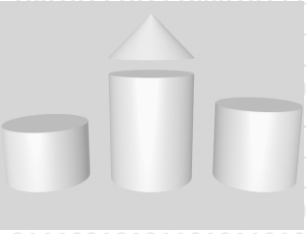


ODABA^{NG}

ODABA 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 Einführung

Mit ODABA^{NG} wird das erste terminologie-orientierte Datenbanksystem bereitgestellt, das einen nahtlosen Übergang von der Entwicklung des Fachkonzeptes zur benutzerfreundlichen Anwendung ermöglicht. Die Daten in der Datenbank so sehen, wie sie im Terminologiemodell definiert wurden, das ist der Wunsch vieler Kunden, dem ODABA ein großes Stück entgegenkommt.

Da terminologie-orientierte Datenbanken eine konzeptionelle Erweiterung der objekt-orientierten Datenbanken sind, kann auf vielen bekannten Standards und Features aufgesetzt werden.

2 Vom Wort zur Anwendung

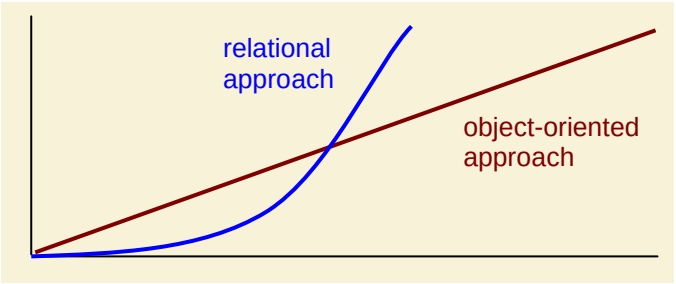
ODABA^{NG} ist von der Philosophie her terminologie-orientiert. Grundlage aller ODABA^{NG}-Anwendungen sind Terminologiemodelle[1] (erweiterte Concept Systems [9][10]).

Ein weiterer Grundgedanke, der in allen ODABA-Entwicklungsphasen Pate stand, ist Atomisierung. Jede Information wird in kleinsten Einheiten gespeichert, weil das die einfachste Möglichkeit ist, diese Teile später zu beliebigen Kompositionen zusammenzuführen. So könne z.B. aus Dokumentations-Topics einer Terminologiedefinition WEB-Seiten, MS Word- oder Open Office-Dokumente, UML-Diagramme, aber auch das ODABA^{NG}-Objektmodell erzeugt werden.

Es ist nahe liegend, terminologie-orientierte Konzepte in einem OODBMS umzusetzen. ODABA^{NG} orientiert sich dabei an den Object Data Standard ODMG 3.0[2], geht in vielen Punkten aber über diesen Standard hinaus.

Objektorientiert oder relational

Eine Anwendung objektorientiert zu entwickeln impliziert meistens einen höheren Startaufwand, bringt aber langfristig in der Wartung und in der Wachstumsphase eines Projektes erhebliche Einsparungen [11]. Während der Aufwand in relationalen Projekten exponentiell wächst, gehr man bei objektorientierten Entwicklungen eher von einem linearen Anstieg aus. Durch die Verwendung eines OODBMS entfällt außerdem die Implementierung des Data-Layers, d.h. praktisch: keine Select, keine Join Anweisungen. Es kann direkt und sofort auf die Daten zugegriffen werden.



Im Allgemeinen unterscheiden sich OODBMS von Relationalen Datenbankbetriebsystemen (RDBMS) dadurch, dass sie neben dem Daten- oder Objektmodell das funktionale Modell unterstützen. Theoretisch ist auch noch das Dynamische Modell Bestandteil der objektorientierten Modelle, wird aber selten und dann auch nur sehr fragmentarisch von den OODBMS unterstützt. ODABA^{NG} unterstützt das dynamische Modell durch System- und Anwendungsereignisse, wodurch ereignisgesteuerte Anwendungen entwickelt werden können, die auf der Active Data Link Technology [6] beruhen.

Das Objektmodell ist ausdrucksstärker als das relationalen Modell, d.h. vieles, was in relationalen Datenbankanwendungen programmiert werden muss, ist in objektorientierten Datenbankanwendungen eine Frage der Definition.

Da viele objektorientierte Datenbanken die Schemadefinition mit der Methodendefinition (z.B. in Klassendefinitionen für C++) verknüpfen, entsteht der Eindruck, dass objektorientierte Datenbanken integrierter Bestandteil von Programmiersprachen oder deren Erweiterungen sind. So lassen z.B. bekannte OODBMS wie Fast Objects oder ObjectStore nur einen Klassentyp pro Objekttyp zu. Dieser enge Zusammenhang von Programmiersprache und Objektmodell ist theoretisch nicht zu begründen und wirkt sich in der Praxis oft hinderlich aus. Deshalb unterstützt ODABA^{NG} verschiedene Implementierungsklassen für jeden Objekttyp.

ODABA^{NG} kann mehr

ODABA^{NG} zeigt, dass auch ein klassischer Zugang zu einem OODBMS möglich ist, bei dem Objektmodell und funktionales Modell getrennt definiert werden. ODABA^{NG} erreicht dies durch die Trennung von Datentyp und Klasse. Da Klassen Spezialisierungen von Datentypen sind, kann das Objektmodell unabhängig vom funktionalen Modell entwickelt werden. Dort, wo es erforderlich wird, können Datentypen zu verschiedenen Klassen spezialisiert werden. Dadurch wird die Abhängigkeit von einer Programmiersprache aufgehoben und das OODBMS somit flexibler.

Da in ODABA^{NG} ein und derselbe Objekttyp Basis für unterschiedliche Klassentypen sein kann („Person“ kann gleichzeitig als C++, ODABA Script-Klasse und GUI-Klasse implementiert werden), werden die Klassendefinition aus dem Objektmodell abgeleitet.

