Entity-Relationship Modeling Revisited

Antonio Badia
Computer Engineering and Computer Science Department
University of Louisville
Louisville KY 40292
abadia@louisville.edu

ABSTRACT

In this position paper, we argue the modern applications require databases to capture and enforce more domain semantics than traditional applications. We also argue that the best way to incorporate additional semantics into database systems is by capturing the added information in conceptual models and then using it for database design. In this light, we revisit Entity-Relationship models and investigate ways in which such models could be extended to play a role in the process. Inspired by a paper by Rafael Camps Pare ([2]), we suggest avenues of research in the issue.

1. INTRODUCTION

Entity-Relationship modeling is a basic tool in database design, and as such it plays a very important role in Information System modeling. It is widely agreed that E-R models do a good job of capturing the basic semantics of many different situations. While this model, like any other conceptual models, is limited on its expressive power, this limitation is a necessary trade-off for its formality, simplicity and wide applicability. In fact, part of the modeling process is to get rid of information that is out of scope or not relevant to the application at hand ([6]). Thus, the E-R model is praised for having drawn the line between generality and expressive power at a good point, behind which we can expect diminishing returns.

However, as databases are asked to support more complex applications, capturing more domain semantics is becoming a more pressing need. Including more semantics can only be achieved if the domain semantics are gathered and represented in the conceptual model, at the requirement specification phase ([22]). Therefore, exploring enhancements and extensions to the E-R model becomes a legitimate and important area of research.

Since, as stated above, the E-R model has achieved a good equilibrium between expressive power on one hand and simplicity and wide applicability on the other, any addition should be very well *motivated*, in the sense that it should be shown to be truly needed, with the benefits of adding it to the model clearly outweighing any drawbacks. Moreover, such addition should be

- *minimal*, in order to protect the simplicity of the E-R model, which is acknowledged as one its virtues;
- *at the conceptual level*, to fit with the rest of the model. This means that any constructs added should be *declarative*, *high-level*, and independent of any computer implementation.
- *powerful*. In order to justify the addition, any construct should substantially contribute to the semantics of the model, allowing analysts to capture relevant information not previously available. In particular, any new constructs should have a clear and positive impact on database analysis and design.

Clearly, this is a tall order that will be difficult to fulfill. Since capturing semantics is a never-ending task, there has to be a clear goal in mind (and, if possible, a cost/benefit analysis) for any extension proposed. What kind of additions should we consider? Where should the model be extended? In this paper, we look at the structural makeup of the E-R model, in order to suggest some possible extensions. We illustrate the extensions with examples, to provide the motivation. We also discuss the larger issue of capturing semantics and the role of conceptual models. We close by offering some (tentative) conclusions. Our goal is to motivate and encourage research in conceptual modeling and to engage the community in a discussion on the foundations of such research.

2. CHARACTERISTICS OF ENTITY-RELATIONSHIP MODELS

One way to understand the strengths and weakness of a model is to look at its internal structure, the basic elements that it is built upon and the way in which they can be composed. E-R models have a *layered approach* to organizing information, in that the basic components of an E-R model, *attributes*, *entities* and *relationships* can only be combined in certain ways, not freely ([4, 29]). In particular, only entities and relationships can have attributes and only entities can be involved in relationships. Thus, it is not possible for attributes to have attributes, or to be involved in relationships; and it is not possible for relationships to be involved in other relationships. We call the information expressed by relationships on attributes *attribute constraints*

(since it is usually about some condition on the values that some attribute(s) can take) and the information expressed by relations on relationships *relation constraints*. These two cases are not the only ones; we can have lack of information about an entity and some relationship that the entity is involved with[1]. Not being able to express these kinds of constraints obviously puts bounds on what can be expressed in the model; the question is whether these bounds affects E-R models in such a way as to justify removing the limitations, and if so, whether what is to be gained positively contributes to the database design.

