THEORETICAL FRAMEWORK FOR DEFINING VALIDITY AND QUALITY IN MODELING

(Research-in-Progress)

Fatma Mili
Computer Science and Engineering
Oakland University, MI48309-4478
mili@oakland.edu

Krish Narayanan
Computer Science
Eastern Michigan University, MI48197
Krish.Narayanan@emich.edu

Abstract The increasing number of Object Oriented databases and knowledge bases raises the need for some criteria of model validity, modeling guidelines, and quality criteria to be used in modeling. Although some researchers have developed some specialized sets of modeling guidelines, we found no general-purpose framework to evaluate these proposals and customize them. In this paper we develop a theoretical framework for characterizing modeling. Modeling is seen as a transformation process. These transformations must be valid, and must increase the quality of the model. The framework is centered around these two issues of validity and quality.

1. KNOWLEDGE MODELING

Modeling is a central activity in databases. It consists of defining a set of constructs (relations, classes, constraints) that, together, capture the contents of interest and exhibit model-specific structural quality (normal forms, schema invariants). Software design is also a modeling activity. According to Jocabson et al. [14], "In design we shape the system and find its form (including its architecture) that lives up to all requirements – including nonfunctional requirements and other constraints – made on it." In other words, design is a modeling activity selecting a form to capture contents. Developing knowledge bases is also driven by the same concerns of capturing contents while exhibiting some structural quality. Overall, regardless of the domain, modeling consists of selecting a form to capture contents. The same contents can be captured in different ways. While the different materializations of the same contents are semantically equivalent, they are not necessarily equivalent with respect to other nonfunctional criteria (readability, reusability, etc.).

Quality is an underlying concern in all applications. Traditionally, the term "quality" is used to encompass compliance with nonfunctional requirements, while the terms, "correctness" or "validity" are used to characterize models that capture the right contents. A number of authors have defined sets of

quality criteria for specific applications, for example, databases [18,22,25], software engineering [7,8,12], software engineering specification [13,24], and knowledge bases [11,19]. Conspicuously absent in the literature, is a general foundational framework defining the concepts of validity and quality. Without such general framework, it is hard to assess individual proposals. In the absence of such a general framework, it is difficult to assess the merit of individual proposals, to compare different sets of quality criteria, and to reuse and adapt quality criteria across domains of application. Without such general framework, we have no basis for comparing the quality criteria put forth by Emam et al [7], with those presented by Tari et al [23]. Also, without a general framework, it is hard to assess the reusability value of any of them in the context of a knowledge base represented using Description Logics [1,5].

In this paper, we introduce formal concepts underlying the notions of modeling validity and modeling quality. We formulate these concepts using the paradigm of modeling by transformation as implied by Tari et al [23]: "Given an initial [database] schema, find an *equivalent* one that is *better* in some respect." In other words, we conceptualize the process of modeling as a process of transformation that starts from an initial model and transforms it iteratively until a satisfactory solution is reached. Figure 1 shows a hypothetical step of the modeling process. The overall process is not an arbitrary transformation process, though. It is highly constrained:

- Transformations must be valid: They must transform a model into an "equivalent" model (e.g. same data, same functionality).
- Transformations must be useful: They must transform a model into a "better" model (e.g. more modular, more reusable).

Any modeling effort is driven by these two concerns. In Figure 2, we show the duality between the two concepts of equivalence and preference. The circle represents the universe of all possible schemas. The slices within the circle represent sets of (semantically/functionally) equivalent schemas. Within each slice, some schemas are preferred to others. The concentric circles are used to represent different levels of quality. For example, the closer to the center of the circle, the "better" is a schema. Modeling transformations are constrained by the fact that they must transform a schema into another that is equivalent (thus in the same slice) yet useful (thus closer to the center) than the original schema.