Die strikte Trennung von Objektmodell und funktionalem Modell erhöht die Flexibilität der Objekttypen, da sie nicht auf einen Klassentyp (z.B. C++ Klasse) festgelegt werden.

ODABA^{NG} orientiert sich weitgehend am Object Data Standard ODMG 3.0 [2] und unterstützt dessen Anforderungen hinsichtlich des Objektmodells, wie z.B.

- Vererbung
- Relationships mit inversen Referenzen
- Enumerations
- Module/Namespace-Hierarchie

Aber ODABA^{NG} ist dieser Standard nicht genug. Es gibt in vielen Punkten entscheidende Erweiterungen, die sich vor allem aus den Anforderungen des Terminologiemodells ableiten. So unterstützt ODABA^{NG} z.B. folgende Erweiterungen:

- **Shared inheritance** erlaubt es, mehrere Objektinstanzen gleichzeitig von einer anderen abzuleiten (*Mehrfachspezialisierung*).
- ODABA^{NG} unterstützt die Modellierung von **Mengenbeziehungen**, d.h. für beliebige Mengen (Extents, Relationships) können Obermengen, Vereinigungen oder Durchschnitte modelliert werden, die von ODABA^{NG} automatisch mitgepflegt werden. Das führt zu erheblichen Einsparungen in der Applikationsentwicklung.
- Enumerations (**Klassifikationen**) können hierarchisch definiert werden.
- Die Module/Namespace-Hierarchie wurde durch eine **Projekt**ebene ergänzt.
- Textattribute können **mehrsprachig** sein.

Dies ist nur ein Teil der praktisch wertvollen Erweiterungen, die das ODABA^{NG} Objektmodell bereithält. Details sind in der ODABA^{NG} Online-Dokumentation [3] beschrieben.

Das funktionale Model von ODABA^{NG} unterstützt C++, OSI (ODABA Script Interface) und GUI-Klassen (Präsentationsmethoden). Über ein .NET-Interface kann mittels C#, MS Visual Basic oder Java auf ODABA zugegriffen werden.

In größeren Projekten, in denen eine strikte Trennung von Database-, Business- und Application-Layer erforderlich wird, zeigt sich schnell, dass persistente Objekttypen nicht genug Information besitzen, um eine zweckmäßige Business- und Applikationslogik zu implementieren. Diese beiden Layer benötigen häufig zusätzliche Informationen über den Applikationszustand.

Deshalb werden in ODABA^{NG} sogenannte **Kontextklassen** unterstützt, die die erforderlichen Laufzeitinformationen bereitstellen und es möglich machen, auch das Verhalten einzelner Properties des Objektmodels zu definieren. Businesslogik wird in ODABA^{NG} über Datenbankkontextklassen implementiert, die mit Objekttypen des Objektmodels oder deren Properties verbunden werden. Ähnlich verhält es sich mit der Applikationslogik, die in Applikationskontextklassen implementiert wird, die an Controls oder andere Elemente der Oberfläche gebunden sind.

Das, was auf den ersten Blick kompliziert wirkt, macht die Applikationsentwicklung tatsächlich einfacher, da die die strikte Trennung der Ebenen die Übersichtlichkeit erhöht.

Kausale Zusammenhänge werden in Ansätzen in relationalen Datenbanken durch Trigger abgebildet. In OODBMS (also auch in ODABA^{NG}) werden zu diesem Zweck meistens Events und Event-Handler definiert. Neben dem Objektmodell und dem funktionalen Modell stellt ODABA^{NG} explizit ein **Kausalmodell** (oder dynamisches Modell) bereit, das die Definition von Ereignissen durch relevante Zustandsübergänge ermöglicht. Ein Mangel bei der Nutzung von Event-Handlern ist die Tatsache, das ein Ereignis in der Regel nur an das Objekt gemeldet wird, das das Ereignis ausgelöst hat, also Änderungsereignisse zu den Personendaten

werden an die Personinstanz geschickt, die sich gerade geändert hat. Ein Kausalitätsschema hingegen beschreibt zusätzlich die Beziehungen zwischen dem ereignisauslösendem Objekt und den auf das Ereignis reagierenden Objekten. Für die Unterstützung eines derartigen Modells sind bisher keine weiteren Beispiele bekannt und ODABA^{NG} beschreitet hier in der Tat Neuland.

Zur Abrundung sind in ODABA^{NG} zwei weitere Modelle standardmäßig integriert – nicht aufsehenerregend, aber praktisch:

Das **Dokumentationsmodell** ist eine Konsequenz aus der terminologischen Orientierung von ODABA. Es ist ISO704 [9] kompatibel, besitzt aber entscheidende Erweiterungen. Neben Begriffsdefinitionen (Konzepten) können alle Entwicklungsobjekte (Datentyp, Property, Funktion, Parameter usw.) in Dokumentbausteinen (Topics) dokumentiert werden. Die direkte Verbindung von Dokumentation und dem dokumentierten Gegenstand hat sich als großer praktischer Vorteil erwiesen. Ergänzt wird das Dokumentationsmodell durch Notizen, die ebenfalls mit beliebigen Entwicklungsobjekten assoziiert werden können.

Das **Administrationsmodell** ist eine weiterer Modellzusatz, den ODABA^{NG} aus praktischen Gründen bereitstellt, um Nutzer und Prozessverwaltung zu unterstützen Nutzer und Nutzergruppen werden aber auch für das Benachrichtigungssystem benötigt, das es ermöglicht, Notizen an andere Entwickler zu versenden.

Die Vielfalt der Möglichkeiten von ODABA^{NG} ist im „Technical Overview“ [5] und in den „Database Concepts“ [8] detailliert beschrieben.

