IDEA GROUP PUBLISHING

701 E. Chocolate Avenue, Suite 200, Hershey PA 17033, USA
Tel: 717/533-8845; Fax 717/533-8661; URL-http://www.idea-group.com

ITP4469

Data Warehouse Schemas: A Software Engineering Approach towards an Efficient Modeling of Complex, Dynamic Conceptual Schemas

Reinhard Jung and Robert Winter
University of St. Gallen, Mueller-Friedberg-Strasse 8, CH-9000 St. Gallen, Switzerland
Tel: +41 71 224 2934; Fax: +41 71 224 2189; {reinhard.jung | robert.winter}@unisg.ch

ABSTRACT

In contrast to most traditional information systems which are based on a static, consistent view of transactional data, a data warehouse comprises several stages of data integration and data aggregation. Hence, the conceptual design of data warehouses addresses not only data structures, but also derivation paths. Integration and aggregation paths result in highly complex schemas and consistent time references must be introduced. Moreover, available meta data often cannot be reused for data warehouse design because warehouse development tools focus on physical data load instead of conceptual integration. In this paper we enrich the Structured Entity-Relationship Model with appropriate extensions for data warehousing design (e.g. derivation rules and schema aggregation) and adapt a commercial CASE toolset to support such extended conceptual modeling. By that means warehouse schemas can be specified efficiently and meta data of existing information systems can be reused.

1 INTRODUCTION

Regardless of the underlying paradigm, today's traditional information systems are developed based on conceptual considerations, i.e. they are designed according to the requirements at first. In contrast to that, data warehouses are usually built upon existing operational systems. Different views on corporate data have to be integrated into the data warehouse which is subsequently subdivided into data marts when appropriate. The foundation on existing systems and their data does not imply the design phase may be omitted [7]. Instead, existing data elements must be integrated (e.g. naming and structural conflicts have to be resolved) and mapped into a common schema.

Due to its purpose, scope and foundation, a conceptual data warehouse schema differs from schemas of conventional information systems:

- The schema is likely to become unusually large and therefore complex [9][11].
- Data aggregation and integration is normally implemented within the warehouse database. Hence, data derivation paths have to be included in its conceptual schema. The precise modeling of redundancy, however, was never required in traditional database design and creates some unsolved problems.
- Not only integration, but also consistent time references are required to transform transaction data into management information. Hence, time has to be dealt with on a conceptual basis.

Based on a short recapitulation of the state of the art in data warehouse / data mart design (section 2), deficits of current practice are identified in section 3. In section 4 schema abstraction levels, derivation rules and time semantics are proposed to overcome the shortcomings of traditional Entity-Relationship (ER) modeling for conceptual data warehouse design. Since additional design concepts cannot be utilized efficiently until a tight integration with CASE environments and corporate meta data is achieved, a prototypical implementation using Oracle's toolset Designer/2000 is summarized in section 5. The paper closes with conclusions and directions for further research (section 6).

2 STATE OF THE ART IN DATA WAREHOUSE / DATA MART DESIGN

There is a large number of commercial data warehousing tools available and it is almost impossible to keep up with the dynamic growth of this market segment. A rather comprehensive tool overview can be found on Datamation's web site (www.datamation.com). For our purposes it is sufficient to apply a simple classification taken from the same source. Data warehousing tools can be categorized as follows:

1. Warehouse construction components (data extract, cleaning, and load).
2. Warehouse operation components (data storage and warehouse management).
3. Warehouse access components.

Tools belonging to the first category are designed to extract the data from operational environments, including tasks like cleaning, or „scrubbing", to establish a data warehouse or data marts. The second category comprises tools for warehouse operation, i.e. specialized database management systems and similar systems. The third category includes end-user tools providing warehouse access to managers (e.g. OLAP tools) and tools for data download into spreadsheet software.

The main problem about the classes of tools as described above is the fact that no real integration is achieved between the different components. The only commonality between the tools is their orientation towards the implementation level:

- a construction component creates the warehouse database.
- an operation component runs it.
- one or more access components are finally used to extract data for decision support.

Although some tools make use of a repository [2], sharing design meta data is far from being a common practice.