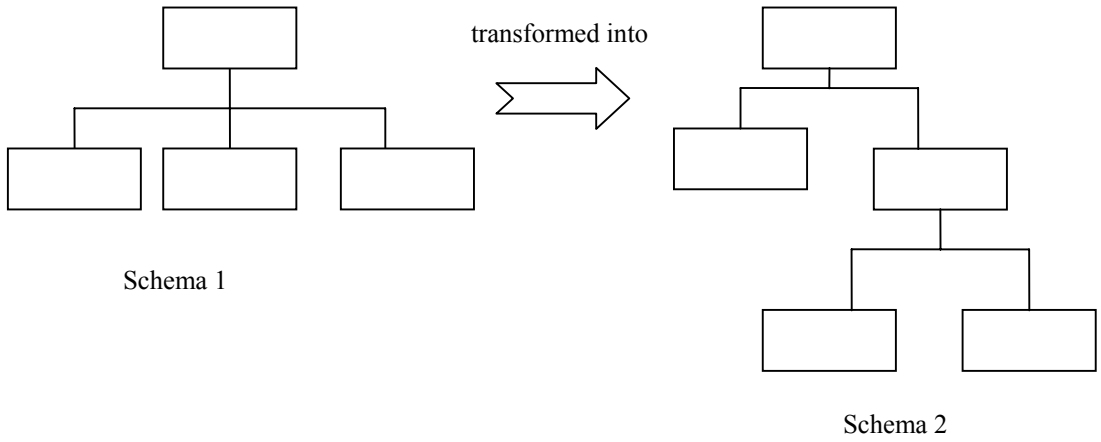

Figure 1: Modeling by transformation

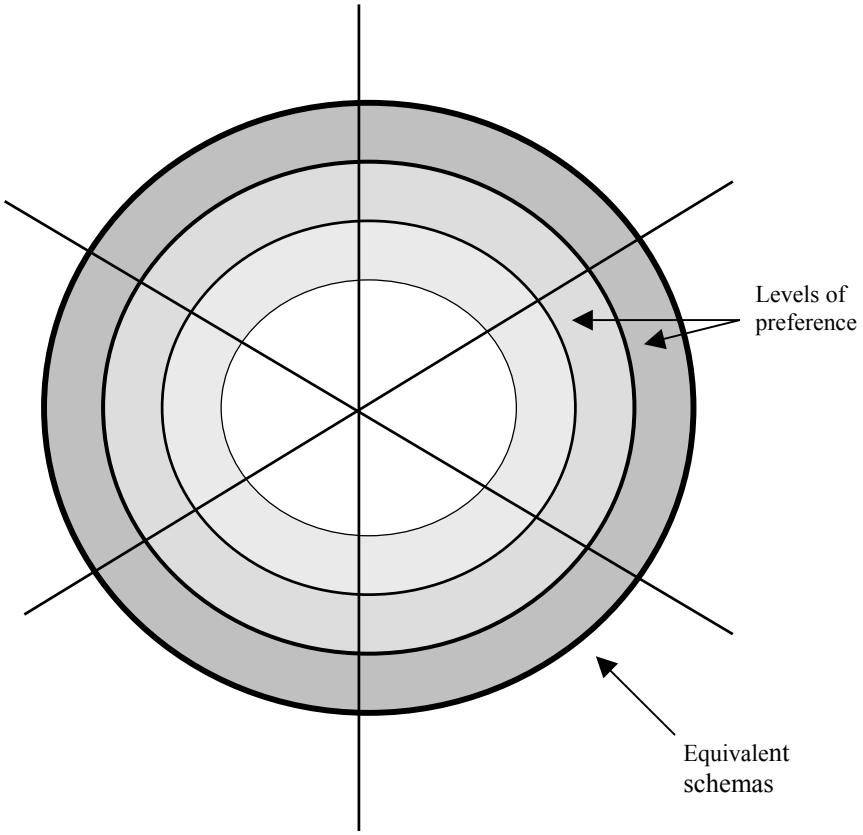

Figure 2: Equivalence and Preference relations

This paper is organized as follows. In section 2, we discuss the notion of modeling validity. We define requirements for the underlying equivalence relations and develop guidelines for constructing these relations. In section 3, we discuss the notion of modeling quality. We identify common forms for the underlying preference relations and discuss the relationship between them. Together, the two notions of equivalence and preference are the cornerstones of quality assurance. They provide the basic ingredients for defining creation and update processes that assure quality. In section 4, we put the two together and unify them around the concept of transformation. We summarize and conclude in section 5.

2. MODELING VALIDITY

When modeling a database, designers subject the initial schema to a sequence of transformations until all relations are in some pre-defined normal form. When modeling an object-oriented model, designers iteratively restructure the initial schema introducing new classes, modifying associations, and moving attributes across classes to increase modularity and maximize reuse for example. In both cases, designers are driven by the goal of imparting the schema with quality properties that it did not initially possess while at the same time ensuring that the information content of the initial schema is preserved.

The validity of transformational systems has been paradoxical: On the one hand, validity has been the very impetus for using transformational systems. The concept of design/modeling by transformations has its origin in Dikjstra's seminal work in top-down stepwise refinement [6], motivated by the premise that correctness of design can be more easily established when design decisions are made in small, provably

correct steps. On the other hand, using transformational systems does not automatically ensure correctness. The individual transformations must be correct and their composition must also be correct. This latter aspect, in particular has often been neglected. Sands [20] for example, shows that many in many transformational systems composition of transformations are often not correct.

In this section we establish generic frameworks for defining validity of transformations for information systems. In software design, validity is based on well-established criteria of software correctness. To discuss the validity of data modeling transformations, we need to discuss the concepts of correctness and what it means to preserve and refine the "contents" of databases and knowledge base schemas.

