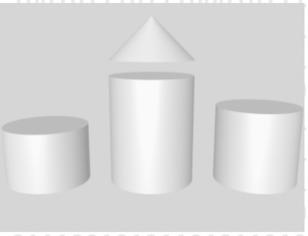


ODABA^{NG}

ODABA$^{\text{Next Generation}}$

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

1 Introduction

With ODABA^{NG} the first terminology-oriented database management system has been provided, which supports a smooth transition from the conceptual design (list of requirements) via technical specifications (process design) up to application and documentation. To meet the concept and terms as being used in the everyday work in the database and in the final product – that is something customers really liked to have, and ODABA is a technology supporting this goal. It is not surprising: Using proper terminology avoids mistakes, increases development efficiency and product quality is growing.

Terminology-oriented database systems are a conceptual extension of object-oriented database systems, i.e. one may refer to most of the standards and technologies known from UML and object-oriented technologies.

2 Suit the application to the words

Basically, ODABA^{NG} is terminology-oriented not only from a philosophical point of view. Meat and potatoes of all ODABA^{NG} applications are terminology models[1], which can be considered as extended Concept Systems [9][10].

Another basic idea of ODABA^{NG} consists in using development resource objects as atomic units in the application development. Development resource objects are documented units that can be combined in many different ways in a project. Documentation topics in the terminology model are development resource objects as well as data types, functions, parameters, forms, controls and many others. Development resource objects can be combined in different ways e.g. within an data model, application, user or technical documentation, WEB sites, UML diagrams and others.

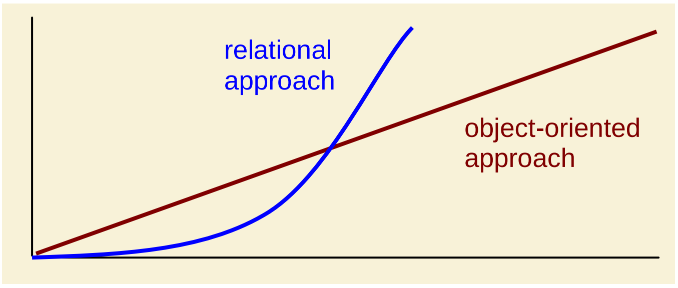
Each development resource object is a document topic, i.e. documentation is the backbone of the technology provided with ODABA^{NG}. Exploring the information from the terminology model, an object model can be created automatically, which covers up to 80% ob the model requirements. From the object model the application design will be derived and finally, you may get documentation and online help with one button click.

With ODABA^{NG} we did not want to provide something new and strange, but we tried to stay with known technologies as far as possible. The Object Data Standard ODMG 3.0 [2] seemed to be an appropriate base and ODABA^{NG} supports most of the features suggested in this standard.

But this was not enough. Several extensions had to be made in order to fulfil the requirements of a terminology-oriented database, but also in order to fulfil extended technical requirements of ODABA^{NG},

Object-oriented or relational

Providing an object-oriented application, usually, implies an increased initial effort. In the long term, however, an object-oriented application is easier to maintain and easier to manage. Typically, a project started ones will grow over time, in which case the effort in a relational application grows exponentially, while object-oriented applications grow line-



ar [11]. Moreover, using object-oriented database technologies releases the developer from implementing a data layer, since this is a built-in function of the object-oriented database management system. Practically, this means: no Select, no Join statements – data can be accessed directly and immediately.

In contrast to relational database management systems (RDBMS), object-oriented database management systems (OODBMS) support the functional model in addition to the data model. Theoretically, the dynamic model is supported by OODBMS as well, but practically, most OODBMS do not provide much support for this model. ODABA^{NG} supports the dynamical model by means of system and application events on different levels. This allows developing event-controlled applications using Active Data Link technologies [6].

The object-oriented data model is more expressive than the relational one. Many things that have to be programmed in a relational environment (joins, selects) simply become part of the schema definition in the object model.

Many OODBMS combine data model and functional model definition within a programming environment (e.g. within a C++ class definition). This sometimes causes the impression, that object-oriented databases are an integrated part of a programming language or a specific program language extension. Thus, known OODBMS as FastObjects or ObjectStore do support only one class type per object type. Since there is no theoretical or practical reason for such a restriction, ODABA^{NG} supports different implementation classes as C++, .NET, OSI and Design classes.

ODABA^{NG} does more

ODABA^{NG} shows, that there is also a classical approach for object-oriented database systems, which clearly separates the definition of the object model and the functional model. This becomes possible, because ODABA^{NG} distinguishes between object types and implementation classes. Since implementation classes are specializations of object types, developing the object model becomes independent from the functional model. When necessary, an object type may be specialised to different implementation class. Thus, ODABA^{NG} applications become more flexible concerning different programming languages.

ODABA^{NG} allows one object type to be he base for different implementation classes. Thus, the „Person“ object type might be implemented at the same time e.g. in a C++ class, in an ODABA Script Interface (OSI) class and in a design (GUI) class. Strictly separating object model and functional model increases flexibility of object types, since those are not restricted to one class type, anymore.

The object-oriented kernel of ODABA^{NG} is based on the Object Data Standard ODMG 3.0 [2] and supports its requirements for the object model concerning:

- Inheritance
- Relationships and inverse relationships
- Enumerations
- Module/Namespace-Hierarchy

This, however, is not sufficient for ODABA^{NG}. Many significant extensions have been provided, which mainly result from the requirements of the terminology model. Most important extensions are features as:

- **Shared inheritance** allows inheriting more than one object instance from another object instance (e.g. creating several employee instances for one person instance, or different implementation classes for an object type).
- **Set relations**, includes definitions of superset and subset relations for relationships and extents, but also defining intersections or union sets in the object model. Set relations are maintained by ODABA^{NG}, which considerably reduces the burden of work in application development.
- **Hierarchical enumerations**, which are also called classifications, provide a way of defining object classes on different levels.
- The module/namespace hierarchy has been expanded by a **project** level above.
- **Multi-lingual text attributes** allow providing text in several languages.