3 Technische Spezialitäten

Stärker als viele anderen Datenbanken ist ODABA darauf ausgerichtet, verschiedenartige Schnittstellen zu anderen Systemen bereitzustellen. Das betrifft nicht nur den Datenaustausch mit anderen Datenbanken und Dokumenten, sondern auch die Unterstützung Ereignisgesteuerter Anwendungen und die verschiedensten Möglichkeiten, mit der Datenbank zu kommunizieren.

Ereignisse en masse

Man kann mit ODABA^{NG} einfache (application driven) Systeme entwickeln. Wesentlich effizienter sind aber ereignisgesteuerte (event driven) Systeme, da diese die Möglichkeit bieten, eine konsistente und robuste Business-Logik zu implementieren. Deshalb erzeugt ODABA^{NG} eine Vielzahl von Systemereignissen auf der Ebene der Property-Instanzen und der Objektinstanzen. Neben Systemereignissen kann der Entwickler relevante Zustandsübergänge als Ereignis definiere, die dann (auch an andere Objektinstanzen) signalisiert werden.

Event-Handler werden in Datenbankkontextklassen als OSI-, C++- oder .NET-Funktion implementiert. Dort werden sie unmittelbar beim Auftreten des Ereignisses behandelt (interne Ereignisse).

Für eine **Active Database** Applikation ist das jedoch nicht ausreichend, da die Anwendung über Ereignisse informiert werden muss. Im Gegensatz zu den meisten anderen DBMS erfüllt ODABA^{NG} alle Anforderungen die **Active Data Link** [6] an das DBMS stellt. Dazu generiert ODABA^{NG} zusätzlich Handler-Ereignisse, d.h. die Ereignisse werden über Handle-Objekte an die Applikation geleitet. Im Gegensatz zur Business-Logik, die unmittelbar reagieren muss, darf die Application-Logik erst reagieren, wenn ein Datenbankzugriff beendet ist. ODABA^{NG} kapselt deshalb Datenbankzugriffe in sogenannte Read-Transaktionen, die die Handler-Ereignisse puffern und gleichzeitig optimieren.

Interne und Handle-Ereignisse sind nicht prozessübergreifend. Prozessübergreifende Ereignisse werden generiert, wenn die Anwendungen über den ODABA^{NG} Server kommunizieren. In diesem Fall werden Server-Ereignisse asynchron zum Client versendet und können dort verarbeitet werden. Serverevents werden eingeschränkt auch durch den Replication-Server generiert.

Reden mit anderen

Applikationsentwicklung mit ODABA^{NG} ist auf verschiedenen Ebenen möglich. ODABA^{NG} Applikationen können implementiert werden, ohne die ODABA^{NG} Entwicklungsumgebung (ODE) zu benutzen. Das Datenmodel kann in einem ODMG kompatiblen ODL-Script (Object Definition Language)) bereitgestellt und mit Hilfe des Schema Loaders in die Schemadatenbank geladen werden. Die Applikation kann dann in C++, C# oder Visual Basic (über ein .NET Interface), aber auch In OSI-Scripten implementiert werden. Das bereitgestellte ICE-Interface ermöglicht darüber hinaus den Zugriff auf die ODABA^{NG} Daten z.B. aus PHP oder JAVA.

Mit der ODE wird jedoch auch eine komfortable Entwicklungsumgebung bereitgestellt, mit deren Hilfe Terminologiemodell, Datenmodell, Anwendungsdesign und Implementierung wesentlich einfacher werden.

Jeder will mal scripten

Neben den Standardinterfaces (COM, ICE, C++) bietet ODABA^{NG} eine komfortable Scriptsprache an. Das ODABA Script Interface (OSI) unterstützt Datendefinition (ODL), Datenanfragen (OQL) und Datenmanipulation (OML). Von der Syntax her ist OSI an JAVA bzw. C# angelehnt und somit für C++, C# oder JAVA-Entwickler leicht verständlich. OSI kann sowohl auf alle Elemente des Objektmodells zugreifen, als auch auf lokal definierte Applikationselemente (applikationsinterne Datentypen, lokale Variablen). OSI betrachtet die gesamte Datenbank wie einen großen Adressraum, in dem die Variablen gespeichert sind. OSI Variablen können vom atomaren Feld über einzelne Objektinstanzen bis hin zu ganzen Mengen alles enthalten.

Die Möglichkeit, Templates (Dokumentmuster) als OSI-Expressions zu definieren, erleichtert die Generierung von Dokumenten und HTML-Seiten.

4 Technische Überraschungen

ODABA^{NG} halt eine Vielzahl technischer Überraschungen bereit. Zum einen ist ODABA^{NG} in verschiedenster Weise skalierbar. Verschiedene Hardware und Systemplattformen werden ebenso unterstützt wie unterschiedliche Client/Server Modelle oder die Speicherung der Daten in unterschiedlichsten Datenbanken. Aber auch bei Standardfeatures wie Transaktionen oder Locking halt ODABA^{NG} immer eine kleine zusätzliche Überraschung bereit, die das Arbeiten erleichtert.

Flexibilität in jeder Hinsicht

ODABA^{NG} ist auf verschiedenen Ebenen plattformunabhängig. Zum einen können ODABA^{NG}-Anwendungen auf verschiedenen MS Windows, LINUX und Sun Solaris Betriebssystemen ausgeführt werden. Auch ODABA^{NG} GUI-Applikationen sind dabei nicht ausgenommen, da mit QT ein plattformunabhängiges GUI-Subsystem gewählt wurde. Die Datenbanken selbst sind in der Lage, Daten im PIF (Plattform Independent Format) abzulegen. Dadurch wird es möglich, Datenbanken zwischen verschiedenen Hardware-Plattformen durch einfaches Kopieren auszutauschen.