In a paper appeared in this journal ([2]), Rafael Camps Pare points out a problem in a seemingly well understood process: that of translating n-ary relations in an E-R diagram to a relational design. The paper is important and timely, not only because of the particular issue it analyzes, but because it calls attention to a larger issue. Camps Pare gives an example in which the usual transformation process from an E-R model to a relational design fails to capture all needed information to ensure a correct database state; additional constraints, enforced through checks, are needed. Additionally, Camps Pare claims that the problem is considered (by the database research community) elementary and solved and therefore, no further research is needed on it, although this is not the real situation. We would like to take this claim further by using Camps Pare's example as the starting point of a more detailed analysis. We observe that, in his example, he introduces information about a ternary relationship that cannot be represented in an E-R model, and cannot be captured in a relational database in the form of a key or integrity constraint. Thus, Camps Pare's example fit in the second category, and has a clear influence on the database design. We argue, via examples, that this situation is a truly relevant (and open) issue.

2.1 Relation Constraints

First, we repeat Camp Pare's example for completeness, and then show several more examples of cases in which relation constraints cannot be expressed in the E-R model and affect the way in which data should be manipulated in the database.

EXAMPLE 1. *Let* Concession *be a ternary relation between entities* Dealer, Product *and* State, *and with an attribute* concession-date. *The relationship is 1 in the* Dealer *side, and many on both* Product *and* State*; the intended meaning is that each product must be sold by a single dealer in each state. Then, two new rules are added: a) each product is distributed by a single dealer, regardless of state; and b) in each state, there is only one dealer. These conditions are properties of the ternary relation (even though they involve only two of the three participating entities), because they constraint the set of legal tuples that can appear on it. As stated above, there is no way to express this constraint in an E-R model; even if explicit binary relations were added, the connection between these relations and the original one cannot be expressed.*

[1]As a quick example, assume an E-R model for a company, which contains entities *Client* and *Representative*. The representatives are employees whose mission is to interact with and assist clients; a business rule is that every client must have a representative, and that every representative must assist several clients. This creates a relationship, which is one-to-many and total on both entities. Note that such a relationship induces a partition on the set of clients. Many properties of such partitions cannot be expressed in the E-R model; for instance, a rule stipulating that all representatives must have the same number of clients (i.e. all sets in the partition have the same size.

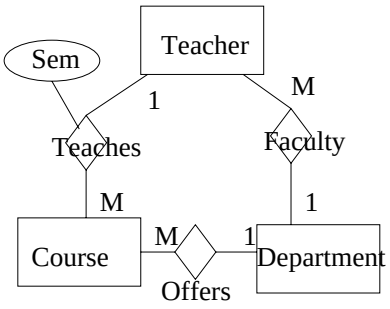

Figure 1: E-R diagram for example 2

The standard procedure for producing a relational design from an E-R model would create one relation for each entity and another relation for the ternary relationship; integrity constraints, based on primary key-foreign key relations, would be added to a proper design. However, the additional information would have to be enforced through checks. Moreover, the design would not be in BCNF anymore (it would have been before under the assumption that all functional dependencies are those generated by keys). Camps Pare goes on to show how to use checks and restrictions to include this additional information and enforce the consistency of the data[2].

The following example can be considered a *complement* to Camps Pare's, since it deals with three binary relationships which should be constrained by a ternary relationship.

EXAMPLE 2. *Assume an E-R model for a university. The model contains entities* Teacher, Class *and* Department. *There are relationships* Teaches *between* Teacher *and* Class *(with attribute* Sem *to indicate a particular semester),* Offers, *between classes and departments, and* Faculty, *between teachers and departments, with the obvious interpretation. A university rule is that teachers can only teach classes offered by the department in which they are faculty. This rule cannot be enforced in the E-R model.*

A figure for this example is given (Figure 1); the figure assumes that each teacher is faculty in one department, and may teach several courses; a course is taught by one teacher and offered by one department; and a department has several faculty and offers several courses. A standard relational design translation of this diagram would generate the following tables (in our examples we assume, for simplicity,

