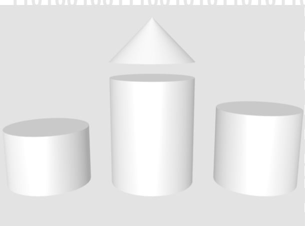


ODABA NG

Database categories and ODABA

Reinhard Karge

22nd May 12

run Software-Werkstatt GmbH
Weigandufer 45
12059 Berlin

www.run-software.com

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

Berlin, May 2012

Table of Contents

1Introduction..4

2Database categories - ODABA and the world of databases 6

2.1Database schema 8
2.1.1P0 database schema 10
2.1.2P1 database schema 11
2.1.3P2 database schema 12
2.1.4P3 database schema 13

2.2Database consistency and intelligence 15

2.3Database queries 17

2.4Implementation view 19

2.5Event handling 20

2.6Transactions 22

3References 23

1 Introduction

ODABA is a terminology-oriented database management system, which allows reflecting IT problems and solutions in terms of human language (terminology model). The theoretical base for ODABA is the “Unified Database Theory”, which classifies different database systems (Key/Value stores, Relational and OO-databases, data warehouse) by schema levels.

As P_3 database, ODABA not only supports OODBMS features, but also Key/Value Store technologies as well as data warehouse technologies (partially). Moreover, ODABA provides an OR-Mapper, that allows transforming ODABA data models into relational models (MySQL, Oracle, …) and also supports RDBMS as database mirror or primary data store.

With OSI, ODABA provides a JAVA like NoSQL scripting language for accessing and manipulating data. Instead of SQL statements, ODABA supports access by database variables and operation paths. A comprehensive C++ API also provides comfortable database access features. The ODABA object definition language (ODL) is an extension of the ODMG 2003 database standard for object-oriented databases. Beside enhanced database concepts, ODABA supports scalable client/server architectures (including NoServer applications), additional storage formats and many other extensions.

With Terminus, ClassEditor, GUI Designer and Object Commander, ODABA provides a series of rapid application development (RAD) tools, which support fast development for complex applications. ODABA and tools provides a suite for application developers in different areas.

ODABA is specialized in handling complex systems with many relationships between different objects. Thus, ODABA is a powerful system for developing business applications, but also simple applications for personal use (as the Media Player example).

ODABA allows managing database information much easier as comparable systems (no SQL required!). Moreover, ODABA technologies make application development simpler and faster than comparable tools of big database vendors.

Terminology based development allows defining customer's requests in terms of human language (terminology model). Terminology models may be transformed into database models, which might be used immediately for GUI application design. Database models may also be scripted using ODL or may be defined by means of the ClassEditor, which also provides wiz-

ards for inexperienced users. Business rules (stored procedures and event handler) may be implemented in OSI, but also in C++, as well as application rules. Based on Qt, GUI kit and Active Data Link technology, ODE allows designing database driven GUI applications.

ODABA also supports document generation features based on Open Document Standard. ODE tools support multilingual documentation objects, which may be composed to documents, HTML pages or online help topics. The complete WEB documentation (nearly 10 000 HTML pages) and many documents have been generated from ODE documentation topics.

ODABA has been used in different production environments since 1994. In August 2010, ODABA has been released as Open Source Software under GPL for Linux and Windows platforms. ODABA is maintained and released (4-6 releases per year) by run-Software and distributed via Source Forge and other Open Source platforms. There is not yet a developer community, but run-Software has moved to Büro 2.0 community and became member of OSB (Open Source Berlin, 2011) in order to improve interaction with the community. ODABA has been the base for several commercial projects as BRIDGE/METAS (knowledge base for statistical offices), BELAMI (accounting and contract management system for MITROPA/DSG) or KUVERT (Insurance management for an Internet Insurance agency). REFEUS (developed by the Refeus group) provides a knowledge collection system for students and scientists.

2 Database categories - ODABA and the world of databases

The paper tries to classify database management systems by different classifications in order to demonstrate the position of ODABA in the world of databases. A general way of classifying databases by degree of order has been introduced in [UDT]. In order to introduce the smallest data unit, atomic data items have been introduced in [UDT] as: : "An atomic state s describes the relation between an identifiable object (o) and a property (p) with a property value (v) at a given point in time (t), i.e. a data item or state consists of four components, where the value is a function of the three other components

```
s = (o,p,t,v) where v = S(o,p,t)
```

Simply said, it means, that a value for a state (fact) makes sense, only, when the object, it belongs to, the property, it describes and the time point of measure is known. 1,79 cm does not make sense, as long as one does not know, that this is my (object) height (property) today (time). The function S, which describes the relation between state identifying components and value is usually called Schema.