A comprehensive approach for data warehouse design has yet to be developed. This state of affairs might be attributed to the different interpretations of the term „conceptual model". On the one hand, during warehouse construction, conceptual models simi-

This paper appears in *Challenges of Information Technology Management in the 21st Century,* the proceedings of the Information Resources Management Association International Conference.

lar to those deployed in conventional systems design are used. On the other hand, the basis of warehouse access are conceptual models focusing on multidimensional aspects, e.g. the hypercube model. The challenge in warehouse design seems to be the integration of these different conceptual models. Glofarelli et al. propose an interesting approach called Dimensional Fact model which transforms pre-existing ER schemas semi-automatically [6]. However, by simply transforming schemas, their interdependencies (e.g. the data flow within the warehouse or towards the data marts) remain being implicit.

3 WHERE TRADITIONAL CONCEPTUAL MODELING FAILS

Conceptual database design methods aim at creating schemas which are later used to generate databases for conventional information systems. On the one hand, such transactional systems mostly have comparably small schemas because of their limited scope. On the other hand, the schemas comprise mostly transactional data for day-to-day operations. For such „static" data, a current state and maybe a few obsolete states are stored, while its derivation, change history and utilization remain implicit and is usually hidden in application code. Finally, although time references are commonly used, time semantics are not addressed explicitly.

Hence, the features of conceptual database design methods are in some ways restricted to transactional systems [6]. Management information systems or, more generally speaking, decision support systems require a completely different type of database. The schema of a data warehouse has to provide an integrated data view of a broad range of operational systems [7]. Furthermore, it has to deal with dynamic aspects of data, e.g. data derivation for integration or aggregation purposes.

Therefore, a suitable design approach for data warehouses has to deal with the following characteristics:

1. ***Very large schemas***: Due to the broad scope of data warehouses as regards enterprise data, the resulting schema is likely to become very large and, therefore, very complex. This is also true if the data warehouse is physically subdivided into several data marts because an integrated view is nevertheless necessary to depict interdependencies between the marts.

The design approach of choice should provide mechanisms to navigate through the schema without loosing the overall picture and should support abstraction levels in schema design. Required features are:

- *Arrangement Rules*: Consistent rules are needed to increase readability and to arrange the schema's components according to the actual design task [4].
- *Clustering/Disaggregation Capabilities*: An efficient handling of the schema is largely dependent on suitable capabilities for clustering and aggregation/disaggregation. The latter is especially important if the overall schema comprises several data mart subschemas.

2. ***Redundancy***: On the physical level, according to the initial idea of data warehousing, the database comprises both elementary data (e.g. amount of invoice) and integrated and/or aggregate data (e.g. daily turnover). Hence, redundancy is introduced. Since the data warehouse designer needs to control this redundancy, it would be appropriate to provide conceptual information about redundancy and the underlying data flows.

Traditionally, conceptual data models (e.g. the ER model) deal with aggregation and dependencies on the entity type level, but do not usually cover derived attributes because redundancy is avoided wherever possible. In data warehouse databases, however, derivation rules and the resulting derivation paths represent important conceptual details of the schema and have, therefore, to be taken into consideration during the design phase.

3. ***Temporal aspects*** of data objects, i.e. their relation to time periods or to certain points in time, must be made explicit to allow for their correct integration, aggregation and propagation.

Warehouse data is typically structured along multiple dimensions, e.g., the data element *turnover* can be structured by the dimensions *product*, *region* and *dealer*. Although *time* is only one of many possible dimensions, it is most likely the one occurring in every warehouse data object. A very critical issue in this context is that data from operational systems has typically no explicit relation to time but almost always an implicit one becoming important during warehouse design. There are two different ways in which data can be related to time:

- *Reference*: The data is related to a specific point or period of time, e.g. a monthly turnover references the turnover summed up from the first to the last day of the month.
- *Validity*: The data reflects the state in a point or during a period of time (e.g. a monthly value measured on the 15th day of the same month).

In contrast to other modeling tasks (e.g. aggregation/disaggregation of entity types), the ER model does not provide suitable constructs for such semantics.