[2]Note that all the examples in this paper may be expressed in constraints (checks or assertions), rules (triggers), or in code (applications). In particular, SQL-99 assertions allow us to express rules that span several tables, and thus are especially useful for relation constraints, which usually do ([20]). Indeed, part of the drive behind refining checks and assertions to SQL-99 is to allow database designers to include more data semantics *within* the database. We note, though, that many systems do not fully implement this part of the standard.

that all entities have a key attribute, indicated with an *-id* ending; we give only the relevant attributes of each schema, and assume the obvious referential integrity constraints):
Teacher(teacherid,. . . ,deptid)
Class(classid,. . . ,deptid, teacherid, sem)
Department(deptid,. . .)

It is clear that even after declaring all integrity constraints, the rule above is not enforced; an additional check must be added, stating that

```
NOT EXISTS (SELECT * FROM TEACHER, CLASS
            WHERE TEACHER.TEACHERID = CLASS. TEACHERID
            AND TEACHER.DEPTID <> CLASS.DEPTID)
```

Note that different cardinalities in the relationships involved may result in different checks. For instance, if *Faculty* were many-to-many (i.e. a teacher can be faculty in more than one department), the rule becomes much harder to write in SQL. A more complex example is the following.

EXAMPLE 3. *Let there be a recursive relationship* ManagerOf *on an entity* Employee*. Two roles are associated with* Employee *through this relationship:* manager *and* managee*. Such relationship is partial on the* manager *role (not all employees are managers) and total on the* managee *role (all employees have a manager). This relationship has several properties which cannot be expressed in the model: it is an irreflexive relationship (no one can be his or her own manager). In most situations, it will also be asymmetric (if employee* a *is the manager of employee* b*, it cannot be that employee* b *is, in turn, a manager of* a*) and transitive (managers have higher-level managers, and so on).*

Again, the relationship properties can be used by a system to check modifications (insertions, deletions and updates) for correctness, but cannot be represented in a E-R model and cannot be stated in terms of integrity constraints in the relational design. In this case, it may be argued that even a check will not do: for the combination of irreflexivity and transitivity implies that there can be no cycles in *Employee* (i.e. employee e_1 being the manager of e_2 which in turn is the manager of e_3,. . . , which in turn is the manager of e_1). However, this condition cannot be checked without the ability to express recursion. Despite the addition of *WITH RECURSIVE* clause to SQL-99 ([20]), we suspect it will still be a while until checks and assertions in commercial database management systems fully and efficiently support the feature[3].

Yet another example shows some more subtle problems.

EXAMPLE 4. *This example is taken from [5]. Let entities* Division*,* Staff *and* Branch *be related through relationships* IsAllocated*, between* Division *and* Staff*, and* Operates*, between* Division *and* Branch*. Participation in* IsAllocated *is total, 1 on the* Division *side and many on the* Staff *side (i.e. all staff members are allocated to one and only one division, and each division is allocated one or more staff members). Participation in* Operates *is total, 1 on the Division side and many on the branch side (i.e. all divisions are assigned one or more branches, and all branches are assigned to one and only one division). The situation is shown in figure 2. One could assume that staff members work at particular branches, and that such a relationship between elements of* Staff *and* Branch *can be inferred from the two explicit relationships. However, we note that both relationships are 1-M, with the 1 on their common domain (*Division*). Therefore some element in* Staff *may be related to more than one branch, even if one one would expect that each staff member works in only one branch.*

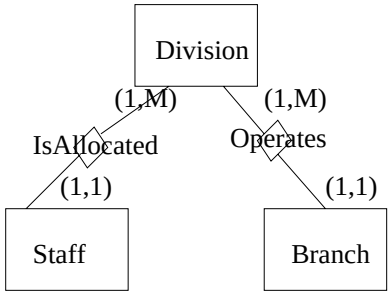

Figure 2: E-R diagram for example 4

This is called the *fan trap* in [5]. Standard relational design would create the following relations:
Division(divisionid,. . .)
Staff(staffid,. . . ,divisionid)
Branch(branchid,. . . ,divisionid)

