the behavior of in instance of a given type. Structure context classes are used to fill transient attributes or references defined in the structure or providing access control on instance level. Structure context functions are called on specific system events as reading, storing or updating an instance.

**Property Context** The property context supports specializing the behavior of in instance of a given type. Property context classes are used to fill transient attributes or control insert and remove actions in collections. Property context functions are called on specific system events as inserting, or removing instances or updating a property value.

# Dynamical Model – Concepts

*The dynamical model of ODABA allows reflecting causalities in the database model, i.e. defining when and why a behavior is activated. This includes process definitions as well as very detailed reactions on events. The dynamical model allows defining events as potential state transitions and reactions, which refer to the method executed in case of an event. Since each time point can be defined as event, the dynamic model implies process control features as well.*

**Conditions**

Conditions are describing the requirements to be met for behaving in a certain way, i.e. for executing a certain method. Conditions may be described as logical expressions. Conditions are typically used for defining validation rules.

**Events**

Events indicate the requirement of executing a certain action. Thus, events usually control processes and actions to be taken.

Application events

Application events are defined as relevant potential state transitions (e.g. an employee now gets more than his boss and did not before). Pre- and a post-conditions provide a simple method of defining an event. When raising an event the associated action (reaction) is executed.

System events

System events are defining specific events raised by the database access system (e.g. read/write or delete/create events). The typical way of handling system events are event handler functions in corresponding context classes.

**Actions**

Actions provide the frame for executing a specific method implemented as program function or template. Besides referring to the method, the action may pass parameters and other information required for executing the method.

**Reactions**

A reaction defines the connection between an event and an action. Thus, the defined action is executed whenever the event is raised. Because the execution of the action may cause other events associated with an action, a chain of reactions is possible just as the consequence of a little event.

**Processes**

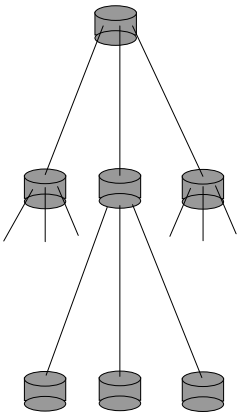
Processes are a special case for executing actions. Processes can be organized in sequences or networks controlled by events or time schedules.

# Technical Concepts – Enhanced features

*ODABA2 databases can be used within a single user environment as well as within a multi-user environment. Standard concepts as client/server support or distributed databases are supported by ODABA as well as enhanced technical features like "very long transactions".*

**Distributed databases**

ODABA2 databases can be defined as consisting of different files located on several servers. An ODABA2 database may be distributed on more than 1 000 000 000 files of the operating system. The maximum size for each database file is about 140 000 GB bytes or 2 GB bytes when running in 32 bit file server mode.



The **Root-Base** contains the basic database information and may consist of up to 4 000 Main-Bases. The allocation of instances to main bases is defined on the extent level, i.e. instances are stored within the Main-Base defined for its Extent. The file for the Main-Base 0 is identical with the Root-Base file.

Each **Main-Base** consists of up to 32 000 data areas. The allocation of an instance to an data area is done by the system according to the data area specifications. The data area 0 file is identical with the Main-Base file.

**Data Areas** can be defined as dynamical data areas or with a given size. As far as there is no space left within a Data-Area instances are stored into the next data area.

Standard database

The default case is a database consisting of **one file**, only, which is Root-Base as well as Main-Base 0 and Data-Area 0. Such a default database does not require any administration on the physical level.

Standard databases are initialized automatically, when being opened the first time.

**Client/Server**

The ODABA Client/Server concept is based on specific ODABA-CS protocols, which is based on TCP/IP. Thus you may start your ODABA-Server somewhere in the WEB and you can access it from any place in the world.

In a client server environment databases can be located on different servers or on the local machine.

File server

The file server version works on Windows platforms, only and is a simple way for running several (updating) applications simultaneously without installing a server. The database size for file server applications is limited to 2 GB. When the database exceeds 2 GB, a multiple dataset database must be defined (more than one data area).