This is just a subset of useful extensions provided with the ODABA^{NG} object model, which are described in detail in the ODABA^{NG} online documentation [3].

The functional model of ODABA^{NG} supports C++, OSI (ODABA Script Interface) and GUI classes (presentation methods). A .NET interface allows accessing the database and implementing methods in C#, MS Visual Basic or Java.

In large projects, which require a strict separation between database, business and application layer, it becomes obvious soon, that persistent object types do not provide sufficient information for implementing a convenient business and application logic. Both layers usually require additional information about the application state.

Therefore, ODABA^{NG} supports so-called **Context classes**, which provide required runtime information and allow defining and implementing the behaviour down to a single property in the object model. Typically, business logic is implemented in ODABA^{NG} applications vie database context classes, which are linked to object types or their properties.

Just as well, the application logic can be implemented in application context classes, which are linked with controls and other GUI elements of the graphical user interface.

What seems to be rather complicate at the first glance, in fact, makes application development easier, since separating project layers increases transparency of the application.

In relational environments, causal dependencies can be implemented via triggers. OODBMS (including ODABA^{NG}) support events and event handlers, instead. Besides the object and the functional model, ODABA^{NG} supports a **causal model** (or dynamic model), which defines events as relevant state transitions of one or more objects. Most event handling database systems send the event just to the object causing it, e.g. update events for a Person object instance are sent to the Person. This is useful in many cases, but not sufficient. A causality model, however, describes the relationship between the event generating object (sender) and reacting objects (receiver). Supporting such a causality model is, indeed, a new area covered by ODABA^{NG}.

In addition, ODABA^{NG} supports two more models, which are not sensational, but nice to have:

The **documentation model** is a consequence from the terminology-orientation of ODABA^{NG}. The documentation model is ISO704 [9] compliant, but provides significant extensions. Besides concept definitions, all development resource objects (object types, properties, functions, parameters etc.) may be documented in document topics, which provide standard documentation features. The direct link between documentation and documentation subject is an important practical advantage. In addition, the documentation model supports notices, which might be linked to any development resource object and allow storing development notes.

The **administration model** is another model extension, which is provided by ODABA^{NG} in order to support user and process administration. User and user groups are also referred to in the notification system, which allows sending notices to other developers.

The variety of ODABA^{NG} features is described in detail in „Technical Overview“ [5] and in „Database Concepts“ [8].

3 Technical specialities

More than other OODBMS, ODABA^{NG} supports interfaces to other systems, not only concerning data exchange with other databases or documents, but also for supporting event driven applications and different ways of communicating with the database.

Events en masse

ODABA^{NG} allows developing simple application driven systems. Much more efficient are, however, event driven systems, since those allow implementing consistent and robust business logic. Therefore, ODABA^{NG} generates a number of system events not only on object type level but also on property level. Besides system events, the developer may define other data events in the causality model.

Event handler can be implemented in database context classes as OSI-Expression, C++ or .NET functions. Context class event handlers react immediately, when the event is signalled.

This is, however, not sufficient for applications that require an **Active Database**, since in this case the application needs information about changes for data and in application states. In contrast to other DBMS, ODABA^{NG} supports the requirements of **Active Data Link** [6] which is a pre-condition for becoming an active database. Thus, ODABA^{NG} generates additional **handler events**, i.e. events are passed through database access handles (e.g. property handles) to the application. In contrast to business logic, which usually needs to react immediately on data events, application logic reacts only, when a database transaction has been committed. ODABA^{NG} collects data events in a so-called read transaction, which also optimises data events.

Context and handler events do not pass process borders. Cross-process events are generates when applications communicate with an ODABA^{NG} server. In this case, server events are sent asynchronously to the clients and might be processed by clients. Limited server events are also generated, when running a replication server.

Moreover, ODABA registers changes in the database, which creates a sort of lazy events, that can explored by the application.

Talk to others

Application development with ODABA^{NG} is possible on different levels. ODABA^{NG} applications might be implemented completely without using the ODABA^{NG} development environment (ODE). The object model can be defined by means of ODL-Script (Object Definition Language). Schema definitions can be loaded to the dictionary by means of a Schema Loader. After implementing the schema, the application might be implemented in C++, C# or Visual Basic (using the .NET interface), but also in OSI or mixed. Via an ICE Interface,) an ODABA^{NG} database might also be accessed from within PHP or JAVA programs.

With the ODE, however, ODABA^{NG} provides a comfortable development environment, which supports specifying the terminology model, defining the object model, implementing the functional model and designing and implementing the application model, which makes life much easier.

Everybody likes to script

Beside standard application programming interfaces (APIs) for .NET, ICE and C++, ODABA^{NG} provides a comfortable script language. The ODABA Script Interface (OSI) supports data definition language (ODL), database queries (OQL) and data manipulation (OML). Considering the syntax, OSI is much similar to JAVA or C# and, hence, easy to understand for C++, C# or JAVA developers. OSI completely supports ODABA^{NG} API functions, which allows accessing all elements defined in the object model (including metadata). In order to store results or intermediate data, OSI also supports locally defined data types, which are not part of the database schema. From within OSI, the database appears like a huge address space, since database extents and properties appear as ordinary variables in an OSI method. OSI variables represent elementary values, but also of complex data types or object collections.