The fan trap can now be described as the fact that a join between *Staff* and *Branch* on *divisionid* would bring the (unexpected) result of a tuple in *Staff* being joined to more than one tuple in *Branch*. In order to avoid this problem, one can add another explicit, binary relationship between *Staff* and *Branch*, say *Works-at*. The question is, what is the relationship between this new relationship and the other ones? Assuming *Works-at* is one-to-many (i.e. each staff member works in one branch, but a branch may have several staff members), in the relational database, the *Stuff* relationship would be changed with the addition of *branchid* as another foreign key. While this would avoid the fan trap, enforcing a rule stating that a staff member can only work at a branch operated by the division where the member is allocated would take us back to the previous situation.

EXAMPLE 5. *Assume a relationship* `Works-In` *between entities* `Employee` *and* `Project`*, with attributes* `start-date` *and* `end-date`*. The rule* an employee cannot work in two projects at the same time *(i.e. an employee must work in one project at a time) cannot be expressed in the E-R model. Note that the relationship is still many on the employee side, since an employee may work in several project,* just not at the same time*. Moreover, the rule creates a functional dependency* eid start-date end-date → pid*, where* eid *is an employee id and* pid *is a project id (both keys for their respective entities). Assuming the relationship is also many on the* `Project` *side, a relationship* EMP-PROJ(eid, start-date,end-date,pid) *would be created. The functional dependency identifies the key of the relationship (the combination (*eid start-date end-date*)). It is therefore important to know this information at design time.*

What do all these examples, including Camps Pare's, have in common? They express properties of relationships, which

[3]Even though some systems, like ORACLE, already provide a functionality similar to *WITH RECURSIVE*, such functionality is not allowed in check or assertion statements.

cannot be expressed in an E-R model. However, they are all relevant to the database semantics since they control which are correct database states. Note that in our examples the relations remain in the same normal form as before, while in Camps Pare's example BCNF is lost. Thus, in some cases the added information is not about normal forms, or integrity constraints, and cannot be captured in the database with key and integrity constraints; the use of checks or assertions is a must[4]. The point remains that the semantics of the domain are not adequately captured without expressing these properties.

2.2 Attribute Constraints

The other level at which information is missing is the attribute level. We illustrate it with an example.

EXAMPLE 6. *Assume an E-R model for a company where* employee *is an entity, and* salary *one of its attributes. Assume further that employees in the company are assigned a rank, and that the rank determines a* salary range *for the employee (a maximum and minimum allowed for the employee's salary).* Rank *may be another attribute of* Employee*; however, since it has associated information (the minimum and maximum values) it may be modeled as an entity, with two attributes* maximum *and* minimum *and a relationship joining it to* employee. *In either case, there is no way to express, in the E-R model, that the salary of the employee should be within the range determined by his or her rank.*

Observe first that in this case the information has nothing to do with normal forms; a translation into relational table would yield a seemingly good design, with tables *Employee(employeeid, rankid,salary...)* and *Rank(rankid, maximum,minimum,...)*. Note also that this information can be captured, as in previous cases, with checks or assertions; in this case, we would an assertion similar to our previous example:

```
NOT EXISTS (SELECT * FROM EMPLOYEE, RANK
            WHERE EMPLOYEE.RANKID = RANK.RANKID
            AND (SALARY < MINIMUM OR SALARY > MAXIMUM))
```