Object server

The object server provides support for all ODABA system classes. This allows developing applications independent on local or client server environments.

**Multi user environment**

ODABA provides several features for supporting parallel processing, i.e. allowing several applications accessing database resources simultaneously.

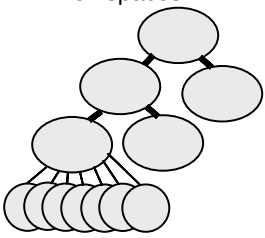
Locking

Locking features are provided as on the instance or collection level as implicit locking or explicit locking within the application. Pessimistic locking strategies are supported as well as optimistic ones.

Transactions

Transactions can be started for a universe (database object). On the application level, short (internal) transactions are supported as well as long (external) transactions. Instances participating in a long transaction are stored temporarily in an external transaction database. Nested transactions are supported.

Workspaces



Workspaces are a special support for very long transactions. Working in a workspace, each modification will be stored automatically in the workspace opened by the user. Changed can be held in a workspace for weeks or month until they are consolidated or discarded.

Workspaces can be arranged in hierarchies, e.g. for work groups and users in work groups. Workspaces can be preallocated or created on demand.

**Recovery files**

By using recovery files, all modifications done on the database can be logged including user identification and time stamp. Thus, the recovery file is a sort of action log on the one hand and can be used for recover the database in case of damage on the other hand.

# Technical Concepts – Performance issues

*Two level access via instance descriptors allows variable length instances but also reduces the efficiency of the data base. To increase the performance ODABA2 uses different techniques combined with additional features.*

**Server cache** — In a client/server environment a server cache can be used to increase the performance. The server cache is optimizing the cache size according to the access frequency.

Descriptor cache — Instance descriptors for frequently read instances are cached on the server side. Thus it is not necessary to read the instance descriptor again and instances can be accessed directly.

Instance cache — Instances can be cached on the server side avoiding rereading instances. The size of the instance cache can be configured as well as the flush time for writing modifications to the database.

**Block access** — Block access is reading and transferring a block of instances, which extremely reduces the net traffic in a client/server environment.

**Views** — Views are processed on the server side and may increase performance extremely as well.

**Transaction buffer** — When processing (adding or modifying) large numbers of instances, performance can be improved extremely by using dynamic transaction buffers.

**Clusters** — Instances accessed together in many cases can be stored in a cluster. When reading a clustered instance not only the single instance, but the complete cluster is read in. Thus, no more file access is required when reading other instances stored in the same cluster.

**Clustered tables** — Instances can be stored in clustered tables for providing faster access. Instances in clustered tables are stored in one block an do not need an instance descriptor.

## Query or script language

*Why not providing query language as a programming script language? With ODABA Script Interface (OSI) ODABA provides a script interface, which allows comfortable queries, but also any type of database programming*

**Language**

Syntactically, OSI is similar to C++/C# or JAVA. It supports multiple inheritance, function overload and other typical program language features.

In contrast to a pure programming language, it provides, however, direct access to the database, since all database types. members or OSI functions are considered as known symbols.

```
collection bool Person::PrintChildren {
VARIABLES
  string     names;
PROCESS
  while ( next )
    if ( children.count ) {
      while ( children.next ) {
        if ( names != '' ) name += '; ';
        names += children.first_name;
      }
      Message(pid + ' has children: ' + names);
      names = '';    }
}
```

**OSI functions**

OSI supports function calls to built-in functions, but also to interface extensions or user-defined functions..

API functions

OSI supports all ODABA application interface class functions including helper classes, i.e. from within an OSI script, any Property or Value function might be called. Moreover, additional support is provided for File access and specific system functions.

Extensions

In order to increase the power of OSI, you may provide additional services by implementing your OSI interface classes.

Context functions

One may also invoke context functions (business rules) from within an OSI function.

**Query language**

As query language, OSI supports most of the requirements of the ODMG. This includes also support for traditional query statements as SELECT or JOIN. Those are considered as OSI built-in functions and might be mixed with other OSI functions.