Another feature provided by OSI is the document template, which is used in order to define documents accessing the database (Open Office) or HTML pages for WEB presentations.

4 Technical astonishments

ODABA^{NG} also provides a number of technical surprises. On the one hand, ODABA^{NG} provides scalability concerning different aspects. Different hardware and system platforms are supported as well as a number of client/server models or storing data in different types of databases. But also for database standard features as transaction mechanisms or locking, ODABA^{NG} always supplies some nice extensions.

Flexibility in every sense

ODABA^{NG} provides platform independency on different levels. On the one hand ODABA^{NG} applications may run in different operating systems (MS Windows, LINUX, Sun Solaris), including ODABA^{NG} GUI applications, which are based on QT (Nokia), which provides a platform independent GUI subsystem. Data might be stored in a database as PIF data (Platform Independent Format). This makes it possible, simply copying a database e.g. from an Intel machine to a SUN machine.

The current ODABA^{NG} version supports several client server models.

- File server
- Replication server
- ODABA server

File servers are simple to administrate and may be used in local networks. The file server simply trusts the file server functions of the operating system and does nothing more than delivering requested data sequences and locking data areas in the database file.

Replication servers are also „stupid“ servers, which are mainly used for accessing a server accessible via internet. The client loads a local database copy, which is updated from the server, whenever a transaction has been send to the server. This makes clients performing like local applications, even though the database is located on a server somewhere in the internet.

The ODABA server is intended to be used in local networks, too. In contrast to the file server, business logic is mainly executed on the ODABA server.

The application development is not affected by the client/server model, i.e. the client/server model might be replaced at any time by another one.

Store the data where you want

As most DBMS, ODABA^{NG} supports interfaces to external data formats. Besides ESDF (Extended Self Delimiter Files, a CSV extension), ODABA^{NG} supports Flat Files, OIF (Object Interchange Format) and XML, as well as access to several relational databases.

A completely new path has been stricken 2006, when the multiple storage interface (MSI) had been created for ODABA^{NG}, which allows storing data and metadata in different storage formats. Thus, ODABA^{NG} data, but also schema definitions might be stored e.g. in XML or XML schema, but also in OIF and ODL. Theoretically, it becomes possible to store an ODABA^{NG} database completely in XML storage format. This does not make much sense, but the feature is used mainly for importing or exporting complex data structures or for processing SOAP and WSDL requests.

MSI becomes more interesting in connection with relational databases. Since relational and object-oriented databases are identical concerning the amount of information that can be stored [7], it becomes obviously possible to store the content of an ODABA^{NG} database also in a relational database. In order to follow customer preferences concerning specific data storage formats, data can be stored in an Oracle, MS SQL Server or MySQL database without loosing the benefits of object-oriented development. An object-relation mapper transfers the data model to a relational schema. In order to support Active Data Link, additional meta-information is stored in an Object Manager database.

Concurrency – no problem

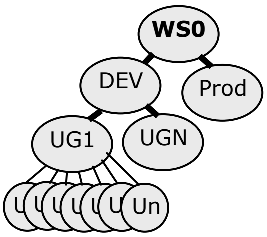
ODABA^{NG} developers need not to deal very much with concurrency control and locking. ODABA^{NG} supports explicit locking for object instances and collections, but usually, locking strategies are controlled by the system. Implicit locking strategies for pessimistic and optimistic locking help the developer and prevent the application from deadlock situations.

Transactions

Sooner or later, every application needs transaction management, since nobody can estimate the consequences of complex changes. In an object-oriented environment, a simple action as inserting an instance to a collection, implicitly may insert the instance also to supersets or inverse collections. In addition, events are fired, which may cause other reactions and data modifications etc.

Any modification in the database (as create, delete, insert, remove or update) leads to a number of determined consequences as maintaining supersets and subsets or inverse relationships and indexes. Other, undetermined, consequences are caused by reactions implemented in the business logic, which may react on update events causing further other modifications. Each step in such a chain of reactions may fail. Hence, each modification is encapsulated automatically in an internal (short) transaction, which rolls back all the changes, when a problem has been detected.

Moreover, ODABA^{NG} supports long transactions, which keep changes in memory or store those on an external device until committing the transaction. In addition, ODABA^{NG} supports save transactions, which store after images in a transaction log before writing those to the database. The transaction log also provides a log file, which allows tracking changes and restore data in case of data crash,



But also concerning transactions, ODABA^{NG} offers something special. Persistent transactions called **work-space** enable users storing modifications in its own work-space or in a group workspace within a workspace hierarchy. Each workspace defines a very long transaction, which may take days or even weeks. Changes within a workspace become visible only for users working in the same workspace or below. In contrast to normal transactions, workspaces are accessible from several processes and users. When the changes in a workspace are completed, the workspace can be discarded or consolidated in order to store changes in the upper workspace or in the database.

Data versions

Data versioning is not a typical database feature. Data versioning means, that changes of an instance or index are stored in a copy while the old state of the instance or index is kept. ODABA^{NG} supports different ways of data versioning.

Instance versioning allows keeping several versions of an object instance, but does not include indexes and relationships. Instance versioning is easy to handle, but often not sufficient.

With consistent versioning, ODABA^{NG} provides a versioning feature, which completely preserves the data base state at certain points in time. Using consistent versioning, one may always go back in time in order to browse previous database states, but changing the past is not possible.

Consistent versioning is also used for creating schema versions in the dictionary. A new schema version is just a new database version in the dictionary. Since previous schema definition can still be read, this technique is applied for “online schema evolution” in order to update older instances to the new schema version.