4 CONCEPTS AND METHODS FOR CONCEPTUAL DESIGN OF DATA WAREHOUSES / DATA MARTS

In this section we present three general concepts contributing to overcome the shortcomings of traditional ER modeling for conceptual data warehouse design: Schema abstraction levels are needed to preserve the ability of developers and key users to handle very large schemata usually found in data warehousing projects. Derivation rules are needed for the conceptual representation of derivation paths (e.g. for data integration and aggregation). Finally, explicit time semantics have to be introduced into conceptual modeling to allow for longitudinal analyses and integration of heterogeneous transaction data. The information model extensions resulting from these three concepts are presented in subsection 4.4.

All concepts are based on the Structured ER (SER) model initially proposed by Sinz [17] and used (e.g. by SAP AG) for the conceptual design of its R/3 databases [14]. Basically, the SER model creates an object type tree based on existential dependencies instead of an ER type network. Being a foundation for dynamic integrity control, existential dependencies have been continuously discussed in conceptual modeling. But in contrast to early proposals to represent existential dependencies as properties of „weak" object types [3], it was shown in [16] and [21] that they should rather be represented by a special class of (directed) relationship types. A relationship type representing an existential dependency is designated as a reference type. When reference types are used for conceptual modeling, the distinction between entity types and relationship types becomes obsolete. Instead, it can be differentiated between dependent (i.e. referencing) object types and independent (i.e. non-referencing) object types. All relationship types of the ER model are dependent object types. Since subtype, aggregate and associative instances reference other instances, these types are also dependent object types.

Since references are directed, a SER schema comprising only object types and reference types can be processed as a directed graph. In particular, such a schema can be arranged graphically following some simple arrangement rules, and reference cycles can be detected easily. Hence, in particular for complex schemas, the tree structure allows for structured navigation and simplified

and/or better quality control.

4.1 Schema abstraction levels

As mentioned above, ER schemas and their textual documentation lose their capability as an efficient communications platform for developers and key users with increasing complexity. Several techniques have been proposed to preserve a schema's usability for these purposes:

- Decomposition into subschemas using functional or process-based decomposition (e.g. „Production Planning", „Sales and Distribution", „Finance", „Asset Management", „Controlling", „Materials Management", „Human Resources", etc. in SAP's R/3 system [15])
- Arrangement of schema elements based on existential dependencies to allow for a "structured", specific interpretation [17]
- Schema condensation by means of a complex graphical notation [14]

However, even subschemas of SAP's R/3 system still comprise 300+ object types and 650+ relationship types, so even a scaled down graphical representation requires about 35 letter size pages to be printed and the textual documentation becomes a book with 448 pages. Obviously, schema decomposition, arrangement rules and a complex graphical notation cannot guarantee the usability of the SER model for the development of very large schemas.

Hence, SER schemas have to be clustered. But not every clustering operation results in a consistent clustered schema: The advantages of clustered SER schemata (with regard to visualization and documentation) coincide with a more or less significant loss of information that may even lead to inconsistent models. Based on earlier work on ER schema clustering [5][8][12][19][20] and SER schema clustering [10], a set of rules is proposed in [25] preserving certain formal consistency properties:

- Object type aggregation and object type selection may both be used for schema clustering.
- Object type aggregation must not create new object types.
- Object type aggregation must be based on references.
- Schema clustering cannot be performed algorithmically.
- An object type may be aggregated into more than one cluster.
- Reference clustering rules depend on object type clustering.

Following these rules, consistent clustering operations can be distinguished from inconsistent ones, so quality properties of the detailed schema are preserved in the clustered schema. By applying these rules to SAP's R/3 PP and SD subschemas, the number of schema elements was reduced by 75% [25].

Schema clustering rules cannot only be used for schema simplification, but also in an inverse way for schema refinement. In contrast to schema clustering, no semantic information is lost in schema refinement. Instead, semantic information must be added. Although schema refinement by applying inverse clustering rules was first analyzed in [24], more research is needed to fully understand the schema refinement process.

4.2 Derivation rules

To capture additional semantics in conceptual modeling, not only abstraction dependencies and existential dependencies, but also derivation rules can be introduced into conceptual modeling [18][22]. Derivation rules are normally ignored in conceptual design because, for a long time, commercial Database Management Systems (DBMSs) have been unable to guarantee the consistency of derived data [1]. With the introduction of database triggers and stored procedures, however, redundancy control can be guaranteed efficiently. Hence, there are no reasons to avoid data derivation in data warehouse databases.