# Tools

*ODABA2 provides three categories of tools as server commands, command line tools and GUI tool. All tools are considered as developer or administrator tools.*

**Command line tools**

Command line tools provide administrative and service functions for developers and database administrators. Command line tools are available on all supported platforms.

**Server commands**

A collection of server commands allows controlling the server and performing specific server commands. Server commands may run on the server or on any client, which provides remote control.

**ODE**

The ODABA Development Environment provides a number of GUI applications for defining and documenting the object model, the functional model and the dynamic model (Class Editor, Designer, Document Composer).

The present version is available on Windows and LINUX platforms. All tools have been developed using QT (Nokia), which provides multi-platform support.

**Object Commander**

The object commander is a data browsing and edit tool, which provides a tree view that allows browsing the database following the links defined by references and relationships. It allows also editing, copying, creating and deleting instances or collections of instances.

The object commander is available on Windows and LINUX platforms.

**OSI**

The ODABA Script Interface provides a Java-like programming language, which can be used for browsing or updating the database, but also for schema definitions (ODL). OSI is OQL compliant but contains a number of ODABA specific extensions. OSI includes query language elements as well as program language elements (Java). Thus, OSI can be used at the same time for running queries but also for executing programs.

OSI is available on all supported platforms.

## Multiple storage support

*There is, according o "Unified Database Theory", no difference between information that can be stored in relational and object-oriented databases. Hence, the content of an terminology-oriented data model can be stored completely in relational databases as well.*

**Relational storage support**

ODABA supports generating an entity-relationship model from an ODABA database schema for ORACLE, MySQL or MS SQL-Server. The generated model supports data types defined in the terminology model, but also m:n relationships and shared inheritance.

Accessing data in RDBMS

ODABA applications may store data directly in a relational database. This is not as fast as accessing data in an ODABA database, but allows running SQL queries against an ODABA database. Data stored in the RDBMS can be accessed via ODABA functions. ODABA applications are not affected by the type of data storage chosen for the application.

When running ODABA on a RDBMS, data might be read directly in the relational database, but has to be written via ODABA access functions.

OR Mapper

Since ODABA stored additional system information about instances and collections, an OR Mapper is used for storing instance and collection information. Thus, many extended features (e.g. event management) are available also when storing data in relational databases.

**XML storage support**

ODABA data and dictionary data might be stored in an XML database. ODABA provides specific XML schema extensions in order to provide an ODABA database schema in XML. .

Using XML

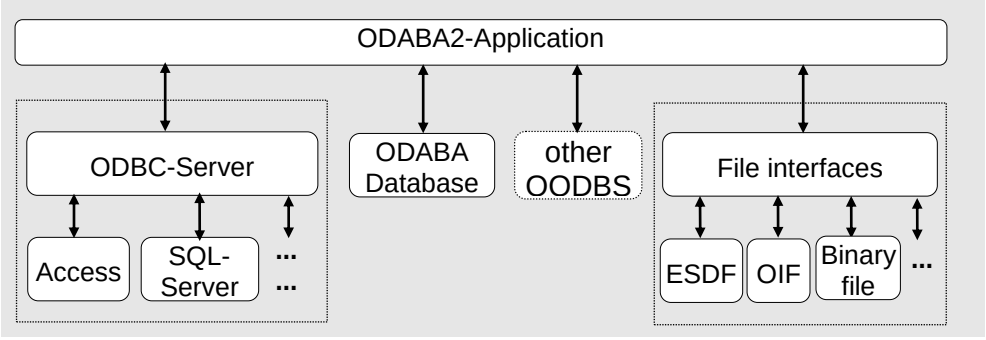
XML data can be accessed easily using ODABA property handle functionality. Mainly, XML is used for data exchange between different systems.

Theoretically, one could run ODABA based on an XML database, but this becomes very slowly in is not the intension of supporting XML.