5 No chance without tools

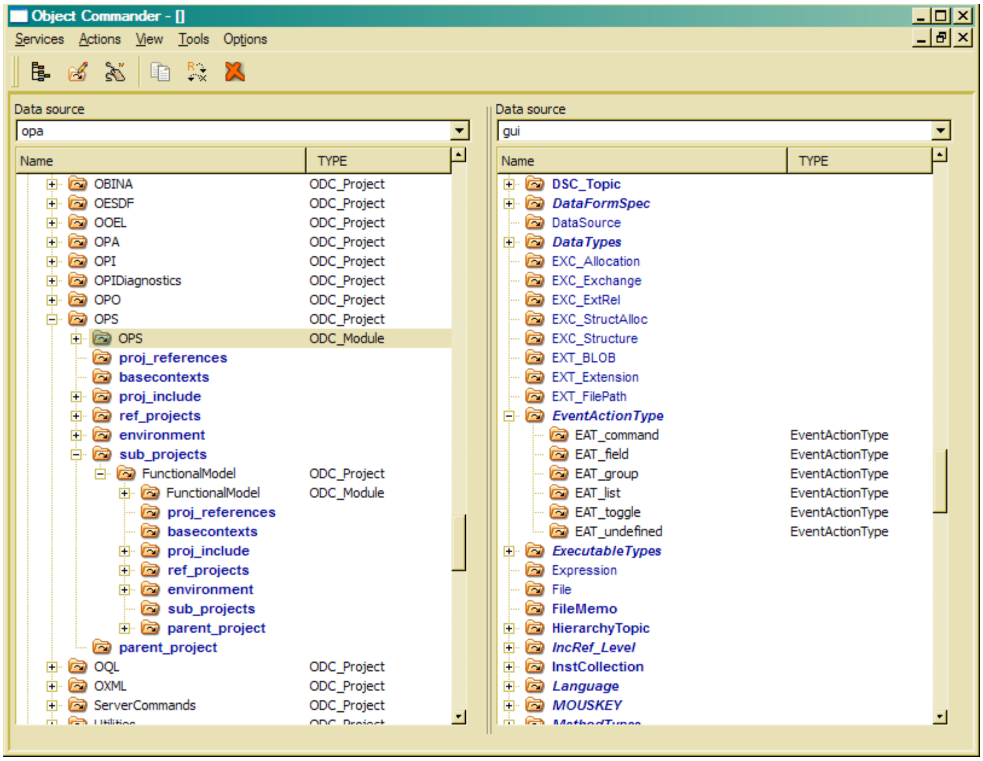
Every database management system (DBMS) needs tools, even the best one. Therefore, the ODABA team has invested a lot in order to provide powerful and ergonomic tools for maintenance and development.

But we did more than this. With ODABA^{NG} development tools you get all the power to customize the same tools according to your specific requirements.

The annoying administration

Compared with other DBMS, the administration effort for ODABA^{NG} is low. Nevertheless, especially complex systems require appropriate tools for data maintenance. Unexpected results may appear resulting from unexpected data as consequence of complex transactions.

One way for analysing data in the database is OSI, which allows accessing each bit of data. Another possibility is a data browser. With the „Object Commander" ODABA^{NG} provides a tool, whit a look and feel of well known Commanders as Norton, Total or Midnight Commander, which may look familiar to many users.



Data source
opa

Name	TYPE
OBINA	ODC_Project
OESDF	ODC_Project
OOEL	ODC_Project
OPA	ODC_Project
OPI	ODC_Project
OPIDiagnostics	ODC_Project
OPO	ODC_Project
OPS	ODC_Project
OPS	ODC_Module
proj_references	
basecontexts	
proj_include	
ref_projects	
environment	
sub_projects	
FunctionalModel	ODC_Project
FunctionalModel	ODC_Module
proj_references	
basecontexts	
proj_include	
ref_projects	
environment	
sub_projects	
parent_project	
parent_project	
OQL	ODC_Project
OXML	ODC_Project
ServerCommands	ODC_Project

Data source
gui

Name	TYPE
DSC_Topic	
DataFormSpec	
DataSource	
DataTypes	
EXC_Allocation	
EXC_Exchange	
EXC_ExtRel	
EXC_StructAlloc	
EXC_Structure	
EXT_BLOB	
EXT_Extension	
EXT_FilePath	
EventActionType	
EAT_command	EventActionType
EAT_field	EventActionType
EAT_group	EventActionType
EAT_list	EventActionType
EAT_toggle	EventActionType
EAT_undefined	EventActionType
ExecutableTypes	
Expression	
File	
FileMemo	
HierarchyTopic	
IncRef_Level	
InstCollection	
Language	
MOUSKEY	

Integrated in the object commander you will find several functions for database maintenance, consistence checks and repair tools.

For people who like console applications, corresponding utilities might be called via console commands. A pendant to the Object Commander is the **OShell**, which is a console application similar to the DOS or Bash console. Within the OShell, the database can be browsed like browsing a file system.

The right development tools

A terminology-oriented development environment begins, of course, with the word (definitions in the terminology model) and ends with words (the documentation). Besides, the complete development process is supported by document topics and notices.