The absence of information at the attribute level is at the root of some of the most important limitations of database systems today: their inability of sharing data. The field of *heterogeneous information integration* has studied this problem in depth; while we cannot even review all research in a limited space (see [9] for an excellent overview), we point out that most approaches call for semantic information about attributes in order to deal with heterogeneity. As an example, imagine two databases with information about restaurants that are to be integrated. Both have a table *restaurant* with an attribute *meal-cost* in them. However, in one the meal cost may include taxes, and not include them in the other. The price may be in American dollars in one, in Canadian dollars in another. As another example, imagine two two colleges which merge, each one with its own database system -which must also be merged. Both have a table *student*, with attribute *grade*. However, in one database grade is expressed as one of A,B,C,D,F; in the other, the same information may be expressed as a number between 1 and 10, or between 1 and 100 (note that there is a difference in precision between the scales)[5].

In most approaches, deciding about semantic relationships between entities and relations comes down to deciding about semantic relationships between attributes. Even when there are differences in the data model (called *structural (or schematic or representational)* differences [19]), which can be solved by *restructuring* the data, one still needs to find out if (basic) classes are compatible (or identical). The exact relationship between the meaning (semantics) of two representations has to be established; this is called *semantic* differences. This refers to differences "about the meaning, interpretation or intended use of data" ([16]). Determining the exact relationship among classes (equality, subsumption, intersection, disjointness) classes greatly depends on relationships among the attributes in those classes. Thus, identifying the right relationships among attributes is crucial ([18, 24, 25, 26, 17]).

Thus, most approaches to solving semantic heterogeneity must start by analyzing the semantics of attributes. Given that most database systems keep a very poor set of such metadata, enriching the database schema is one of the first task in most approaches ([3, 11, 27, 28]). This is a very complex task, which cannot be fully automatized[6]. It seems desirable that conceptual models try to capture at least some of this metadata.

3. DISCUSSION: CAPTURING SEMANTICS

As stated in the beginning, any extension of a conceptual model should be well motivated. Since the E-R model manages to balance expressivity and complexity quite nicely, it should even be discussed whether any extensions are necessary.

We argue that new applications, from E-commerce to Decision Support to Document and Workflow Management, demand more semantics from database systems. Traditional applications, based on a model of small, local transactions, involved simple, localized database access. Today's applications require complex, global data manipulation. It is not enough to achieve a certain normal form anymore; it is also necessary to ensure (as much as possible) that the data semantics cannot be misread by applications by adding checks and assertions that enforce meaningful manipulation of database contents. Note that Camps Pare already points out that referential integrity constraints alone will not enforce correct database states, but the lost semantics are perfectly representable in a relational database -through checks and assertions. The argument now is that the use of such checks and assertions should be considered an *intrinsic* part of relational design.

Even if one agrees with this point of view, it is still possible to argue that semantics cannot be completely captured in *any* conceptual model. Therefore, the limitations of the

[4]In the last example, a key dependency expresses the constraint perfectly.

[5]And these examples are simplified! The same information may be called *score* in one of the systems (or worse yet, it may be called *grade* and mean something completely different).

[6]A very similar problem is observed in integration of legacy systems, where one of the main obstacles to analyzing the system is the absence of documentation and metadata, coupled with the fact that the original designers are usually not at hand to tell us what the intended meaning of all those attributes was.

E-R model are an expected trade-off for their ease of use, intuitive appeal and clear semantics. However, an underlying assumption is that E-R models capture enough semantics to guarantee a good relational design. The examples in subsection 2.1 tells us that it is possible to insert into the database incorrect information even if the design is technically correct. Thus, even if a boundary will always exist, the argument here is that it has come time to move it.