The state does not say anything about the complexity of a fact or state, i.e. the property might be complex (as address (p) consisting of country, city, zip code, street and house number) or may also define a collection of values (given names, but also children of a person are typical examples for collection properties. Since the value is a function of object, property and time, data can be arranged in terms of those components, i.e. a schema for storing data may be defined without knowing the data itself.

Thus, one important DBMS classification can be described as schema classification, which defines typical characteristics for the degree of ordering properties in different DBMS families. In [UDT], four database families have been introduced, which provide a complete classification for DBMS by schema characteristics:

- State model (P_0 model) - A simple schema typically used as base for key/value and document stores often called NoSQL databases (HBase, BigTable, MongoDB CoiuchDB and many others).
- Type model (P_1 model) - Arranging properties in data types, which is typically for relational databases, i.e. the entity-relationship model (MySQL, Oracle etc.)

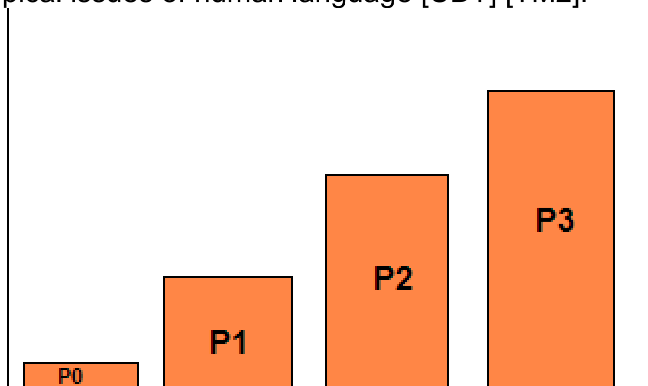
- Class model (P_2 model) - Arranging properties in types and objects and collections, which includes collection and relationship properties. The class model is the base for object-oriented databases according to [ODM] specifications, the object model (Versant, ObjectStore ...)
- Classification model (P_3 model) - Arranging properties in types, objects in collections and collections in collection hierarchies of any nesting level. The classification schema is the base for terminology models and implicitly plays a role for data warehouse systems. Because of its terminology orientation, these databases are called terminology-oriented DBMS (ODABA).

In the following sections we will consider DBMS of these four categories, its advantages and disadvantages and typical use cases. Common for all databases is the fact, that object and time dimension are often not explicitly defined in a database schema, i.e. object and time dimension are considered as open dimensions in most DBMS. Thus, schema definitions mainly describe the way, properties are arranged in a database.

2.1 Database schema

A database schema defines the way, data is arranged in the data store, i.e. is defines the schema function S. State identifying components are handled differently in different DBMS. In general, the object dimension is considered as open dimension, which simply defines a collection of data items belonging to one or more kind of individual objects. In order to locate a single individual object in the collection, usually one or more object identifiers are defined in the schema and access functions are provided, which allow accessing an object instance by means of those identifiers. The time component is ignored in most schema functions and the application cares about time in the one or the other way. Hence, the main aspect schema functions are focused on, is the way properties are arranged.

While P_0 DBMS require a minimal schema definition, P_3 databases require rather complex schema specifications. Hence, P_0 databases are very simple to create and are very flexible in use. Relational (P_1) DBMS order properties in type (table) definitions and allow defining relationships between tables in terms of link attributes. Object-oriented (P_2) DBMS support collection properties (relationships), i.e. a property may represent a collection of related objects. Terminology-oriented (P_3) DBMS support set relations (superset/subset relations) and hierarchical classifications, which result from typical issues of human language [UDT] [TM2].



The graphic above is not result of exact measures, but reflects the schema definition effort by practical experience and amount of standard definition elements of P_1, P_2 and P_3 schema definitions. It becomes obvious, that the simplest way to start is running a P_0 database, since this requires nearly no effort for preparing the database.

Theoretically, P_1 schemata include P_0 schemata, P_2 the P_1 schemata and P_3 the P_2 schemata. This is, however, not the case in practice. Thus, relational and object-oriented DBMS do not support ad-hoc attribute extensions for types and tables and are, hence, not as flexible as P_0 databases. On the other hand, object-oriented and relational DBMS schemata are equivalent and can be transformed into each other [ORM]. P_3 DBMS ODABA is an extension of the [ODM] standard for object-oriented database models. Moreover, by providing property extension features for data types, ODABA also includes P_0 DBMS features concerning flexibility.