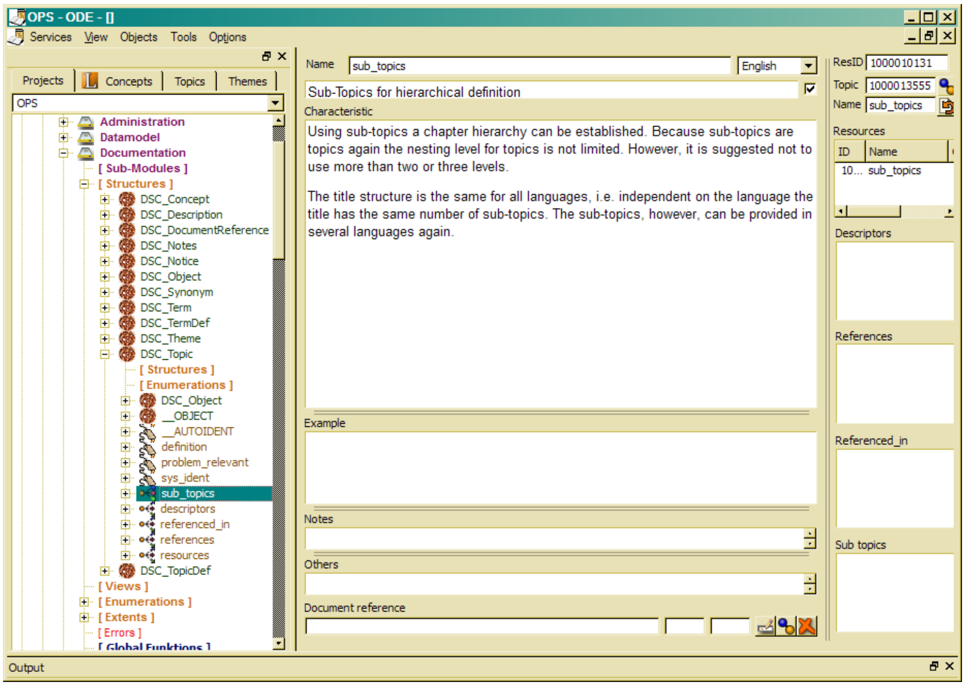
Needless to say that development is possible without ODABA^{NG} development tools, simply referring to OSI/ODL and application program interfaces (API).

When getting deeper into ODABA and its philosophy, the ODABA^{NG} Development Environment (ODE) offers a number of additional features and enhancements. Moreover, you may extend or customize the tools, which are delivered with all the sources and resource databases.

The ODE provides a series of tools that completely support the development process from the problem analysis up to the final product and its documentation. Just-in-time documentation is a basic principle all ODE tools are based on. Beside documentation of development resources, concept definitions, themes and hierarchical topic structures are supported.

The Terminology model - Terminus

Development within the ODE starts in the terminology model (problem analysis) supported by **Terminus**. Here, the problem can be described in detail within a terminology model (domain model). From the information provided in the topics, documents, UML graphs, HTML sites and other types of presentation might be generated.



Name: sub_topics English

Sub-Topics for hierarchical definition

Characteristic

Using sub-topics a chapter hierarchy can be established. Because sub-topics are topics again the nesting level for topics is not limited. However, it is suggested not to use more than two or three levels.

The title structure is the same for all languages, i.e. independent on the language the title has the same number of sub-topics. The sub-topics, however, can be provided in several languages again.

Example

Notes

Others

Document reference

ResID 1000010131

Topic 1000013555

Name sub_topics

Resources

ID	Name
10...	sub_topics

Descriptors

References

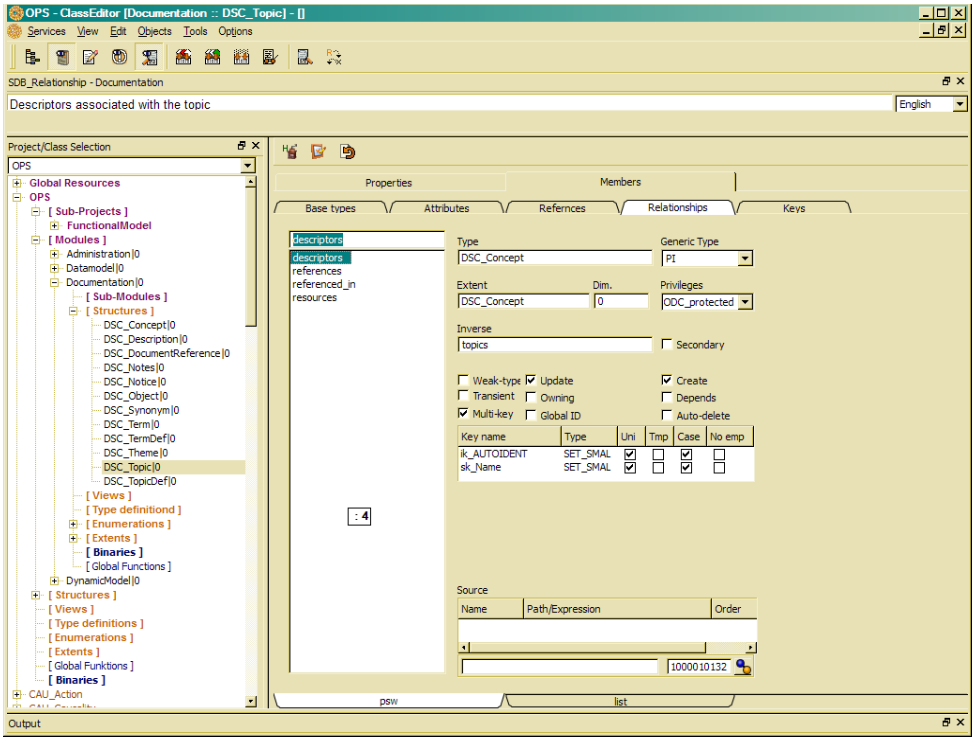
Referenced_in

Sub topics

Terminus is not yet completed, but the prototype available already supports many important features. The first complete Terminus version is expected in 2011.