Although some basic types of derivation can be differentiated that normally depend upon some particular type of abstraction dependency, the taxonomy of derivation rules is far from being as complete as the taxonomy of abstraction dependencies. As a consequence, derivation rules cannot be formalized to such an extent, and respective invariant properties remain quite abstract (see [23] for a detailed analysis). A graphical notation for six general classes of derivation rules (with examples) is illustrated in Figure 1.

Except for functional derivations, derivation relationships can be formalized as a relation between [22]:

- the target object type.
- a set of target attributes.
- a set of object types triggering a regeneration.
- a derivation rule.
- a set of source object types.
- a set of source attributes.

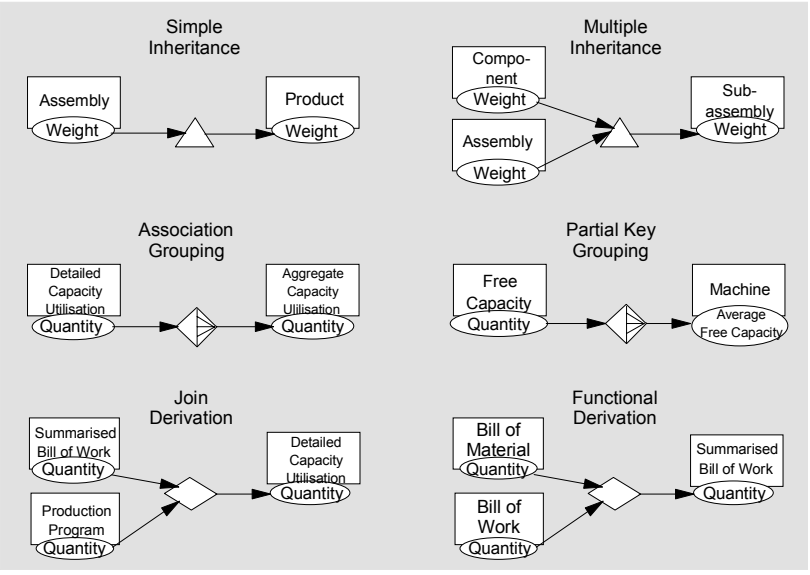
Although derivation rules may be formalized using the tuple relational calculus [13], we prefer a structured, SQL-like formalization to allow a straightforward generation of database triggers (see section 5). In contrast to other SQL-oriented rule declarations (e.g. [18]), we do not specify derivation rules as a SQL-database object. Instead, rules are only represented in the conceptual model explicitly. In the schema's implementation, they are distributed over various SQL database trigger declarations by a trigger generator.

4.3 Time semantics

Transaction data time references are explicitly related to a specific point or period of time. For example, each production program entry represents the total amount of planned production for some product from the first to the last day of some month. But implicitly, each production program entry has another time reference: Since plans are changed frequently, the amount is valid during a certain period of time only.

To specify data loads into the data warehouse in a general

Figure 1. Classes of derivation rules [23]



form, it is necessary to identify a taxonomy of explicit and implicit time references. If this information is included in the conceptual model (e.g. by using ”cumulative”, ”overwriting” and ”archiving” relationships between object types), it will be possible to automatically generate load procedures.

Data warehouse input data can be differentiated into net-change transaction data and regenerative ”snapshot” data. Both can explicitly and/or implicitly reference a point in time or a time period. If net-change data have no explicit time reference the data warehouse data are simply accumulated by loaded data. If input data have an explicit time reference, it has to be checked whether the loading cycle and the time references are identical. If this is true, updates have to be identified by a comparison of old and new data. Otherwise, warehouse data is replaced by the ”newer” input data. If, on the other hand, snapshot data are loaded into a warehouse data structure having no explicit time reference, it can be replaced by the ”newer” data. If it has explicit time reference, it has to be checked whether time horizon of existing and time horizon of loaded data are identical. If this is true, a new implicit time reference is generated and input data are stored in addition to existing warehouse data. Otherwise, the warehouse data are accumulated by loaded data.