Finally, one could also argue that on real-life systems development, an E-R model is only one tool in a thorough Requirement Specification phase, and that a good designer will pick up information like the one in our examples in one way or another and will incorporate it into a Requirement Specification Document. This whole document, and not only the E-R model, should be taken into account when designing the database. However, many of the examples above would be simply written down in natural language, because of a lack of tools to (semi)formally express such constraints[7]. This information tends to be lost and not make it into the database design, since it may be difficult to interpret (because of vagueness and ambiguity) and there are no established methods to incorporate such information into a design. We contend that the E-R model is the place where the added information should be placed, for at least two reasons. First, the new, additional information carries its full meaning only in the context of the basic information already present in the E-R model. Second, the new information must be used in the database design process. There are well known methodologies to transform an E-R diagram into a database design; therefore, adding information to the E-R model opens up the possibility of extending said methodologies as a proven way to incorporate the extra semantics into database design. At the very least, having all information gathered together in one model increases the chances of that information being used in its totality in a consistent, coherent manner.

4. CONCLUSION

E-R models are widely used because of their simplicity, intuitive appeal and ability to capture useful semantics across many different domains. In the past, it may have been argued that, while having more semantics is always desirable, the E-R model offered a good trade-off point. However, with databases called to support more complex and sophisticated applications, the need to capture more domain semantics is growing. The structural information that the E-R model contains give us enough to determine normal forms, which in turn is the basis to ascertain the correctness of the database schema. We now need more: correctness of the database should guarantee that changes to the extension (insertions, deletions or updates) do not lead to a state which deviates from what is allowed in the real world. Certainly, the problem is well known within the database research community; probably, most of this paper is *common knowledge* in certain circles. However, the amount of research devoted to it seems to be quite limited[8]. It seems, then, that more research should be devoted to the issue.

What are some possible avenues of research? In this author's opinion, researchers should carefully develop and add constructs to the E-R model. Such constructs should follow the requirements laid out in the introduction (well motivated, minimal, conceptual, and powerful). This implies that the benefits of any possible construct (expressive power, wide use) should be measured and balanced against its drawbacks (complexity). We suggest that research on relationships over relationships, and attributes over attributes (and relationships over attributes) are promising approaches. We also emphasize the need to show how to use any additional semantics in database design.

In the end, it is clear that no conceptual model will ever capture all the semantics of an application, and that there is a need to balance expressiveness and complexity. However, we should periodically revisit the boundaries of our models in an attempt to expand them. Precisely because of the usefulness and importance of the E-R model, we should revisit it often.

[7]We do not wish to start here a war on conceptual models; we simply will acknowledge the existence of OCL ([23]) and hope that some day it will be given proper semantics.

[8]There is clearly a wealth of research in E-R models (see for instance [1, 7, 8, 13] for attempts to add more expressive power to the model). However, as far as the author is aware, most of these attempts do not move the E-R model towards solving the problem discussed here.

5. REFERENCES

[1] Atzeni, P., Batini, C., Lenzerini, M., Villanelli, F. *INCOD: A system for conceptual design of data and transactions in the entity-relationship model*, in Proceedings of the 2nd International Conference on Entity-Relationship Approach to Information Modeling and Analysis, 1981.

[2] Camps Pare, R. *From Ternary Relationships to Relational Tables: A Case Against Common Beliefs*, SIGMOD Record, v. 31, n. 2, June 2002.

[3] Castellanos, M. *Semantic Enrichment of interoperable databases*, in Proceedings of the RIDE-IMS, 1993.

[4] Chen, P. *The Entity-Relationship Model -Towards a Unified View of Data*, ACM Transactions on Database Systems, v. 1, n. 1, 1976.

[5] Connolly, T., Begg, C. and Strachan, A. *Database Systems*, Addison-Wesley, 1999.

[6] Davis, F. *Requirements Specification: Objects, Functions and States*, Prentice-Hall, 1993.

[7] Dos Santos, C. S., Neuhold, E. J. and Furtado, A. L. *A Data Type Approach to the Entity-Relationship Model*, in Proceedings of the 1st International Conference on Entity-Relationship Approach to Software Engineering, 1980.

[8] Eder, J., Kappel, G., Tjoa, A. and Wagner, R. *BIER: The Behavior Integrated Entity Relationship Approach*, in Proceedings of the 5th International Conference on Entity-Relationship Approach, 1986.