From terminology model to data model - Class Editor

Concepts in a terminology model become object types in the object model, which can be processed with the **Class Editor**. Since the terminology model contains up to 80% of the schema information, expanding the terminology model to an object model becomes really easy.


OPS - ClassEditor [Documentation :: DSC_Topic] - []
Services View Edit Objects Tools Options
SDB_Relationship - Documentation
Project/Class Selection
Properties
Members
Base types
Attributes
Refernces
Relationships
Keys
psw
list
Output

Descriptors associated with the topic — English

OPS
- Global Resources
- OPS
  - [Sub-Projects]
    - FunctionalModel
  - [Modules]
    - Administration|0
    - Datamodel|0
    - Documentation|0
      - [Sub-Modules]
      - [Structures]
        - DSC_Concept|0
        - DSC_Description|0
        - DSC_DocumentReference|0
        - DSC_Notes|0
        - DSC_Notice|0
        - DSC_Object|0
        - DSC_Synonym|0
        - DSC_Term|0
        - DSC_TermDef|0
        - DSC_Theme|0
        - DSC_Topic|0
        - DSC_TopicDef|0
      - [Views]
      - [Type definitiond]
      - [Enumerations]
      - [Extents]
      - [Binaries]
      - [Global Functions]
    - DynamicModel|0
  - [Structures]
  - [Views]
  - [Type definitions]
  - [Enumerations]
  - [Extents]
  - [Global Funktions]
  - [Binaries]
- CAU_Action

descriptors
- descriptors
- references
- referenced_in
- resources

: 4

Type: DSC_Concept — Generic Type: PI
Extent: DSC_Concept — Dim.: 0 — Privileges: ODC_protected
Inverse: topics — ☐ Secondary

☐ Weak-type ☑ Update ☑ Create
☐ Transient ☐ Owning ☐ Depends
☑ Multi-key ☐ Global ID ☐ Auto-delete

Key name	Type	Uni	Tmp	Case	No emp
ik_AUTOIDENT	SET_SMAL	☑	☐	☑	☐
sk_Name	SET_SMAL	☑	☐	☑	☐

Source

Name	Path/Expression	Order

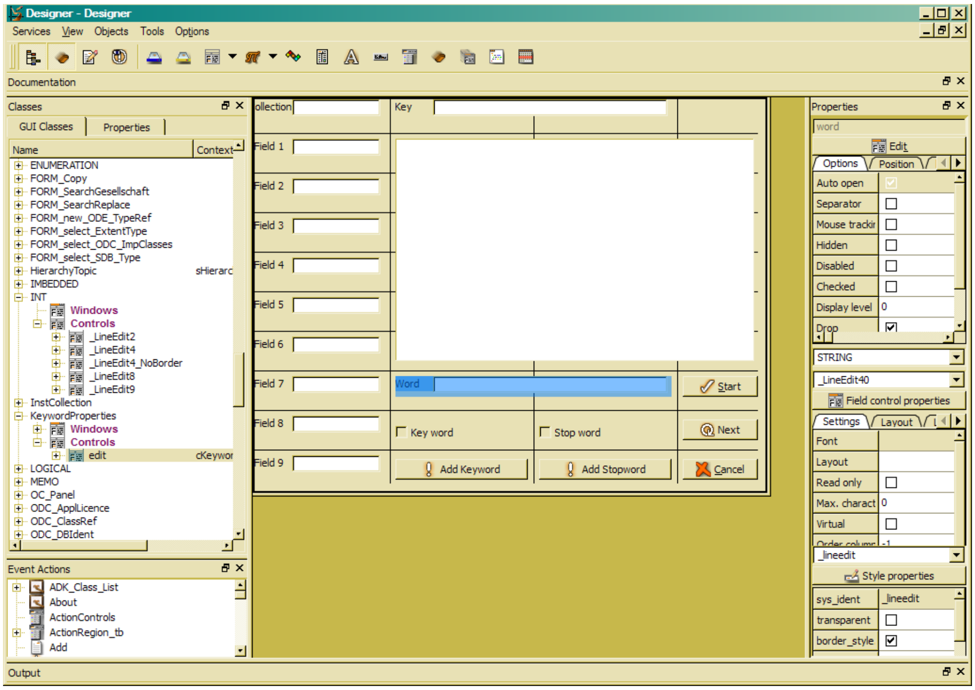
1000010132

The document topics linked with the concepts and characteristics automatically become documentation topics of object types and properties. Documentation can be shown and improved while defining details of the data model.

Several wizards support the developer in order to complete rather complex data definitions. After finishing the data model, it will be checked. Errors are reported and after correcting the errors, the schema changes from the development state to the production state.

Application design - Designer

By means of the **Designer**, a GUI application can be designed immediately based on the object model defined in the Class Editor. Due to the ADL technology [6] a well functioning prototype can be provided without writing a single line of program.



Classes:

```
+ ENUMERATION
+ FORM_Copy
+ FORM_SearchGesellschaft
+ FORM_SearchReplace
+ FORM_new_ODE_TypeRef
+ FORM_select_ExtentType
+ FORM_select_ODC_ImpClasses
+ FORM_select_SDB_Type
+ HierarchyTopic                 sHierarc
+ IMBEDDED
- INT
    Windows
  - Controls
    + _LineEdit2
    + _LineEdit4
    + _LineEdit4_NoBorder
    + _LineEdit8
    + _LineEdit9
+ InstCollection
- KeywordProperties
  + Windows
  - Controls
    + edit                       cKeywor
+ LOGICAL
+ MEMO
+ OC_Panel
+ ODC_ApplLicence
+ ODC_ClassRef
+ ODC_DBIdent
```

Event Actions:

```
+ ADK_Class_List
  About
  ActionControls
+ ActionRegion_tb
  Add
```

Auto open	✓
Separator	☐
Mouse trackir	☐
Hidden	☐
Disabled	☐
Checked	☐
Display level	0
Drop	✓

Font	
Layout	
Read only	☐
Max. charact	0
Virtual	☐
Order colum	-1

sys_ident	_lineedit
transparent	☐
border_style	✓

The designer provides design patterns for typical application elements (dialog, list and tree browser, virtual tab, tree application, wizard etc), which generate complex design elements for the application.