2.1.1 P_0 database schema

Often, P_0 DBMS are divided into key/value stores and document databases. Key/value stores (e.g. HBase, BigTable) simply implement the schema function by mapping any kind of value to object identifiers (key), property name and time stamp. Here, property becomes an open schema dimension, too, and one might add any property or new version for a value without defining it in an explicit schema. The key/value store is the most consequent implementation of a schema function S, since it exactly does, what the schema function requires: assigning a value v to an (o,p,t) vector.

Document databases (e.g. MongoDB, CouchDB) usually store one entry per object, which contains all the properties belonging to the object. Number and type of properties are not limited, i.e. properties become an open dimension, too, just being ordered vertically. The way object properties are stored in the "document" depends on the document database. In general, any kind of semi-structured representation as JSON, XML, OIF, which refers to property/value pairs, might be used. Usually, these databases do not handle time dimension and consider data as timeless. Object (document) versions are possible by extending the key component for documents but property versions are usually not supported.

In any case, P_0 schema is the most flexible and most simple database definition. The price for simplicity is a lot of implementation work, when problems become more complex.

2.1.2 P_1 database schema

P_1 DBMS order properties in data types (or tables). Ideally, each object instance is stored in one table row containing all its relevant properties. Thus, a P_1 schema defines the set of all possible properties in the database in terms of table/attribute pairs. The a strict P_1 schema definition concerning properties will limit the use of P_1 databases, since one cannot expand properties at run-time. On the other hand, it provides a sort of trusted state, since one can rely on the definitions provided in the schema. Moreover, constraints might be defined in order to guaranty logical consistency for values and indexes based on properties (attributes) defined for the schema.

P_1 schemata are typically defined for relational DBMS (MySQL, Oracle, MS SQL, ...). In order to define relationships between object instances (entity/relationship schema), relational DBMS usually do one step toward to P_2 schemata by including kind of collection support into the schema definition. Relational DBMS are highly standardized and used since more than 50 years. Schema definitions as such are not very difficult, but usually, database schemata have to be normalized (i.e. no redundancy), which makes definitions more difficult. Also artificial tables for defining M:N relationships make definitions a bit more complicate.

The time component is not explicitly supported in relational DBMS and applications have to decide, where to store the time component. This also means, that data retrieval does not include the time component, as long as it is not part of the table instance (column) or property name.

Still, relational schemata are comparable simple to define. Because of high degree of standardization, one may easily transfer schema definitions between different kind of relational DBMS. Hence, relational databases are typically used for providing business applications or for storing statistical data. Because of ACID requirements, relational DBMS perform not very well in distributed systems, where P_0 databases often provide better solutions. Relational databases also cause problems, when data structure becomes very complex, since nested joins will reduce the performance extremely.

2.1.3 P_2 database schema

Similar to P_1 DBMS, P_2 DBMS (mainly referred to as object-oriented DBMS) arrange properties in types. Object-oriented DBMS are known since about 1990. In contrast to relational DBMS, object-oriented DBMS allow complex properties, i.e. types create a type of ontology. Conceptually, type hierarchies belong to the P_1 schema, but most relational DBMS do not support types as such. The extension of P_2 schemata compared with P_1 schemata is the support of collection properties (relationships between individual objects). Another schema extension for object-oriented DBMS is the support of inheritance relations, which, in fact, are just a special kind of relationships.

Thus, P_2 schemata are type and collection based, A special effect resulting from set relations (properties) in contrast to table relations are inverse relationships in order to reflect bi-directional links between object instances. Similar to relational DBMS, object-oriented DBMS do not support time component explicitly. One may, however, define time as part of the data type (attribute) or in the property name. Thus, open time component might be implemented on object instance level, but not on property level. This, however, is not a limitation of P_2 schemata but of implementation of many object-oriented DBMS.

P_2 schemata are closer to human language reflections and thus, easier to understand. Nevertheless, P_2 schemata include a lot of possible errors and most object-oriented DBMS do not support all requirements, which result from collection support. Thus, it depends on the database provider, how much support is given for schema definitions, and this is, in most cases, not very much, which makes schema definitions even more difficult to handle.

P_2 schema databases are a good mean for handling complex problems. They are faster accessing linked objects but are slower when accessing large amount of object instances. Many P_2 DBMS support online schema evolution, which makes changes in the data model easier. Thus, P_2 DBMS are predestinated for business applications, which require often extensions or changes when the business model changes.