[9] *Heterogeneous and Autonomous Database Systems*, Elmagarmid, A., Rusinkiewicz, M. and Sheth, A. eds., Morgan Kaufmann, 1999.

[10] Fankhauser, P., Kracker, M. and Neuhold, E. *Semantic vs. Structural Resemblance of classes*, SIGMOD Record, Special Issue on Semantic Issues on Multidatabases, v. 20, n. 4, 1991.

[11] Garcia-Solaco, M., Castellanos, M. and Saltor, F. *Discovering interdata resemblance of classes for interoperable databases*, in Proceedings of the RIDE-IMS, 1993.

[12] C Goh and S. Bressan and S. Madnick and M. Siegel, *Context Mediation: New Features and Formalisms for the Intelligent Integration of Information*, Sloan Working Paper 3941, 1997.

[13] Gogolla, M. and Hohestein, U. *Towards a Semantic View of en Extended Entity-Relationship Diagram*, ACM Transactions on Database Systems, v. 16, n. 3, 1991.

[14] R. Hull, *Relative Information Capacity of Simple Relational Database Schemata*, SIAM Journal of Computation, Volume 15, number 3, August 1986.

[15] R. Hull and C. K. Yap *The Format Model: A Theory of Database Organization*, Journal of the ACM, 31(3), 1984.

[16] Kashyap, V. and Sheth, A. *Semantic and Schematic Similarities between Database Objects: A Context-based Approach*, VLDB Journal.

[17] Kim, W., Choi, I., Gala, S. and Scheevel, M. *On Resolving Schematic Heterogeneity in Multidatabase Systems*, Distributed and Parallel Database Systems, Kluwer Publishers, v. 1, n. 3, 1993.

[18] Larson, J., Navathe, S. and Elmasri, N. *A Theory of Attribute Equivalence in Databases with Application to Schema Integration*, IEEE Transactions on Software Engineering, vol. 15, number 4, April 1989.

[19] Miller, R. *Using Schematically Heterogeneous Structures*, Proceedings of the ACM SIGMOD International Conference, 1998.

[20] Melton, J. and Simon, A. R. *SQL 1999: Understanding Relational Language Components*, Morgan Kaufmann, 2002.

[21] Miller, R., Ioannidis, Y. and Ramakrishnan, R. *Schema Equivalence in Heterogeneous Systems: Bridging Theory and Practice*, Information Systems, vol. 19, 1994.

[22] Pressman, R. *Software Engineering*, McGraw-Hill, 1997.

[23] Rumbaugh, J., Jacobson, I. and Booch, G. *The Unified Modeling Language Reference Manual*, Addison-Wesley, 1999.

[24] Song, W. W., Johannesson, P. and Bubenko, J. A. *Semantic Similarity Relations in Schema Integration*, in Proceedings of the 11th International Conference on the Entity-Relationship Approach, 1992.

[25] Li, W. S. and Clifton, C. *Semantic Integration in Heterogeneous Databases using Neural Networks*, in Proceedings of the 20th VLDB Conference, 1994.

[26] Spaccapietra, S. and Parent, C. *View Integration: A Step Forward in Solving Structural Conflicts*, IEEE Transactions in Knowledge and Data Engineering, 6(2), 1994.

[27] Sciore, E., and Siegel, M., and Rosenthal, A. *Context Interchange Using Meta-attributes*, in Proceedings of CIKM, 1992.

[28] Sciore, E., Siegel, M. and Rosenthal, A. *Using Semantics Values to facilitate interoperability Among heterogeneous Information Systems*, ACM Transaction on Database Systems, June 1994.

[29] Thalheim, B. *Entity-Relationship Modeling: Foundations of Database Technology*, Springer-Verlag, 2000.

[30] Ouksel, A. and Naiman, C. *Coordinating Context Building in Heterogeneous Information Systems*, Journal of Intelligent Information Systems, 1993.