2.1 Informational contents of database and knowledge base schemas.

Databases and knowledge bases are used to make decisions on the basis of query results. Queries can be binary, e.g., "is this the most promising stock?", numerical, e.g. "how much was spent on supplies during the month of July 2003?", or of arbitrary type, e.g. "identify the set of documents related to information quality". The contents of a database are thus reflected in the results returned by the queries submitted to it. Thus, intuitively, schema transformations are valid if queries submitted to the database under the transformed schema return the same result as if they were submitted under the initial schema. In other words,

> ***Two database schemas S_1, S_2 are considered to be equivalent if and only if queries submitted to a database return the same results whether the database is modeled using schemas S_1 or S_2.***

This definition needs to be refined in two ways:

- The definition of schema equivalence requiring identical results on <u>all</u> queries is too stringent. Given two distinct schemas S_1, S_2 there is always at least one query whose results will be different when submitted to databases using schemas S_1 *and* S_2 respectively. For example, the query "show the database schema" would distinguish between any two distinct schemas. The definition of equivalence must therefore be defined relative to a set of queries of interest:

  > ***Given a set Q of queries, two database schemas S_1, S_2 are considered to be equivalent if and only if every query q from Q submitted to a database returns the same results whether the database is modeled using schemas S_1 or S_2.***

- The definition of schema refers to a (same) database modeled using different schemas. To clarify the meaning of this statement, consider the following sequences of events:
  Seq1: Create schema (DB1, S1); Add employees {E1, E2, E3}; Add projects {P1, P2}; …
  Seq2: Create schema (DB2, S2); Add employees {E1, E2, E3}; Add projects {P1, P2}; …
  The above two sequences start with the creation of a database (DB1 and DB2) using schemas S1 and S2 respectively. They then proceed with identical histories. Under such conditions, we say that DB1 and DB2 have identical update histories. We refine the definition of equivalent schemas accordingly:

  > ***Given a set Q of queries, two database schemas S_1, S_2 are considered to be equivalent if and only if every query q from Q submitted to any pair of databases DB1, DB2 with identical update histories, modeled using schemas S_1 or S_2 respectively returns the same results.***

The notions of equivalent database histories and equivalent schemas are defined more formally and in more details in [16]. The definitions above are sufficient for the purposes of this paper.

2.2 Necessary and Sufficient Conditions for Valid Transformations

Modeling by transformations generally consists of a sequence of refinements and decompositions whereby a schema S_i is transformed into schema S_{i+1} or decomposed into S_{i1} ¤ S_{i2} where ¤ is some composition operator. Transformational systems are valid if the final product (S_f) is correct with respect to the initial product (S_0). Transformational systems are guaranteed valid by ensuring that

- Individual transformations are valid, e.g.. the transformation of S_i into S_{i+1} is correct.
- The composition of transformations is a also valid.

In this paper, we only focus on simple transformations refining one schema into another.

We define IC_Q (for identical contents relative to queries in Q) to be the relation containing all pairs (S_1, S_2) of schemas that are equivalent, i.e.,
IC_Q={(S_1, S_2)| S_1 and S_2 are equivalent relative to queries Q as defined above}.

Individual transformations are valid –relative to Q- if they transform a schema S into a schema S' such that (S, S') belongs to IC_Q.

It is obvious from its definition that IC_Q that it is reflexive, symmetric, and transitive. Because of the transitivity, in particular, a composition of transformations valid with respect to a set of queries Q is also valid with respect to the same set of queries. In other words,

A sequence of individually valid (relative to Q) transformations is valid relative to Q.

The above two clauses establish the validity of using $\boldsymbol{IC_Q}$ as a basis for modeling transformations. Yet, for all of its theoretical interest, relation IC is often not convenient to work with. Its definition does not always lend itself to a closed form expression. In fact, it is not necessary to compute IC as long as we can verify that a transformation would generate a pair in IC. In practice, it is sufficient to know of some valid transformations, and use only those. In other words, rather than aiming at formulating and using IC, it suffices to identify an equivalence relation EQ such that, EQ is an equivalence relation, and EQ is compatible with IC. Thus, the requirements on the relation EQ are:

- EQ is an equivalence relation, i.e. EQ is reflexive, symmetric, and transitive.
- All pairs in EQ are also pairs in IC.

In Figure 3, we show the relationship between the two relations IC and EQ. The equivalence classes of EQ are contained within equivalence classes of IC.