Derzeit unterstützt ODABA^{NG} drei verschiedene Servertypen.

- File-Server
- Replication-Server
- ODABA-Server

File-Server können wegen ihres geringen Administrationsaufwandes einfach in lokalen Netzwerken eingesetzt werden, indem die File-Serverfunktionen des Betriebssystems genutzt werden.

Replication-Server sind ebenfalls „dumme“ Server, die vor allem für WEB-Datenbanken eingesetzt werden. Beim Client wird ein lokales Duplikat der DB angelegt, das durch den Server bei Änderungen zeitnah aktualisiert wird. Dadurch verhalten sich Client-Anwendungen bezüglich ihrer Performance wie lokale Anwendungen, obwohl sich die Datenbank auf einem WEB-Server befindet.

Der ODABA-Server ist ebenfalls für lokale Netzwerke konzipiert. Im Gegensatz zum File-Server wird jedoch ein Großteil der Businesslogik durch den Server ausgeführt.

Auf die Applikationsentwicklung hat die Art des gewählten Servers nur geringen Einfluss. Das Servermodell kann also für jede Anwendung nach Zweckmäßigkeit ausgewählt und nach Belieben geändert werden.

Speichern ganz nach Belieben

Wie die die meisten DBMS stellt auch ODABA^{NG} Schnittstellen zu externen Dateiformaten bereit. Neben ESDF (Extended Self Delimiter Files, eine Erweiterung von CSV), unterstützt ODABA^{NG} Flat Files, OIF (Object Interchange Format) und XML, sowie den Zugriff auf verschiedene relationale Datenbanken.

Ein völlig neuer Weg wurde 2006 eingeschlagen, als in ODABA^{NG} die Vorraussetzungen für unterschiedliche Speicherformate (Multiple Storage Interface – MSI) für Daten und Metadaten geschaffen wurden. So kann sowohl das ODABA^{NG} Datenmodell (Dictionary) als auch Daten in XML Schema/XML gespeichert werden. Theoretisch kann eine ODABA^{NG}–Datenbank vollständig im XMLBase-Speicherformat abgelegt werden, was allerdings nicht besonders praktisch ist. Eingesetzt wird das XML-Format aber für den Datenaustausch und zur Verarbeitung von WSDL- bzw. SOAP-Anfragen.

Interessant wird MSI im Zusammenhang mit relationalen Datenbanken. Da der Informationsgehalt relationaler und objektorientierter Datenbanken theoretisch identisch ist [7], kann der Inhalt einer ODABA^{NG}-Datenbank auch in einer relationalen Datenbank abgelegt werden. Um den Kundenwünschen nach Lösungen mit speziellen Datenbanksystemen nachzukommen, können die Daten in einem dem Kunden entsprechenden Datenbanksystem gespeichert werden. Derzeit werden Oracle, MS SQL Server und MySQL unterstützt.

Konkurrenz – kein Problem

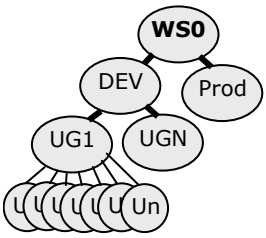
Mit Concurrency Control und Locking hat der ODABA^{NG}-Entwickler meistens wenig zu tun. ODABA^{NG} unterstützt zwar das explizite Sperren von Objektinstanzen und Mengen. Im Allgemeinen muss sich der Applikationsentwickler darum aber nicht kümmern, weil ODABA^{NG} implizite Lock-Mechanismen zum optimistischen oder pessimistischen Sperren automatisch bereitstellt.

Transaktionen

Natürlich ist eine Transaktionsverwaltung unbedingt erforderlich, da die Auswirkungen von Änderungen für den Applikationsentwickler oft nicht transparent sind.

Da jedes Ändern (Hinzufügen, Modifizieren, Löschen) zu weiteren Änderungen führen kann (z.B. Entfernen inverser Referenzen, oder Folgeaktionen der Business-Logik, die durch einem Event-Handler veranlasst werden), sichert

ODABA^{NG} durch eine interne (kurze) Transaktion die Konsistenz der Datenbank. Außerdem unterstützt ODABA^{NG} „lange“ Transaktionen, die bei Bedarf auch auf einem externen Speichermedium zwischengelagert werden. Zusätzlich kann man auf „sichere“ Transaktionen zurückgreifen, die auftretende Fehler beim Schreiben einer Transaktion verhindern. Die ODABA^{NG} Log-Dateien, die zum einen Änderungen protokollieren, und zum anderen Restore-Funktionalität unterstützen, sind ebenfalls ein gängiges Feature.



Aber auch bei den Transaktionen geht ODABA^{NG} einen Schritt weiter. „Workspaces“ (persistente Transaktionen) erlauben einem oder mehreren Anwendern in verschiedenen Workspaces sitzungs- bzw. prozessübergreifend zu arbeiten. Wie in jeder anderen Transaktion sind die Änderungen nur für den jeweiligen Mitarbeiter sichtbar. Wenn die Arbeiten nach Tagen (oder Wochen) abgeschlossen sind, können sie konsolidiert oder verworfen werden. Erst nach der Konsolidierung werden diese Änderungen für andere Nutzer sichtbar. Workspaces können Hierarchien bilden und somit z.B. auch für Nutzergruppen angelegt werden.

Daten versionieren

Datenbankversionierung ist ein nicht ganz übliches Feature. Versionierung bedeutet, dass Änderungen in einer neuen Kopie der Instanz oder eines Indexes abgelegt werden.

ODABA^{NG} unterstützt zwei Arten von Versionierung. Die eine ist die Instanzenversionierung, durch die es möglich wird, ältere Instanzenzustände zu erhalten. Da Relationships von dieser Versionierung nicht betroffen sind, ist diese Form der Versionierung oft nicht hinreichend. Deshalb unterstützt ODABA^{NG} zusätzlich die konsistente Versionierung, bei der der Zustand der gesamten Datenbank zu einem bestimmten Zeitpunkt „eingefroren“ wird.