2.1.4 P_3 database schema

The P_3 schema consequently extends the P_2 schema family by adding support for collection hierarchies. Collection hierarchies are supported in two ways: either by hierarchical classifications provided in the schema definition or by defining set relations in terms of subsets and supersets. Including classifications and set relations into the database schema is one more step closer to human language, because P_3 DBMS are called terminology-oriented DBMS.

One important P_3 DBMS currently known is ODABA, which became a P_3 database about 2000. It completely includes the ODMG schema definition requirements [ODM] not only conceptually, but also complies with the ODMG object definition language (ODL), so that it becomes easy to upgrade from P_2 to P_3 schema. Basically, ODABA is based on the terminology model 2 [TM2], which defines the essential database requirements resulting from human language model. The P_3 schema is limited to a predefined number of properties, too. In order to reach the flexibility of key/value and document stores, but also for meeting human language requirements, ODABA supports property extensions, i.e. one may add any kind of property to any object instance, i.e. object instances are extendable beyond its data type definition. ODABA also supports the time component in different ways from individual object version up-to consistent object space version. Thus, time becomes an open component similar to the object dimension. In addition, ODABA provides several schema extensions, which result from the terminology approach (instance ownership, instance dependency, delete empty, auto-deletion).

Terminology based P_3 DBMS are closely related to human language reflection, which makes schema definitions easier, on the one hand, but on the other - human language is not easy. ODABA provides GUI tools besides the ODL, which support terminology and database model transformation, defining schemata assisted by wizards and also include schema checking functions.

Besides the terminology-oriented database ODABA several OLAP tools (e.g. SuperStar) provide elements of the P_3 schema especially concerning the set hierarchy and aggregation aspect, which is not covered by ODABA. Those must be considered, however, as database tools rather than as independent database systems.

Similar to P_2 DBMS, P_3 DBMS are a good mean for solving complex problems. Moreover, the terminological approach supports user-friendly project

specifications. Set hierarchy support provides automatic maintenance of aggregates but is not yet supported by ODABA.

2.2 Database consistency and intelligence

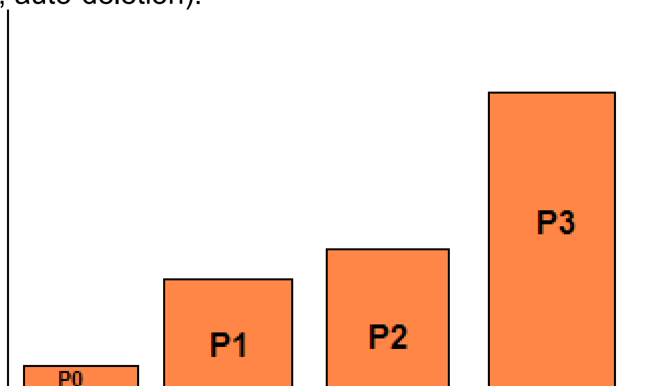
It is not a big help, when the schema is just more difficult to define, but nothing more. Hence, the more complex a schema definition is, the more consistency requirements have to be fulfilled. Here, we will talk about consistency rules resulting from schema definitions, but not consistency rules defined by the application (logical consistency). We also expect at any time, that the database system cares about its internal consistency rules.

For P_0 databases, there is not much to say about. As long as the object identifier (key) does not change, the only thing expected is that objects are accessible via key value (and property name, time).

With implementing P_1 databases, table relations and indexes create additional consistency rules. Changing an attribute value, which is part of a key, one expects, the the DBMS maintains the corresponding indexes.

Moreover, referential integrity is expected for P_2 databases (which also includes relational databases in this case), i.e. when deleting a row in a table, all referenced to it should be removed or it should not be deleted at all. This kind of schema consistency is guarantied by most relational and object-oriented DBMS. Object oriented databases also have to maintain the inverse relationship, when being defined, but this is not always the case.

P_3 databases have to guarantee the consistency of defined set relations. In case of ODABA, several schema extensions have been made, which result from the terminology approach (instance ownership, instance dependency, delete empty, auto-deletion).



The advantage with low consistency databases (P_0 schema) is, that they will do exactly, what has been implemented in the application. High consistency databases, however, do a lot of work, which is difficult to implement. But sometimes they seem to behave strange, because one is not always aware of all the (useful) rules that are executed in order to keep the database schema consistent. Schema consistency on a high level is a good thing, as long as one understands the intelligent behavior of the database. Hence, using a P_3 schema as supported by ODABA will reduce the implementation resources for an application extremely, when the problem is complex.