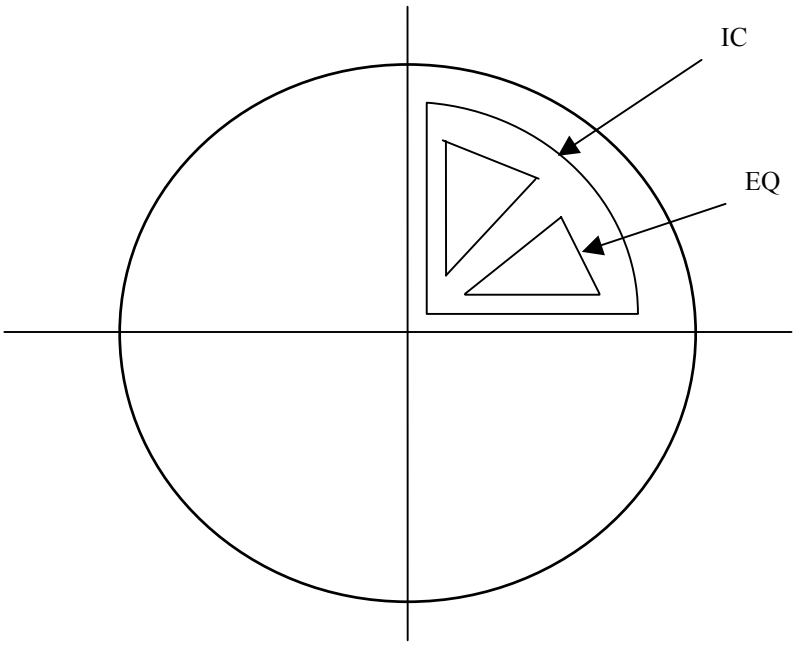

Figure 3: Sufficient vs. necessary validity conditions

2.3 Example, relational databases

We illustrate the concepts presented above using modeling in relational databases. We consider a set of attributes of interest: A_1, A_2,…, A_n and define U(A_1, A_2,..., A_n) as the universal relation. The set Q is the set of all queries that are expressions of U, i.e. those that only involve attributes in the set {A_1, A_2,…, A_n}. A database schema S is a set of relational schemas. Each relational schema is in turn, a relational expression of the universal relation U (projections, selections, aggregations). We say that each schema S is defined by an expression s(U). The universal relation U can be extracted from a schema S using the inverse of s-when it exists (U= s^{-1} (S)), where s^{-1} is the inverse expression of s.
We now define IC"

$$IC=\{(S,S')|\ \text{for all expression q on U, } q(s^{-1}(S))=q(s'^{-1}(S'))\}$$

The above definition associates S to S' by referring them back to U. We use it to extract a formulation relating S and S' to each other directly.
First we use the expression s as the query q, we get

$s(s^{-1}(S))=s(s'^{-1}(S'))$,

We use the assumption that S is a valid schema (i.e. $s(s^{-1})$ is the identity), we get

$S= s(s'^{-1}(s'(U)))$.

Finally, using U= s^{-1} (S), we get

$S= s(s'^{-1}(s'(s^{-1}(S))))$.

The above line states that the transformation from S to S' (captured by $s'(s^{-1}(S))$) is reversible, $s(s'^{-1}(s'(s^{-1})))$ is the identity. If we call t the direct expression such that S'=t(S), the condition for equivalence is that t^{-1} exists and that $t(t^{-1})$=identity. Thus,

$$IC=\{(S,S')|\ S'=t(S) \text{ and t is a reversible transformation}\}$$

The requirement that t be reversible ($t(t^{-1})$=identity) is the guarantee for non-loss transformation. We illustrate this by an example: Let S be a single relation schema S(A,B,C). Let t be the transformation of S

into S' with two relations R1(A,B) and R2(A,C). For the pair (S,S') to belong to IC, the transformation t must be reversible, that is we must have the join of the projected two relations is equal to the original relation for all histories. There is no universal way to determine whether t is reversible or not, but there are sufficient conditions under which we know that the answer is positive. One such condition is that B and C be functionally dependent on A (A→ B; A → C). The functional dependency based non-loss decomposition property is the most commonly used sufficient condition. It states that, given a relation R(A,B,C), such that B and C are functionally dependent on A, the decomposition of S into R1(A,B) and R2(A,C) is a non-loss decomposition, that is the pair of schemas (S, S'={R1,R2}) where S, R1 and R2 are as defined above is equivalent with respect to IC. Most database design algorithms are based on this sufficient property.

In practice, rather than proposing random transformations and checking after the fact whether they are reversible or not, correct reversible transformations are identified before hand and used. For example, relational database modeling by decomposition uses a single transformation that decomposes relations one at a time based on functional dependencies. In these cases, EQ is not identified explicitly. Instead, it is derived from T:

$P=\{(S, S')|\ S'=t(S) \vee S=t(S'), t \text{ in } T\}$

$EQ=P^*$

Here P^* is a reflexive transitive closure of P.

Proposition:

Given an equivalence relation IC, and a symmetric relation P, if relation $P \subseteq IC$, then relation $EQ=P^*$ is an equivalence relation and $EQ \subseteq IC$.

The proof of this proposition and additional examples of its use can be found in [16].