Da konsistente Versionierung jederzeit die Sicht auf eine ältere Version der Datenbank ermöglicht, wird sie auch zur Schemaversionierung verwendet. Eine Schemaversion für ODABA ist somit einfach eine Datenbankversionierung der Ressourcen- oder Schemadatenbank. Die verschiedenen Schemaversionen werden dann für die „Online Schema Evolution“ verwendet, d.h. Objektinstanzen werden zur Laufzeit and die geänderten Typedefinitionen angepasst.

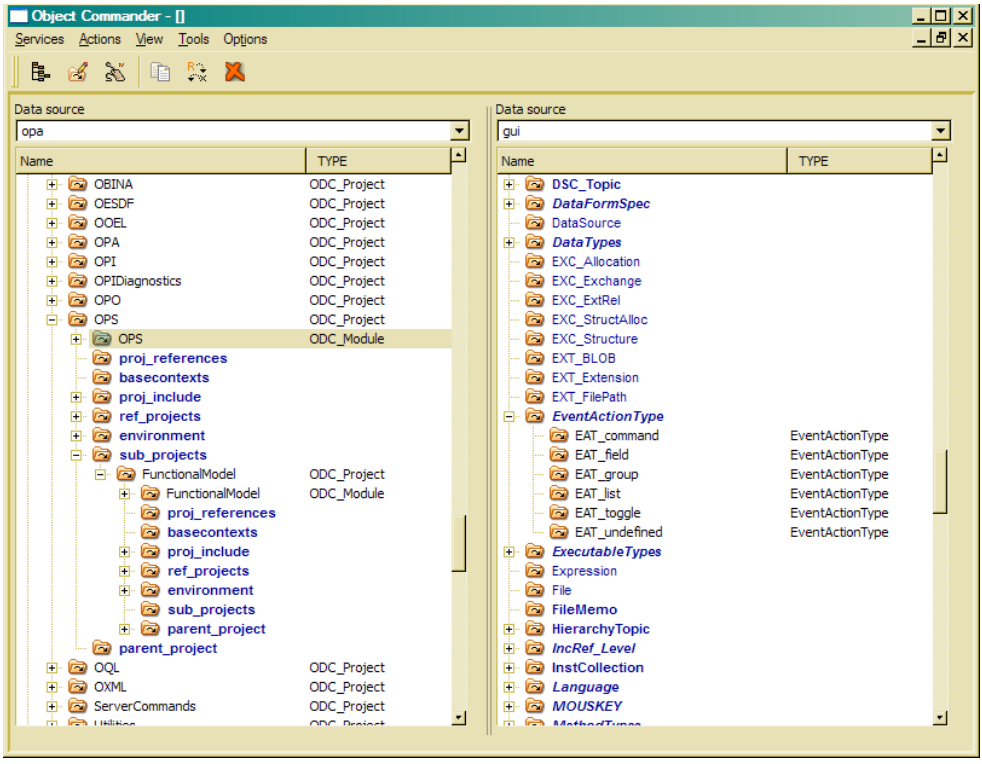
5 Ohne Werkzeuge geht nichts

Die beste Datenbank ist nicht viel wert ohne die rechten Werkzeuge. Deshalb hat Entwicklungsteam viel Aufwand in die Bereitstellung ergonomischer und leistungsstarker Werkzeuge für Wartung und Entwicklung.

Die leidige Verwaltung

Im Vergleich zu anderen DBMS ist der Administrationsaufwand in ODABA^{NG} gering. Dennoch benötigen gerade komplexe Systeme geeignete Tools für die Wartung der Daten, da unerwarteten Ergebnissen auftreten kommen, weil vielleicht nicht das in den Daten steht, was man dort erwartet.

Eine Analysemöglichkeit bieten Anfragesprachen (hier also OSI). Eine weitere sind Daten-Browser. ODABA^{NG} stellt mit dem „Object Commander“, der offenbar die guten Traditionen des Norton- oder Total Commanders aufgenommen hat, ein Tool bereit, das vielen sicher auf Anhieb bekannt vorkommt.



Im Object Commander integriert wurden verschiedene Funktionen zur Datenbankpflege, Konsistenzkontrolle und Reparatur.

Für Freunde der Textkonsole gibt es aber auch entsprechende Konsolkommandos. Das Pendant zum Object Commander ist die **OShell**, ein Konsolenprogramm, das genug Raum für Assoziationen zu bekannten DOS Shells lässt. Auch hier lässt sich die Datenbank wie ein Dateisystem browsen und manipulieren.

Die richtigen Werkzeuge

Eine terminologie-orientierte Entwicklung beginnt mit dem Wort (Terminologiemodell) und endet mit dem Wort (Dokumentation). Dabei wird der gesamte Entwicklungszyklus durch Dokumentationsobjekte (Topics) unterstützt.