ODABA2-Application
ODABA2-Application

# Data Exchange and Communication

*There is no homogeneous environment for database applications. Usually a database application has to communicate with several other applications. ODABA2 provides a rich number of data exchange facilities from within an ODABA2 application as well as from other applications.*




**Data Exchange**

ODABA supports explicit data exchange by means of exchange definition and implicit data exchange by property handles referring to external data sources.

Implicit exchange

Within an ODABA2 application external relations like Access or SQL-Server relations can be accessed the same way as internal ODABA2 Extents. Since all data sources are accesses by means of property handles, the application will not notice different types of the data source.

Explicit exchange

ODABA2 allows defining semantic mappings between an object-oriented and an entity relationship model. This includes attributes as well as relationships. It allows transforming databases based on an ER model to an ODABA2-database and reverse as well as transforming databases between different ER models.

**Supported formats**

Relational databases can be accessed via the ODABA ODBC interface. Other data formats as XML, extended self-delimiter files (ESDF) or flat files can be accessed by means of special access extensions provided by ODABA.

**Accessing ODABA2**

Data within ODABA2 databases can be accessed from within other applications via the C++ or COM API that provides a large scale of access functionality.

# Command line Tools

*Command line tools provide administrative and service functions for developers and database administrators. Command line tools are available on all supported platforms. Most of the tools are available as ODABA Client functions and can be called from a user-defined application as well.*

| | |
|---|---|
| **Server** | The ODABA server provides faster and optimized access to the database. You may run an ODABA server in a local network area but also in Internet. |
| **Server Commands** | A number of server commands can be called from the console on the server or on a remote computer. Thus, the server can be administrated locally but also remote. . |
| **SetupDB** | The SetupDB command allows upgrading a database to a higher schema version. This might increase the performance when upgrading the database in advance. |
| **CopyDB** | The CopyDB command allows copying a complete database or parts of it. |
| **CopyResDB** | CopyResDB allows copying resource objects as structure definitions, implementation classes, forms and others. |
| **PackDB** | PackDB allows compressing a database to remove unused space resulting from deleted instances. |
| **DBSystemInfo** | DBSystemInfo displays system information for the database as database version, schema version, main-bases etc. |
| **DBStatistics** | DBStatistics provides an overview about count and space used for each instances and indexes by type. |
| **Workspace** | The Workspace command allows enabling or disabling workspaces as well as consolidating or discarding workspaces or listing existing workspaces in a tree. |
| **License** | License services are provided when an application is delivered as application that needs a customer license. ODABA itself does not require a license but any application might do so. |
| **TestCS** | TestCS is a stress test utility for client/server environments. |
| **OShell** | The ODABA command utility supports many of the program functions that are available for the system classes. Thus, it provides a command language that you can run immediately on your computer in an ODABA console. |

# ODE - Overview

*The ODABA development environment **ODE** is based on harmonized development phases, starting with defining the problem within a natural language up to reaching the implementation and maintenance. The ODE is supporting phases as analysis of requirements, object modeling, GUI design and method implementation and maintenance.*

**Object model**

The object model can be defined in two phases. The conceptual phase provides a terminology model, which can be transformed into technical model.

Terminus

With Terminus, ODABA provides a tool that allows defining concepts as terminology model in advance. From the documentation you can generate an ODABA object model, UML or documentation.

Class Editor

The object model can be generated from the terminology model provided with Terminus. Definitions can be completed in the Class Editor, which allows defining structures, enumerations and extents.

**Functional model**

The functional model allows implementing methods for structures in different classes.

Class Editor

The Class Editor supports the implementation of methods for defined structures. Classes and methods are generated from documentation and can be implemented as C++-functions or OSI-expressions.

Designer

The GUI design includes the design of several project resources as application, menus, windows and controls. It provides also specific GUI context classes, which allow reacting on GUI events.

**Dynamic model**

The dynamical model allows defining user events and reactions.

Causality Design

The causality designer allows defining the dynamic model. Reactions and action sequences (scenarios) can be defined for handling different events. The Causality Design is supporting causal relationships on the database level.