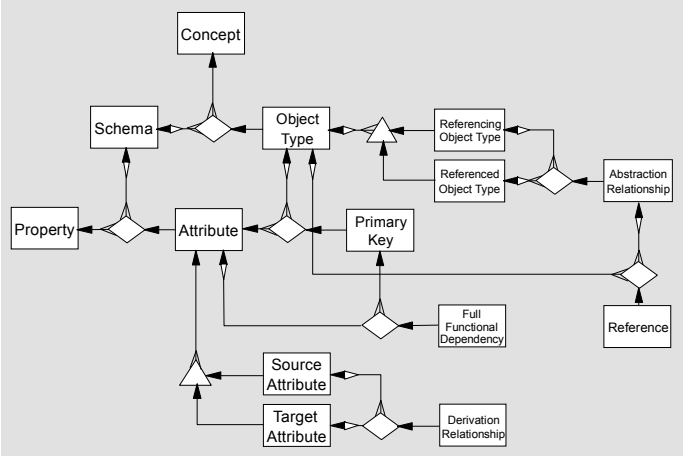
Thus, the way of loading data into the data warehouse can be derived from properties of data that should be specified during conceptual modeling. Of course, our taxonomy is far from being applicable to all kinds of data warehouses and input data. We are, however, confident additional research will identify extended taxonomies general enough that respective schema specifications can be processed by generators to derive load procedures automatically.

4.4 Extended information model

Productivity enhancements in data warehouse development require the various design, specification and generation tools to be based on a common repository. Ideally, such a repository is even shared with traditional (code-based or 4GL) application development tools. Application development repositories implement generalized information models of the supported tools. If a sufficiently elaborated base information model is available, the identification of extensions is quite straightforward because all concepts proposed in this paper can be specified by means of:

- additional associations between existing information objects (e.g. clustering relationships as associations between entities) and/or
- additional attributes for existing information objects (e.g. derivation rules as additional attributes of attributes, explicit/implicit time references, time horizons, and loading cycles as additional attributes of entities).

Figure 2. Conceptual repository schema [23]



As an example of information model enhancements, extensions covering general abstraction dependencies and derivation rules are illustrated in Figure 2. When linked to a schema, concepts become object types and properties become attributes. For every object type, certain attributes build the primary key. To prevent update anomalies, full functional dependency must be given between the primary key and all other attributes. Like abstraction dependencies link referencing object types to referenced object types, derivation rules link source attributes to target attributes.

5 DEVELOPING DATA WAREHOUSES USING ORACLE'S DESIGNER/2000

Modern CASE toolsets combine various analysis, design, implementation and test tools so the entire application life cycle is covered. Phase transitions tools (generators, reverse engineering tools), common reporting, project planning and quality control tools as well as, most important of all, a common repository for all systems views and development tasks complement isolated tools into an integrated application development environment.

The tight connection between data warehouse data and operative Information System (IS) data should be reflected by a tight integration of traditional application development (in particular data modeling) with Data Warehousing development. Hence, the proposed concepts for enhanced conceptual modeling should be integrated with a commercial CASE environment used for traditional IS development. To prove the feasibility of such an integration, we choose Oracle's Designer/2000, one of the leading products for client/server application development. Designer/2000's repository was extended to store additional associations and attributes, and an additional generator was developed to process the additional conceptual information and transform it into appropriate database objects. The generator is not presented in this paper.

Designer/2000's repository is extensible both with regard to additional associations between its 60+ information object classes and with regard to additional attributes of its information object classes.

Schema clustering creates m:n aggregation relationships between entity types and is, therefore, represented as an additional association between two ”entity” instances in the repository. The definition of such a repository extension is illustrated by Figure 3. Since derivation rules link one target attribute to one or more source attributes, they may be represented as additional attributes of an ”attribute” instance instead of creating an additional association between ”attribute” instances. The former case is illustrated by a screenshot of Designer/2000's repository administration utility in Figure 4.