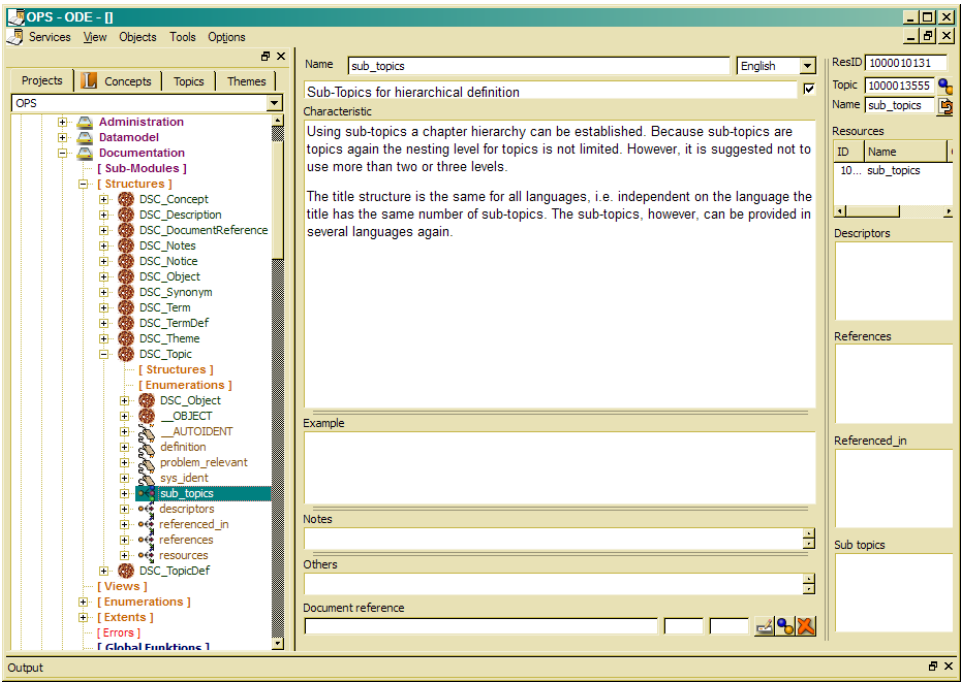
Natürlich kann der gesamte Entwicklungszyklus einer Applikation ohne ODABA^{NG}-Entwicklungswerkzeuge erfolgen, da alle erforderlichen Informationen über OSI Scripte bereitgestellt werden können.

Lässt man sich aber weiter auf die ODABA Philosophie ein, bietet die ODE (ODABA^{NG} Development Environment) eine Reihe von angenehmen Erleichterungen. Hinzu kommt, dass man die ODE relativ einfach an die eigenen Bedingungen anpassen kann, da die Quellen frei verfügbar sind.

Die ODE stellt eine Serie von Werkzeugen bereit, mit deren Hilfe eine Anwendung komplett von der Problemanalyse bis zur Dokumentation entwickelt werden kann. „Just In Time Documentation“ ist ein Grundprinzip, das allen ODE-Werkzeugen zugrunde liegt. Neben der Dokumentation der Applikationselemente werden Begriffsdefinitionen (Concept Systems) und hierarchische Zusammenstellungen von Topics unterstützt.

Das Terminologiemodell - Terminus

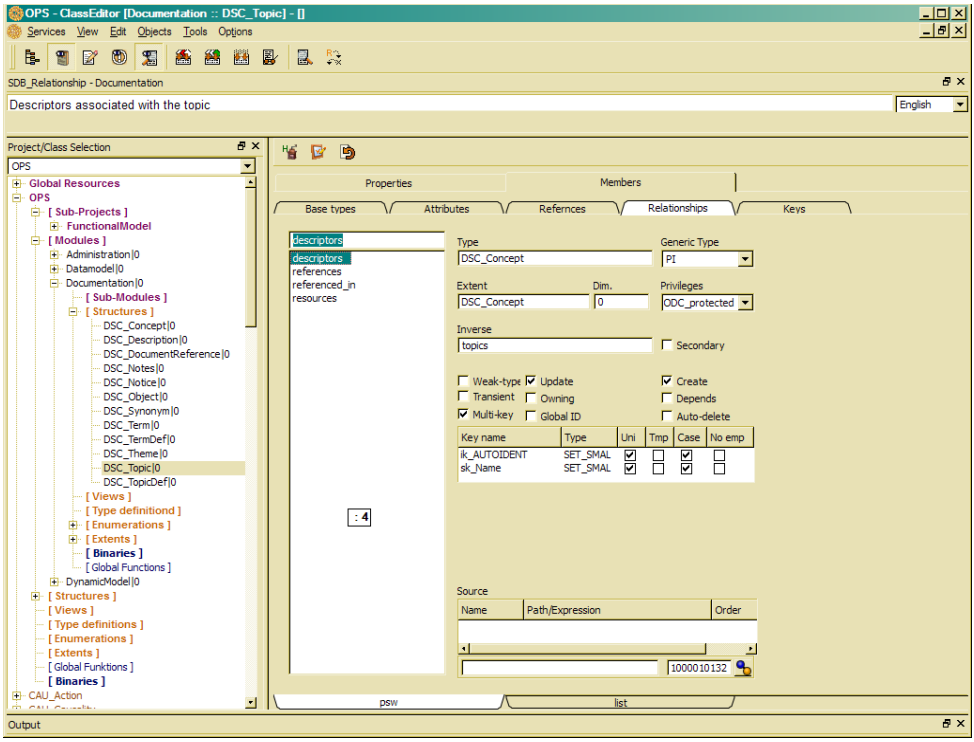
Die Entwicklung in der ODE beginnt mit der Dokumentation der Problemanalyse im **Terminus**. Hier kann man das Problem detailliert in einem Terminologiemodell (Domainmodell) beschreiben. Aus diesen Informationen können Dokumente, UML, HTML-Seiten oder andere Presentationsformen generiert werden.



Terminus befindet sich derzeit in der Entwicklung und wird 2011 bereitgestellt. Die bereitgestellte Implementierung von Terminus ist eine Skizze, der als Prototyp für Terminus angesehen werden kann.