2.3 Database queries

Many implementation work is related to data access. Since the schema for a P_0 database does not contain much information, access is simple, as long as single object instances are involved. Selecting object instances by conditions (SELECT operation), becomes more difficult and joins will create a lot of work, when running a P_0 database. In order to run complex queries, P_1, P_2 or P_3 schemata are more efficient. By providing common query rules (SQL), relational DBMS (P_1 schema) allow specifying complex queries.

Query languages are typically used for accessing data in relational DBMS, but similar query languages are also supported in object-oriented and terminology-oriented DBMS. In addition, P_2 and P_3 schemata allow using traversal paths, which becomes possible because of collection properties. Thus, one might ask for the collection of all children of employees in a company like

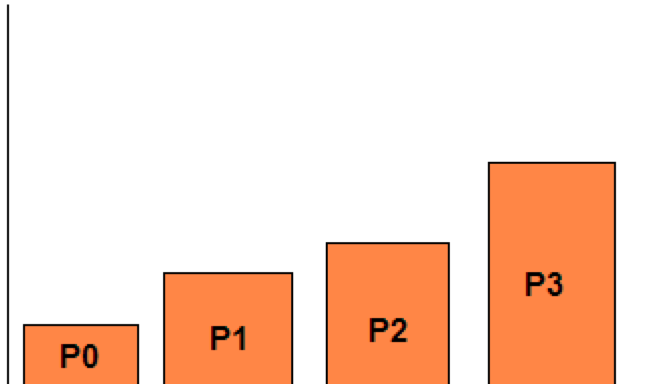
```
Company('run-software').employees().children()
```

This could also be expressed in a rather complicate SQL query, since a property employees could not be defined in a P_1 schema. One more feature becomes possible because of the functional model, which is typically part of P_2 and P_3 extended schemata. The functional model implements behavior (rules) on type level, which might be included in an operation path:

```
Company('run-software').employees().ChildrenIncome()
```

Here, ChildrenIncome might be a complicate algorithm implemented in a function for data type Person. Since elements of SQL queries (SELECT, FROM, WHERE etc) may also be implemented as generic functions, operation paths allow expressing each kind of SQL or OQL query, but are more flexible, since they might be mixed with other functions and the sequence of operations is not fixed. Finally, operation paths are shorter and much closer to human language and thus, easier to handle. On the other hand, they may contain hidden operations, since referring to object type functions, it is not always obvious, what really happens like in an SQL statement.

In principal, traversal and operation paths are based on P_2 schemata and could be supported by any object-oriented DBMS (which is practical not the case). Terminology-oriented DBMS require support of operation paths because of its human language orientation. The graph illustrates query language features. It does not consider operation paths for P_2 schemata.



Although the P_3 schema provides maximum support for efficient queries, all the extended query features as provided by ODABA could also be provided by any object oriented P_2 DBMS or by object relational DBMS, which provide an object-oriented view to relational databases. For complex problems, this will reduce the amount of implementation effort extremely. On the other hand, when handling large amount of simple structured data, P_0 queries will perform better in terms of time and implementation.

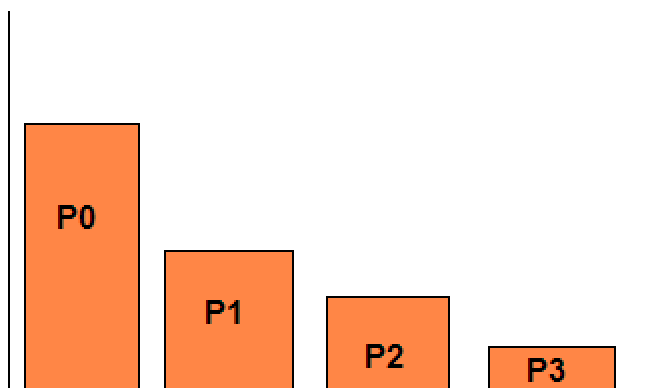
In general, simple queries are more efficiently executed in P_0 or P_1 databases. Complex queries, however, are served much better by P_2 or P_3 DBMS.

2.4 Implementation view

Creating a database is usually a means to an end, and not just fun. The end, usually requires implementation in order to provide the required results. One, but not the only way is dividing implementation into 3 layers:

- Database access layer
- Business layer
- Application layer

Since application and business layers do not depend on the database schema family, only database access rules are considered here. Because the missing schema information, the database layer for P_0 databases becomes very expensive (which is not so important, as long as the data structure is simple). P_1 (relational) databases also require remarkable effort, since row selections and joins have to be defined in order to gather the required information. Since P_2 databases support collections, many queries on the database layer become obsolete, but database access layer operations are still required when it comes to aggregation. This is also not necessary in many cases, when running a P_3 database, which supports set hierarchies, so that P_3 databases rarely require operations on database access layer.