It is, of course, not sufficient just to extend the repository. Additional information objects and attributes must be filled by specifications during conceptual design. Since Designer/2000's graphical design tools are not directly extensible, additional conceptual information can only be recorded by using form-based repository management tools. Although this method forces developers to create specifications with two different tools (graphical versus form-based), it is easy because additional associations and attributes of the repository are automatically covered by the interactive repository management forms. Figure 5 illustrates the specification of a derivation rule by using the form-based interface. By multiplying the values of the „quantity“ attributes of referenced „production program“ and „bill of work“ instances, respec-

Figure 3. Additional association "Entity Aggregation" in Designer/2000's repository

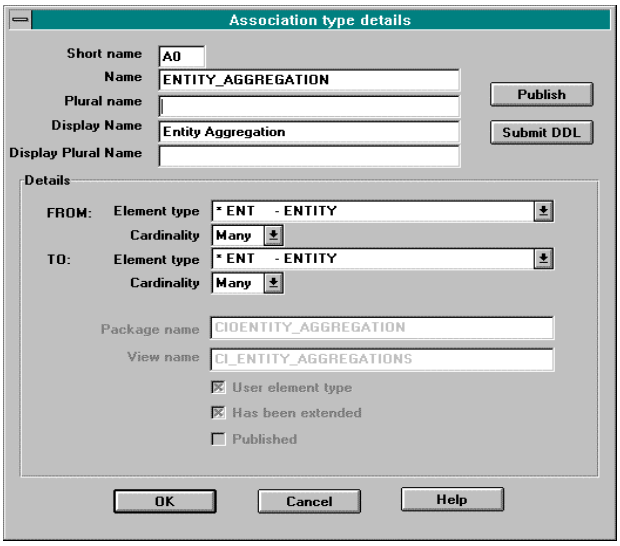
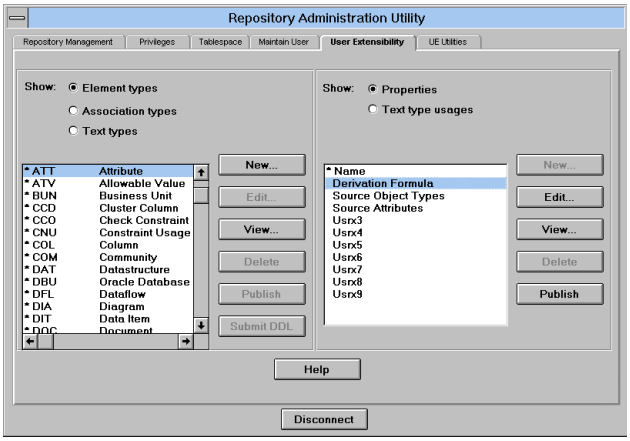


Figure 4. Additional attribute "Derivation Formula" in Designer/2000's repository



tively, the value of „quantity" attributes of „detailed capacity utilization" instances is derived.

6 CONCLUSION AND FURTHER RESEARCH

Data Warehousing is a new paradigm as regards management support that will become even more important in the near future. However, it has got very specific characteristics and problems that have to be dealt with. Although traditional concepts and techniques for systems development are reusable, some extensions and modifications have to be done, especially as far as controlled redundancy is concerned. The concepts presented building blocks for a comprehensive data warehousing design methodology. We have shown solutions for the generation of consistency preserving triggers and have demonstrated the extension of a CASE toolset to become more useful for warehouse development. Further research is especially neccessary as regards both the taxonomy of derivation rules and the time references as a basis for corresponding generation rules. Another interesting research topic is guidelines for the design of multidimensional databases.