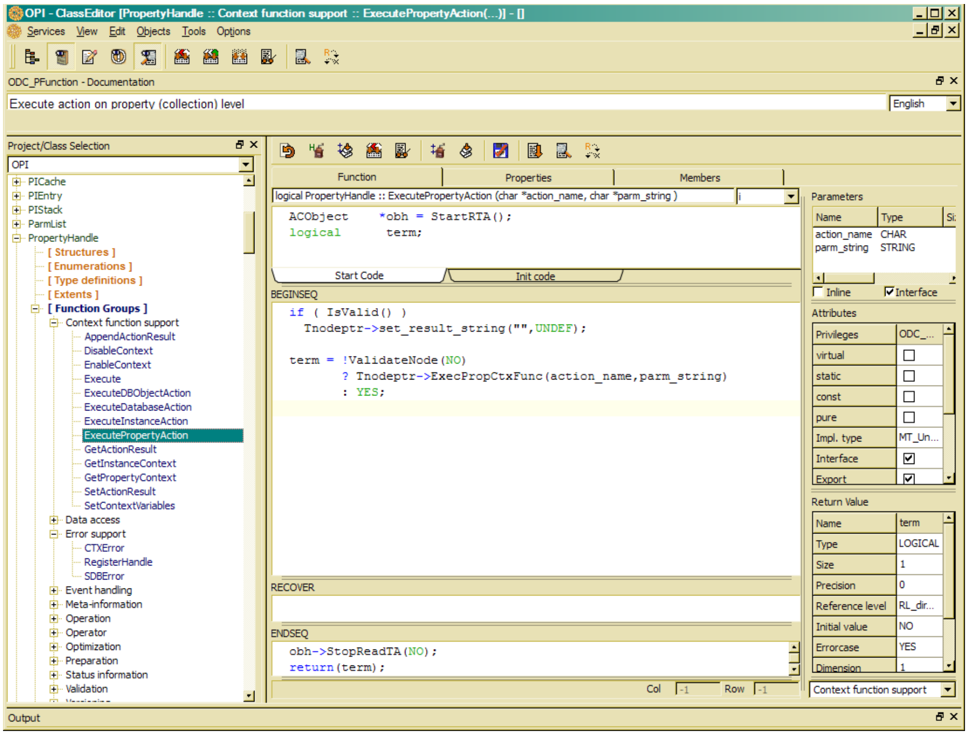
All forms and controls, as well as regions or columns in tables and trees are referring to properties or extents as data source. A GUI framework feeds the GUI elements with data from the defined data sources and ADL cares about updating data whenever necessary.

Since ODABA^{NG} GUI elements are very powerful, the effort for application development can be reduced to 20% and less. Moreover, GUI resources provide documentation features, which can be referred to later on in the application documentation or generated online-help.

Programming in ODE - Class Editor / Designer

In order to define business rules or more sophisticated application behaviour, programming becomes necessary in many cases. For implementing functions in one of the supported programming languages, one may use any well known programming environment.

Continuing with ODE, however, one benefits not only from automatic code generation, but also from just-in-time documentation feature, which helps to keep documentation up-to-date.



Execute action on property (collection) level

logical PropertyHandle :: ExecutePropertyAction (char *action_name, char *parm_string)

```
  ACObject     *obh = StartRTA();
  logical       term;
```

BEGINSEQ

```
  if ( IsValid() )
    Tnodeptr->set_result_string("",UNDEF);

  term = !ValidateNode(NO)
         ? Tnodeptr->ExecPropCtxFunc(action_name,parm_string)
         : YES;
```

RECOVER

ENDSEQ

```
  obh->StopReadTA(NO);
  return(term);
```

Name	Type
action_name	CHAR
parm_string	STRING

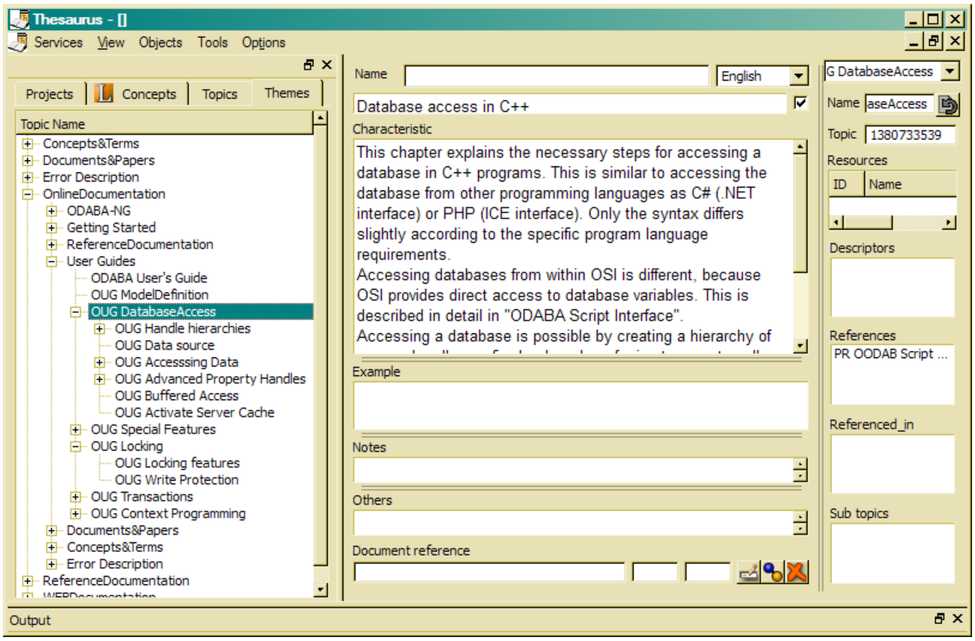
Privileges	ODC_...
virtual	☐
static	☐
const	☐
pure	☐
Impl. type	MT_Un...
Interface	☑
Export	☑