2.5 Equivalence and validity are relative concepts

In the above discussion, we have defined contents and equivalence relative to a pre-defined space. According to the definition given, the contents of a database are expressed in terms of results of pre-defined queries. Any additional relations or attributes will remain invisible. We justify this approach in two ways:

1. The definition of contents is not universal. Different definitions of contents lead to different definitions of validity.
2. The rationale for overlooking knowledge outside the space of interest.

First, to illustrate the fact that the definition of contents is not universal, we use the example of relational database modeling. In our example above, the definition of contents is end-user oriented. It only focuses on the query results returned to the users. This is one alternative perspective on contents, and is the minimal required. Some database design approaches take the perspective of a database designer and focus on preserving more than the data (e.g. functional dependencies).

The second aspect concerns the justification for having information that is outside the scope of the space of interest (invisible to the queries). This need may be better seen in the context of Object Oriented (OO) modeling. In OO models, we make extensive use of "abstract" classes introduced for convenience and reusability. We illustrate this with the example in Figure 4. In the figure, the designer is only interested in the three classes, square, rectangle, and diamond. The conceptual model to the left contains exactly those classes. In the model to the right, one class, Employee, and some associations have been added. The intent is that the added information does not affect the contents of interest.

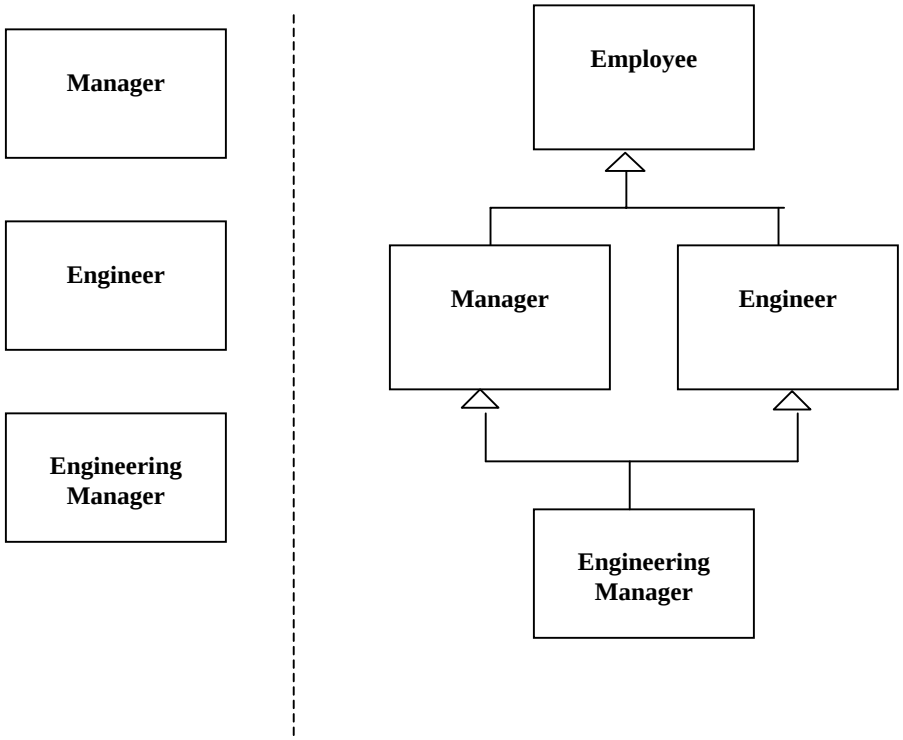

Figure 4: Two conceptual models with same contents

2.6 Summary on equivalence relations

Given a set Q of queries of interest, we define the concept of equivalence between alternate models through the equivalence relation is IC defined by:

IC={(S, S')| for all q in Q, q(S)=q(S')}

This relation is used to define valid transformations: i.e. transformations that do not affect the contents. In practice, IC is not always easy to compute and use. Other, more restrictive relations can be used. We construct alternate relations using a generator relation (root) P, EQ = P^*.

Requirements on EQ:

- EQ $\subseteq$ IC
- EQ is an equivalence relation

Construction of EQ:

1. Build a relation symmetric relation P.
2. Construct EQ as the reflexive transitive closure of P.

Validation of EQ:

1. Prove that P is symmetric.
2. Prove that P $\subseteq$ EQ.

Evaluation of EQ:

EQ is evaluated in terms of two criteria: completeness and simplicity.

- Completeness: EQ is complete if IC $\subseteq$ EQ, i.e. if R is a root of EQ.
- Simplicity: The smaller and the simpler is R, the easier it is to use.

In Mili et al., [17] we define criteria of comparing relations in terms of their relative completeness and simplicity. These criteria can be adapted and used in the context of interest.

3. MODELING QUALITY

Whereas the equivalence relation is used to characterize *valid* schemas transformations, ordering relations are used to characterize *useful* transformations. A transformation is valid if it preserves contents; a transformation is useful if it transforms one model into a "better" model. Also, whereas the equivalence relation is based on contents and is typically *user-oriented*, the preference relation is based on structure and is typically *designer motivated.* Software correctness for example is based on the external behavior of software, whereas software quality is focused on readability, understandability, maintainability, and other qualities of interest to the developer.