Vom Terminologiemodell zum Datenmodell - Class Editor

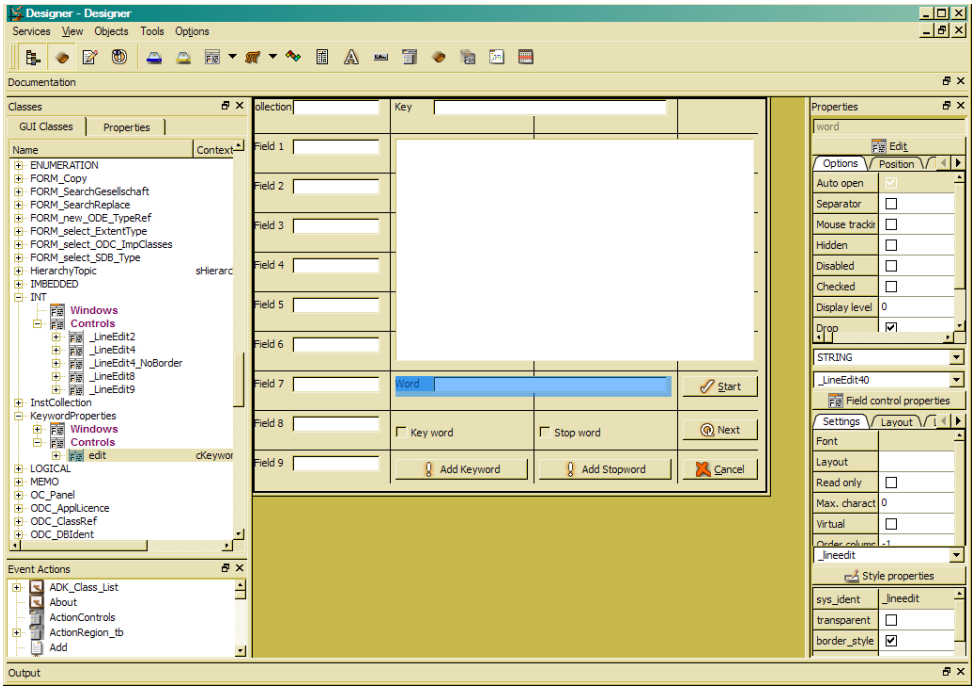
Die Terminologieobjekte werden dann im **Class Editor** zu Objekttypen des Objektmodells oder zu Klassen spezialisiert. Da 50-80% der Schema-Informationen bereits im Terminolgiemodell enthalten sind, wird die Erarbeitung des Objektmodells zum Kinderspiel.



Hier, wie in allen weiteren Phasen, ist die Dokumentation des gerade ausgewählten Objektes sofort verfügbar und kann eingesehen oder überarbeitet werden. Ist das Datenmodell fertiggestellt, wird es geprüft und in dem Produktionszustand überführt.

Applikationsdesign - Designer

Unmittelbar auf dem Datenmodel aufsetzend kann mit Hilfe des GUI-Designers das Applikationsdesign erfolgen. Dank der ADL Technologie [6] kann man in kürzester Zeit einen funktionsfähigen Prototyp bereitstellen.

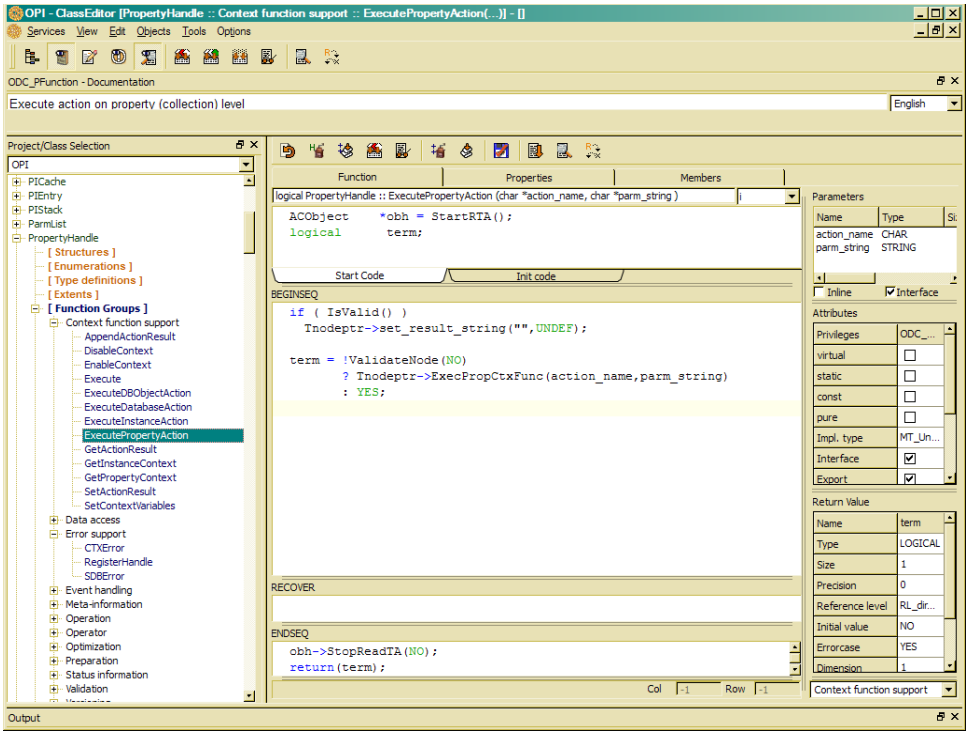


Für die Prototypentwicklung ist in den meisten Fällen keine Programmierung erforderlich. Alle Controls, sowie Regions oder Columns in Tables und Trees werden über Datenquellen (Property und Extent-Namen) mit der aktiven Datenbank verbunden. Dadurch kann eine Applikation nach dem Design sofort ausgeführt werden. ODABA^{NG} GUI-Elemente sind äußerst leistungsfähig, was den Aufwand für die Applikationsentwicklung drastisch reduziert.