REFERENCES

1. Adiba M., Derived Relations: A Unified Mechanism for Views, Snapshots and Distributed Data, in: *Proceedings of the Seventh International Conference on Very Large Data Bases*, Cannes, September 1981, pp. 293-305.
2. Chaudhuri, S, Dayal, U., An Overview of Data Warehousing and OLAP Technology, *ACM SIGMOD Record* 26(1), March 1997, pp. 65-74.
3. Chen, P.P., „The Entity-Relationship Model - Towards a Unified View of Data", *ACM Transactions on Database Systems*, 1 (1976), 1, pp. 9-36.
4. Eicker, S., Jung, R., Nietsch, M., Winter, R., Development of Data Warehouses for Production Controlling Systems, in: Kocaoglu, D.F., Anderson, T.R. (eds.): *Proc. of PICMET '97 - Innovation in Technology Management - The key to Global Leadership*, Portland State University 1997, pp. 725-728.
5. Feldman, P., D. Miller, Entity Model Clustering: Structuring a Data Model by Abstraction. *The Computer Journal*, 29 (1986), 4, pp. 348-360.
6. Golfarelli, M., Maio, D., Rizzi, S., Conceptual Design of Data Warehouses from E/R Schemes, in: El-Rewini, H. (ed.): *Proc. of the Hawaii International Conference on System Sciences 1998*, pp. 334-343.
7. Inmon, W.H., *Building the Data Warehouse*, 2nd edition, Wiley: New York et al. 1996.
8. Jaeschke, P., A. Oberweis, W. Stucky, Extending ER Model Clustering by Relationship Clustering, in: Elmasri, R. et al. (Eds): *Entity Relationship Approach - ER'93*, Springer, Berlin, 1994, pp. 451-462.
9. Meredith, M.E, Khader, A.: Designing Large Warehouses, in: *Database Programming & Design* , vol 9, no. 6, 1996, pp. 25-30.
10. Mistelbauer, H., Concentration of Data Models - From Project Data Models to an Enterprise-Wide Data Architecture, *Wirtschaftsinformatik*, 33 (1991), 4, pp. 289-299 (In German).
11. Poe, V., Clear, careful, and realistic: Guidelines for Warehouse Development; *Database Programming & Design*, vol. 7, no. 9, pp. 60-64.
12. Rauh, O., E. Stickel, Entity Tree Clustering - A Method for Simplifying ER Design, in: Pernul, G. / A.M. Tjoa (Eds.): *Entity Relationship Approach - ER'92*, Springer, Berlin, 1992, pp. 62-78.
13. Rauh, O., E. Stickel, Entity-Relationship Modeling of Information Systems with Deductive Capabilities, Research Report, Europa-Universität Viadrina, Frankfurt/Oder 1994.
14. SAP AG (Ed.), *SAP Information Model*, Walldorf/Baden 1994.
15. SAP AG (Ed.), *System R/3*, General Information, Walldorf/Baden 1994.
16. Scheuermann, P., G. Schiffner, H. Weber, „Abstraction Capabilities and Invariant Properties Modeling within the Entity-Relationship-Approach", in: Chen, P.P.S. (Ed.): *Entity-Relationship Approach to Systems Analysis and Design*, North-Holland 1980, pp. 121-140.
17. Sinz, E. J., The Structured Entity-Relationship Model (SER-Model), *Angewandte Informatik* 30 (1988), 5, 191-202 (in German).
18. Tanaka, A.K., S.B. Navathe, S. Chakravarthy, K. Karlapalem, ER-R - An Enhanced ER Model with Situation-Action Rules to Capture Application Semantics, *Proc. of the 10th Int. Conference on the Entity-Relationship Approach*, San Mateo 1991, pp. 59-75.
19. Teorey, T.J., G. Wei, D.L. Bolton, J.A. Koenig, ER Model Clustering as an Aid for User Communication and Documentation in Database Design, *Communications of the ACM*, 32 (1989), 8, pp. 975-987.
20. Vermeir, D., Semantic Hierarchies and Abstractions in Conceptual Schemata, *Information Systems*, 8/2 (1983), pp. 117-124.
21. Webre, N.W., „An Extended Entity-Relationship Model and Its Use on a Defense Project", in: Chen, P.P. (Ed.), *Entity-Relationship Approach to Information Modeling and Analysis*, North-Holland, Amsterdam, 1983, pp. 173-193.
22. Winter, R., Design and Implementation of Derivation Rules in Information Systems, *Data and Knowledge Engineering*, 26 (1998), pp. 225-241.
23. Winter, R., Using Invariants of an Extended Conceptual Model to Generate Reusable Consistency Control, in: Nunamaker, J.F. / R.H. Sprague (Eds.): *Proc. of the 30th Hawaii Int. Conf. on Systems Sciences*, Vol. 3, IEEE Computer Society Press, 1997, pp. 168-178.
24. Winter, R., Towards an Integration of Structured Techniques for Data Modeling and Function Modeling in Information Systems Development, in: Dias Coelho, J. et al. (eds.): *Proc. 4th European Conference on Information Systems*, Lissabon 1996, pp. 1003-1010.
25. Winter, R., Formal Validation of Schema Clustering for Large Information Systems, in: Ahuja, M.K. / D.F. Galletta / H.J. Watson (Eds.): *Proc. of the First Americas Conference on Information Systems*, Pittsburgh 1995, pp. 448-450.