In sum, whereas the equivalence relation takes the form

EQ={(S, S')| S and S' capture the same information relevant to a set of queries Q},

the preference relations take the form

PS ={(S, S')| structure(S') is preferred to structure(S)}.

The two notions are therefore orthogonal. Modeling activities must be controlled by EQ and driven by PS. Whereas EQ is defined from the functionality of the resulting database, there is a wider range of variability when it comes to the preference relations. There are two common forms used to express preference relations:

- Canonical/normal forms: A syntactic or semantic constraint is used to define acceptable representations. Modeling consists of transforming an initial model to a canonical representative from the same equivalence class.
- Preference criteria: Criteria express preference relations (avoid multiple inheritance, prefer low coupling, etc.) Generally, more than one criterion is relevant to any modeling task. Modeling consists of transforming an initial model into one that "optimizes" the various preference relations and balances between them.

3.1 Requirements on preference relations

We consider the two forms of preference in turn.

3.1.1 Canonical form

Canonical forms identify a subset of the universe U of schemas that is deemed acceptable. They are distinguished from other preference criteria by the facts that they are binary (met or not met) and that they capture a mandatory (rather than desirable) property.

N={S| S is in normal form}

This set defines a two-level preference relation:

PS_N={(S, S')|S'$\in$N},

where PS_N identifies all pairs of repositories where the second one is in normal form. Of course PS_N is meant to be used along with EQ. We are interested in transformations

S$\rightarrow$ S'

such that (S, S') $\in$ EQ$\cap PS_N$. This leads us to the requirements on PS_N relative to EQ: Every schema must admit at least one equivalent normal schema, i.e.

EQ$\cap PS_N$ is a total relation or

Every equivalence class of EQ contains at least one element from N.

Violation of the above condition signals that either the canonical form is too restrictive and needs to be weakened, or the equivalence relation is too selective, and its equivalence classes need to be clustered, to the extent that that is admissible. We distinguish the two situations as follows:

- If $IC \cap PS_N$ is not total, then no EQ relation is consistent with PS_N. This happens when the normal form is restrictive. Horn clauses are an example of such normal forms. They are used with the understanding that they restrict the expressive power of the language, i.e. it is understood that some expressions cannot be transformed into equivalence Horn clauses.
- If $IC \cap PS_N$ is total but $EQ \cap PS_N$ is not, then it is possible to find an EQ relation consistent with PS_N . One could then either target EQ and make it into a larger relation with larger equivalence classes, or target PS_N as above.

The conjunctive normal form in predicate calculus and the relational normal forms in relational databases are prototypical examples of canonical forms. They both are admissible normal forms in the sense that any FOL expression can be transformed into an equivalent CNF expression, and any relational database schema can be transformed into an equivalent schema with all relations in normal form. On the other hand, they illustrate two different uses of normal forms:

- Conjunctive Normal Forms are a used mostly as a means to facilitate processing (e.g. compare two expressions for equality), and enable the use of some algorithms efficiently. There is not necessarily an inherent preference for CNF over other alternate canonical forms. The main motivator is to improve computational efficiency by reducing syntactic variability.
- Relational Normal Forms are used to minimize redundancy and update anomalies. As such they truly express a preference: In order to minimize redundancy, normalize the database.

Although the motivation is different, the underlying transformational process is the same: Given a model, transform it into another model that has equivalent semantics but is in canonical form.

3.1.2 Quality criteria

Quality criteria are the most common form of expression of structural quality. Quality criteria can be thought of as binary relations capturing the preference of some structures to others:

$PS_i = \{(S, S') | \text{structure (S) preferred to structure (S')}\}$,

although they generally are expressed in much more general terms such as:

- Increase the coherence of software modules, i.e. modules with higher coherence are preferred to modules with lower coherence.
- Decrease the coupling between different modules, i.e. designs with lower coupling are preferred to designs with higher coupling.
- Avoid multiple-inheritance by reducing the number of instances of multiple inheritance and by reducing the number of parents of any given class.
- Keep the size of modules within "readable" ranges. Keep the number of modules within reasonable ranges.
- Structure the software in a way that facilitates specific forms of evolution of the software.
- The criterion defined by Tari [23] for databases: "schema with less redundancy and fewer update anomalies is preferable."
- In [27], it is argued that symmetry is a desirable property of artifacts.

The above list illustrates both the motivation behind the criteria and their level of specificity. The motivators are the so-called "ility" properties of readability, updatability, maintainability, reusability, and so forth. On the other hand, the criteria can be formulated in very general terms such as "increase coherence and reduce coupling" or can be associated with very specific guidelines (no class should have more than 2 parents; no method should be longer than a page". Also, the preference relations may refer to different views (thus different structures) of the artifact considered. The only requirement in terms of admissible preference criteria is that each of the PS_i relations be a partial ordering relation, i.e. asymmetric and transitive. There is no requirement of global consistency because it is understood that the different criteria can be and often are conflicting. It is up to the modeler to balance between them.