Programmieren in der ODE - Class Editor / Designer

Um Businessregeln oder spezielles Verhalten der Anwendung zu implementieren, kommt man um die Programmierung nicht herum. Die Programmierung ein einer der unterstützten Programmiersprachen kann natürlich in jeder beliebigen Entwicklungsumgebung erfolgen.

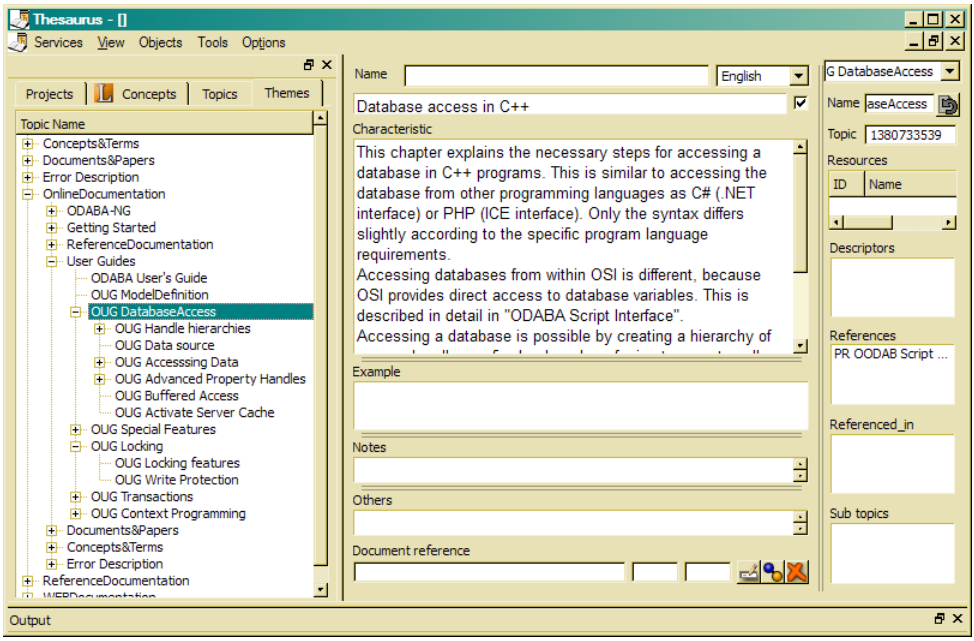
Bleibt man jedoch in der ODE, genießt man nicht nur erhebliche Vorteile durch die automatische Code-Generierung, sondern auch durch die begleitende Dokumentation.



Implementierung von Funktionen ist auf gleiche Weise auch im Designer möglich, der die Programmierung von Event-Handlern in Kontextklassen unterstützt.

Die Anwendungsdokumentation – Terminus

Im letzten Schritt wird die Dokumentation um fehlende Topics ergänzt, eine oder mehrere Dokumentationshierarchien kreiert und die Dokumentation als monolithisches Dokument oder als WEB-Seiten generiert.



Die Bearbeitung der Topics und Erstellung von Topic-Hierarchien erfolgt wieder in Terminus. Von hier aus erfolgt auch die Generierung verschiedenartigster Dokumente.

Kommandozeilen-Tools

Eine ganze Serie weiterer Werkzeuge wird für die Wartung der Datenbank und die Administration von Client/Server-Anwendungen als Konsole-Anwendung bereitgestellt.

6 Und was kostet das alles?

ODABA^{NG} ist eines der wenigen Systeme, die sowohl unter LINUX als auch unter Windows als Open Source-Produkte auf der Basis einer leicht modifizierten GPL (GNU Public License) benutzt werden können. Damit stehen die meisten Features und Tools allen Entwicklern zur Verfügung, die ebenfalls Open Source-Produkte entwickeln. Das Einzige Third Party Product, das in ODABA benutzt wird, ist QT, das ebenfalls auf der Basis einer GPL benutzt werden kann.

Nicht jede Anwendung soll aber als Open Source Anwendung entwickelt werden. Deshalb kann auch eine kommerzielle Nutzungslizenz erworben werde. Kommerzielle Lizenzen sind reine Entwicklungs-Lizenzen, die nur solange erforderlich sind, wie die Software entwickelt und gewartet wird.

Spezielle RUN-Time-Lizenzen sind nicht erforderlich, außer natürlich bei der Benutzung von Fremddatenbanken wie Oracle oder MS SQL Server, wenn diese genutzt werden.

Darüber hinaus kann man sich bei der ODABA Development Foundation registrieren lassen und mit einem selbst gewählten Jahresbeitrag die Weiterentwicklung fördern aber auch Einfluss auf die Entwicklungsrichtung von ODABA nehmen.

7 Mehr Dokumentation!

Auf unser Dokumentations-Download-Seite gibt es eine Vielzahl von Dokumenten, die in 6 Gruppen unterteilt wurden:

0 – Konzepte

1 – Technische Einzelheiten

2. – Installation und start

3 – Services und Tools

4 – ODABA Entwicklungsumgebung

P – Veröffentlichungen

Zusätzlich gibt es zwei Online-Dokumentationsseiten zu

Alles über die ODABA Datenbank
(http://www.run-software.com/content/documentation/odaba)

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

8 Downloads und Support

Eine Evaluierungsversion steht jederzeit auf unserer Downloadseite (http://www.run-software.com/content/downloads) zur Verfügung. Wir empfehlen unseren Nutzern jedoch, sich bei uns zu registrieren, um mit den aktuellen Neuheiten versorgt zu werden.

Wir bemühen uns, alle Anfragen im Open Source Bereich so schnell und umfassend, wie möglich zu beantworten. Wir stellen jedoch auch verbindliche Supportleistungen mit vereinbarten Reaktionszeiten im Rahmen von Supportverträgen zur Verfügung.

Schulungen zählen genauso zu den von uns angebotenen Leistungen, wie Havariesupport und anderes mehr.

9 Referenzen

[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