Avoiding the database access layer nearly completely when running P_2 or P_3 databases will reduce the implementation effort, again. Mainly the collection support in P_2 and P_3 DBMS will reduces the amount of necessary rules on the database access layer by 80-90% in many applications. Further reduction results from set hierarchy support provided by P_3 DBMS.

2.5 Event handling

Event driven applications have been very successful concerning the GUI part. But also database applications can be implemented much better, when being based on event mechanisms (active databases). For P_0 databases, events are not of interest, since the database layer is implemented explicitly. P_1, P_2 and P_3 databases, however, rely on generic database access rules implemented by the DBMS. At least in order to react on specific events as automatic row deletions, event handling needs to be supported.

In order to implement business logic independent on application logic, event handling is also a good mean. For the business logic, it is not so important, who deleted an object instance and why, but the fact, that is has been deleted. Thus, handling such kind of events is rather the task of event handlers (triggers), which are always called when the database detects such an event.

Different events on object instance (row) level are triggered by relational DBMS (P_1). P_2 databases also need to trigger collection events in order to signal inserting or removing instances from a collection. Unfortunately, only some object-oriented databases support triggers at all. In order to support an active database, triggers for event handling are required on all data levels, which includes not only object instances (rows), but also collections and attributes (properties).

Event handling mainly depends on the schema concepts supported by the database. Thus, P_1, P_2 and P_3 databases could support attribute and instance events. P_2 and P_3 databases could support collection events and P_3 databases could support collection hierarchy events. Practically, most relational databases support instance event handling and database event handling. Just a few object-oriented databases provide limited support for instance and property events. ODABA as P_3 database does not support collection hierarchy events but provides comprehensive support for database, object instance and property (attribute and collection) events. In order to fully comply active database requirements, ODABA also supports events signaling internal state transitions as reading, selecting or unselecting an instance in a collection.

Implementing the business logic completely based on database event handlers requires triggers at least on instance and collection level. For providing an active database, that is able to control, e.g. a GUI framework, property and internal events have to be signaled, too. ODABA provided a active data link technology [ADT], which translates database events into

GUI vents, which allows running complex GUI applications completely based on a generic GUI framework.

2.6 Transactions

Since P_0 systems do not have consistency requirements, there is no need for transaction support, but it might be helpful in order to support logical consistency. Hence, many P_0 DBMS guarantee consistency on a rather low level, while others fulfill ACID (atomicity, consistency, isolation, durability) requirements. Most P_1 DBMS are ACID compliant and use pessimistic or optimistic locking on instance level.

When a P_2 or P_3 DBMS guarantees schema consistency simple operations like adding or deleting an object instance may result into a number of additional operations caused by e.g. relationship or set consistency rules. Since each of those operations may call other operations again by event handler calls, the number of operations caused by a simple function call might even grow. This will increase the risk of conflicts and dead locks, which is not a problem of complex schemata but a problem of complex tasks. Hence, P_2 and P_3 DBMS run each instance modification within an implicit transaction, which fails, when any of the involved operations fails. Locking within implicit transactions might be optimistic or pessimistic. Because of the collection support, P_2 (and P_3) DBMS have to support collection locks, which is usually not the case.

When the success of a transaction depends from one or more instances or collection, which are not updated, but read only, those have to be locked explicitly by the application in order to guarantee consistency. This is not the case for schema consistency but may result from constraints and other logical consistency requirements or event handlers.

3 References

[UDT] Karge R.: *Unified Database Theory*, RUN Software, Orlando (Florida), 2003, www.run-software.com/content/downloads/documentation/P1_UnifiedDatabaseTheory.pdf

[TM2] Karge, R.: *Terminology Model II*, Berlin, 2011 www.run-software.com/content/downloads/documentation/P2_TerminologyModel_v2.pdf

[ORM] RUN Software: *Multiple database storage support*, Berlin, 2007 www.run-software.com/content/downloads/documentation/1.8_ORMappingConcept.pdf

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

[ADL] RUN Software: *Active Data Link (ADL)*, Berlin, 2007 www.run-software.com/content/downloads/documentation/1.7_ActiveDataLink.pdf