The design patterns presented by Gamma et al [10] can be thought of as catalogues of structural patterns (transformations) motivated by preference criteria. For example, Façade, one of the patterns is summarized as follows (page 8):

"Façade: Provide a unified interface to a set of interfaces in a subsystem. Façade defines a higher-level interface that makes the subsystem easier to use."

The motivator here is the ease of use, and the preference criterion is uniformity of interfaces. The associated transformation is to create an abstract interface and have all modules within the same subsystem inherit their interfaces from it.

4. TRANSFORMATIONS

In this paper, we have identified two key aspects of transformations, namely, validity, and usefulness. A transformation is valid if and only if it transforms one model into an equivalent model; a transformation is useful if and only if it is consistent with at least one preference relation. By exhibiting these two components, we can then analyze, evaluate, and compare various proposals of modeling by transformations. This also provides us with a constructive approach towards developing sets of transformations in new domains.

When surveying the literature in modeling by transformations in the field of object-oriented databases, we find a surge of proposals of normal forms and quality criteria. These proposals almost invariably focus on the problem of schema evolution.

Schema evolution is the process by which,

1. Given an initial schema S that meets all structural qualities, including being in canonical form,
2. Given a contemplated change to the schema, that we call δS (e.g. add a class, drop an association, etc.),
3. Evaluate S'=S + δS to check that it meets structural qualities
4. In case it does not, transform S' into S" where S" is semantically equivalent to S' and meets all structural qualities.

In fact, the problem of schema evolution in Object Oriented context is further complicated by the fact that simple primitive alterations to OO database schemas can have a number of unforeseen consequences. These consequences need to be evaluated and addressed before concerns of structural quality. The modified S'=S + δS may violate, not only "cosmetic" structural properties, but also some of the basic syntactic invariants. For example, adding a new "address" to the class Student may result in two attributes with the same name within the same class. This syntactic issue must be resolved before proceeding. The problems arising with additions may be semantic as well. It is possible for the added information δS to be in conflict with the original information S, resulting in an inconsistent S'=S + δS. For example, the addition of a classification link may result in a cycle in the inheritance tree. Additional information must be sought to decide how to break the cycle. Because of these issues, the process of schema evolution is more complex than described above. On the other hand, most proposals address all issues within the same step. That is, the process looks like the following:

1. Given an initial schema S that meets all structural qualities, including being in canonical form,
2. Given a contemplated change δS,
3. Evaluate S'=S + δS to check that it is syntactically correct, consistent, and meets structural qualities
4. In case it does not, resolve conflicts, correct syntax, and generate a final S" that meets syntactic rules and structural qualities.

With the above process, authors have focused on identifying sets of schema invariants that may be violated by S', and proposed transformations to restore these invariants [2,4]. As a result, the invariants proposed include syntactic, semantic, as well as structural invariants and the transformations are not necessarily contents preserving. When they are used to resolve conflicts and disambiguate modification primitives, the transformations are by definition non contents preserving. Formica et al. [9] for example introduce schema Inheritance Normal Forms (INF) for object oriented databases and use it as an intermediate syntax to check absence of inheritance conflicts. Transformations are used to change the contents of the schema in order to eliminate conflicts when they are found. This insight helps explain the difficulty in comparing different formalisms and proposals. It also suggests an approach for bridging that difficulty: by separation of concerns. We need to distinguish between the two stages of modifying and disambiguating contents on the one hand, and modifying structure while preserving contents on the other hand. The second stage can only start when contents have become stable. This separation allows us to distinguish between syntactic schema invariants [2,4] semantic invariants [23] and structural invariants [15]. The requirements and motivations behind each set of invariants is different. The requirements behind their associated transformations are different as well.

5. SUMMARY AND CONCLUSIONS

The increasing number of Object Oriented databases and knowledge repositories raises the need for some criteria of model validity, modeling guidelines, and for quality criteria to be used in modeling. Although some researchers have developed some specialized sets of modeling guidelines, we found no general-purpose framework to evaluate these proposals and customize them. In this paper we develop a theoretical framework for characterizing modeling. Modeling is seen as a transformation process. These transformations must be valid, and must be increase the quality of the model. The framework is centered around these two issues of validity and quality.

The framework presented was motivated by our need to develop and maintain engineering knowledge repositories. We have opted to use the Object Oriented model for its expressive power and support for associations. We found that there is void when it comes to comprehensive definition of model verification and model quality monitoring. Shen [21] has developed a framework for defining syntactic modeling invariant (definition of well-formed models). Blackwell [3] has used the theoretical framework presented here to classify and compare the different modeling approaches presented in the literature. We are currently using this framework for defining equivalence relations, ordering relations, and transformations, for an object oriented model developed for engineering knowledge repositories [15].