Name	term
Type	LOGICAL
Size	1
Precision	0
Reference level	RL_dir...
Initial value	NO
Errorcase	YES
Dimension	1

In order to implement application rules, functions might also be implemented with similar comfort within the designer.

Application documentation – Terminus

The last step is finishing the documentation, which, hopefully, had been updated in all development phases. Nevertheless, additional documentation topics are usually required and topic hierarchies have to be created in order to provide useful and readable documentation.



```
Concepts&Terms
Documents&Papers
Error Description
OnlineDocumentation
  ODABA-NG
  Getting Started
  ReferenceDocumentation
  User Guides
    ODABA User's Guide
    OUG ModelDefinition
    OUG DatabaseAccess
      OUG Handle hierarchies
      OUG Data source
      OUG Accesssing Data
      OUG Advanced Property Handles
      OUG Buffered Access
      OUG Activate Server Cache
    OUG Special Features
    OUG Locking
      OUG Locking features
      OUG Write Protection
    OUG Transactions
    OUG Context Programming
  Documents&Papers
  Concepts&Terms
  Error Description
ReferenceDocumentation
```

English

Database access in C++

This chapter explains the necessary steps for accessing a database in C++ programs. This is similar to accessing the database from other programming languages as C# (.NET interface) or PHP (ICE interface). Only the syntax differs slightly according to the specific program language requirements.
Accessing databases from within OSI is different, because OSI provides direct access to database variables. This is described in detail in "ODABA Script Interface".
Accessing a database is possible by creating a hierarchy of

G DatabaseAccess
Name: aseAccess
Topic: 1380733539
References: PR OODAB Script ...

Creating topics and topic hierarchies will be done with Terminus, again. Terminus also provides features for generating different document types and online help for GUI applications.

Command line Tools

A series of tools described in „Database Utilities“ [12] is provided in order to support maintenance and database administration, as well as running server commands from a remote console.

6 And what does all this cost?

ODABA^{NG} is one of the few systems, which are available as LINUX and as MS Windows version with a GPL (GNU Public License). As long as you publish the source code of the system or application that you have build with ODABA, you need not to pay anything. The only third party product, used in ODABA is QT (Nokia), which might be used as well since it is based on a LGPL.

When using ODABA for developments, which are not intended to become open source products, a commercial development license becomes necessary. As long as you develop and maintain a non open source project you need such a commercial licence.

There are no run-time licences required, except when using external software from within ODABA as Oracle or MS SQL Server.

Finally, you may join the ODABA Development Foundation, which supports further ODABA development and allows you also to influence further development directions and themes.

7 More Documentation!

On our documentation download site at

www.run-software.com/content/downloads/documentation

all ODABA and related documents are published an can be downloaded. Documents have been divided into 6 groups:

0 – Concepts

1 – Technical details

2. – Installation and usage

3 – Services and Tools

4 – ODABA development environment

P – Publications

In addition, the are two online documentation systems available on our WEB sites:

All about the ODABA database
(http://www.run-software.com/content/documentation/odaba)

ODABA GUI Design and development
(http://www.run-software.com/content/documentation/odabagui)

8 Downloads and support

An evaluation version can be downloaded from

http://www.run-software.com/content/downloads

We suggest, however, to register in order to receive news and information about new ODABA versions.

We try to respond to all questions from the open source community as quick as possible, but we will reply to registered users, only, since others will not pass the SPAM barrier. We provide, however, also mandatory support for commercial licence holders within two working days. Moreover, we are open for any sort of special agreements with our customers.

We also offer training, coaching, average support, application development and other services.

9 References

References referred to author **run** are available at:

www.run-software.com/content/downloads/documentation

Also, most documents from Karge, R are available there.

[1] Karge R.: *A terminology model approach for managing statistical metadata*, Open Forum on metadata registries, Berlin, 2005, www.run-software.com/content/downloads/documentation/TerminologyModel.doc

[2] ODMG; *The Object Data Standard ODMG 3.0,* Academic Press, 2000

[3] run: *ODABA Online documentation,* Berlin, 2007-2010

[4] Karge R.: *Real Objects*, Addison Wesley (Deutschland) GmbH, Bonn, 1996,

[5] run; *ODABA Technical Overview,* Berlin, 1995-2007- 2009

[6] Karge R.: *Active Data Link*, run Software, Berlin, 2002-2007, www.run-software.com/download/UnifiedDatabaseTheory.doc

[7] Karge R.: *Unified Database Theory*, run Software, Berlin, 2003,

[8] run; *ODABA Database Concepts,* Berlin, 1995-2010

[9] ISO 704: *Terminology work – Principles and methods*

[10] ISO 1087: *Terminology work – Vocabulary*

[11] Fowler, M.: *Patterns of Enterprise Application Architecture*, Addison Wesley, 2003

[12] run; *ODABA Database Utilities,* Berlin, 1995-2010