REFERENCES

1. Baader, F. et al (Editors). *The Description Logic Handbook: Theory, Implementation and Applications*. Cambridge University Press, 2003.
2. Banerjee, J. et al. "Semantics and Implementation of schema evolution in Object-oriented Databases." *Proc. of ACM SIGMOD*. June 87. pp 311-322.
3. Blackwell, L. "Survey of the literature on repository's quality." Master's thesis, Oakland University, 2002.
4. Breche, P. "Advanced Primitives for Changing Schemas of Object Databases." *Proc. of CAISE 96*. Springer-Verlag. May 96
5. Devanbu, P., M.A. Jones. "The Use of Description Logics in KBSE Systems." *ACM Transactions on Software Engineering and Methodology*, 6 (2). April 1997. pp 141-172.
6. Disjkstra, E. W. A. "Constructive Approach to the Problem of Program Correctness." *BIT* 8, 3. 1968. pp 174-186.

7. Emam, K. E. et al. "The Prediction of Faulty Classes Using Object-Oriented Design Metrics." *The Journal of Systems and Software*, 56. 2001. pp 63-75.
8. Feather, M.S. "Rapid Application of Lightweight Formal Methods for Consistency Analyses." *IEEE Transactions on Software Engineering*, 24 (11). November 1998. pp 949-959.
9. Formica, A. "An Efficient Method for Checking Object Oriented Database Schema Correctness." *ACM TODS* (23) 3. September 1998. pp 333-369.
10. Gamma et al. *Design Patterns: Elements of reusable Object-Oriented Software*. Professional Computing Series, Addison Wesley. 1994.
11. Gil, Y., Tallis, M. "Transaction-Based Knowledge Acquisition: Complex Modifications Made Easier." *Proceedings of the Ninth Knowledge Acquisition for Knowledge-Based Systems Workshop*. 1995. Banff, Alberta, Canada.
12. Grundy, J. et al. "Inconsistency Management for Multiple-View Software Development Environments." *IEEE Transactions on Software Engineering*, 24 (11). November 1998. pp 960-981.
13. Heitmeyer, C. et al, "Using Abstraction and Model Checking to Detect Safety Violations in Requirements Specifications." *IEEE Transactions on Software Engineering* , 24 (11). November 1998. pp 927-948.
14. Jacobson, I. et al. *The Unified Software Development Process.* Addison Wesley, 1999.
15. Mili, F. *et al.* "Knowledge Modeling for Engineering Knowledge Repositories." *AI in Design 2001.*
16. Mili, F., Narayanan, K. "Theoretical Framework for Defining Validity and Quality in Modeling." *Technical Report SECS-MILI-03-1*. August 2003.
17. Mili, A., Desharnais, J., Mili, F. *Computer Program Construction*. Oxford University Press. 1993.
18. Pipino, L. et al. "Data Quality Assessment." *Communications of the ACM.* April 2002. pp 211-218.
19. Preece, A.. "COVERAGE: Verifying Multiple-Agent Knowledge-Based Systems." *Knowledge-Based Systems*, 12. 1999. pp 37-44.
20. Sands, D. "Total Correctness by Local Improvement in the Transformation of Functional Programs." *ACM TOPLS*, 18 (2). pp 175-324.
21. Shen, W. "Syntactic and Semantic Invariants in OO Models with constraints." Master's thesis, Oakland University, 2002.
22. Storey, V., Wang, R. "Modeling Quality Requirements in Conceptual Database Design." *Proceedings of the 1998 Conference on Information Quality*. October 1998. pp 64-87.
23. Tari, Z. et al. "Object Normal Forms and Dependency Constraints for Object-Oriented Schemata." *ACM Transactions on Database Systems*, 22 (4). December 97. pp 513-569.
24. van Lamsweerde, A. et al. "Managing Conflicts in Goal-Driven Requirements Engineering." *IEEE Transactions on Software Engineering*, 24 (11). November 1998. pp 908-926.
25. Wang. Y. et al. "Toward Data Quality: An Attribute-based Approach." *Decision Support System,* 13. 1995. pp 349-372.
26. Wick, C. "Knowledge Management and Leadership Opportunities for Technical Communicators." *Technical Communication*. Fourth Quarter 2000. pp 515-529.
27. Zhao, L., Coplein, J. O. "Symmetry in Class and Type Hierarchy." *Proceedings of the 40th Conference on Tools Pacific*, Australian Computer Society. 2002. pp 121-135.

APPENDIX

Summary of variables

S: Schema
Q: Set of queries q
U: Universal relation
R: Relation
A,B,C: Attributes
IC_Q: Identical contents relative to queries in Q
EQ: Equivalence relation
PS : Preference relation