Towards a Metrics Suite for Object-Relational Mappings

Stefan Holder[1,*], Jim Buchan[2], and Stephen G. MacDonell[2]

[1] Max-Planck-Institut für Informatik, Campus E1 4,
66123 Saarbrücken, Germany
sholder@mpi-inf.mpg.de
[2] School of Computing and Mathematical Sciences, Auckland University of Technology,
Private Bag 92006, Auckland 1142, New Zealand
{jbuchan,smacdone}@aut.ac.nz

Abstract. Object-relational (O/R) middleware is frequently used in practice to bridge the semantic gap (the 'impedance mismatch') between object-oriented application systems and relational database management systems (RDBMSs). If O/R middleware is employed, the object model needs to be linked to the relational schema. Following the so-called forward engineering approach, the developer is faced with the challenge of choosing from a variety of mapping strategies for class associations and inheritance relationships. These mapping strategies have different impacts on the characteristics of application systems, such as their performance or maintainability. Quantifying these mapping impacts via metrics is considered beneficial in the context of O/R mapping tools since such metrics enable an automated and differentiated consideration of O/R mapping strategies. In this paper, the foundation of a metrics suite for object-relational mappings and an initial set of metrics are presented.

Keywords: Object-oriented software development; Relational database; Impedance mismatch; Object-relational mapping; Software metrics.

1 Introduction

Applications designed using object-oriented (OO) principles often achieve persistence of the object model using a relational database management system (RDBMS). This introduces a so-called 'impedance mismatch' between an application based on an OO design paradigm and an RDBMS designed according to quite different principles from relational theory. Object-relational (O/R) middleware is often used to bridge this semantic gap by providing a mechanism to map the object model to relations in the database. This layered approach to O/R mapping using middleware achieves the design goal of loose coupling between application and relational schema, making it possible to change the relational schema without the need to also change the application source code.

If O/R middleware is employed in this way, the object model needs to be linked to the relational schema using the mapping mechanism offered by the O/R middleware. In the case where a developer creates the object model first and then creates the relational

[*] The work reported here was conducted primarily while the first author was at Auckland University of Technology on exchange from Fulda University of Applied Sciences, Germany.

R.-D. Kutsche and N. Milanovic (Eds.): MBSDI 2008, CCIS 8, pp. 43–54, 2008.

schema by mapping the object model to relations, the so-called forward engineering approach, the developer is faced with the challenge of choosing from a variety of mapping strategies for class associations and inheritance relationships. These mapping strategies have different impacts on the non-functional properties of application systems, such as performance and maintainability [9, 11].

The problem for the developer, then, becomes selecting the mapping strategy that best suits the desired non-functional requirement priorities for a particular application. There are a number of approaches to addressing this problem and there is some tool support to automate the selection and generation of the mapping in the O/R middleware. In [8], for example, a single inheritance mapping strategy is used and a fixed mapping strategy for one-to-one, one-to-many, and many-to-many associations is applied, respectively. This approach is straightforward; however, the different impacts of alternative mapping strategies are not considered.

Philippi [11] therefore suggests a model-driven generation of O/R mappings. In his approach, the developer defines mapping requirements in terms of quality trade-offs. The mapping tool then automatically selects mappings that fulfill the specified requirements based on general heuristics regarding the impacts of mapping strategies. However, mapping impacts strongly depend on concrete object schema characteristics such as inheritance depth and the number of class attributes [9, 13], properties not considered in [11]. Furthermore, while model-driven generation of O/R mappings is said to ease mapping specification for the developer, such an approach reduces the developer's control over the mapping process and may result in a sub-optimal mapping for a given application. The developer should therefore have the option to define and refine mappings manually when required.

We therefore suggest that concrete schema characteristics such as inheritance depth and the number of class attributes should be considered in the selection of O/R mappings. We propose an approach that incorporates schema characteristics into metrics that will provide more accurate and sensitive measures of the impacts of a given mapping specification on non-functional requirements. We further suggest the application of metrics for O/R mappings in order to give the developer sophisticated feedback on the impacts of the chosen mapping. The quantification of mapping impacts using metrics is considered beneficial in the context of O/R mapping tools since these metrics provide a semi-automated mechanism for a developer to evaluate the appropriateness of a selected mapping in terms of its likely impact on desired application non-functional requirements such as performance and maintainability. As its main contribution, this paper provides the foundation for a metrics suite for inheritance mappings and defines several initial metrics.

While it is feasible to measure the impact of both association and inheritance mapping strategies with O/R metrics, we focus here on metrics for inheritance mapping strategies, for two reasons. First, the three basic inheritance mapping strategies (described in the following section) are applicable to all inheritance relationships, so the developer will *always* be required to choose one of them for each inheritance relationship. This is in contrast to association mapping strategies, whose applicability depends on the cardinality of the association, and so sometimes there is only limited choice or even no choice. Second, the impacts of inheritance mapping strategies strongly depend on the characteristics of the object model, such as inheritance depth and number of attributes. Thus, the drivers for the application of O/R metrics for inheritance relationships are more compelling.

The remainder of this paper is structured as follows. In Section 2, basic strategies for mapping entire inheritance hierarchies are described. In Section 3, the semantics of mapping strategies for individual inheritance relationships are defined. The goals of measurement are set in Section 4 before an initial metrics suite for inheritance mappings is proposed in Section 5, based on the defined inheritance mapping semantics. In Section 6, the coverage of the proposed metrics suite is explained. Finally, a conclusion to this paper is given in Section 7.

2 Basic Strategies for Mapping Inheritance Hierarchies

In the literature, inheritance mapping strategies are generally described as being applicable to whole class hierarchies. These 'pure' mapping strategies are explained in this section using the notation suggested in [9].

2.1 One Class - One Table

Following the 'one class – one table' mapping strategy, there is a one-to-one mapping between classes and tables. The corresponding table of a class contains a relational field for each non-inherited class attribute. Thus, object persistence is achieved by distributing object data over multiple tables. In order to link these tables, all tables share the same primary key. In addition, the primary keys are also foreign keys that mimic the inheritance relationships of the object schema [9]. Each row in a table, then, maps to objects in that table's corresponding class and subclasses.

It can be seen that for this mapping strategy, only one table needs to be accessed in order to identify the objects that match the query criteria of polymorphic queries. Whereas a non-polymorphic query only returns the objects of the class against which the query is issued, a polymorphic query returns the objects of the specified class and its subclasses for which the query criteria match.

This mapping strategy is commonly recommended if the object model's changeability (i.e. ability to easily change) is of primary importance because new classes can be added easily, without the need to modify existing tables. A significant drawback of this mapping strategy, however, is that multiple joins are needed to assemble all attribute data of an object. Moreover, if no views are used, it is relatively difficult to formulate ad-hoc queries because multiple tables need to be accessed to retrieve all object data [9].

2.2 One Inheritance Tree – One Table

Following the 'one inheritance tree – one table' mapping strategy, all classes of an inheritance hierarchy are mapped to the same relational table. This mapping strategy requires an additional relational field in the shared table, which indicates the type of each row.

This mapping strategy offers the best performance for polymorphic queries and allows easy ad-hoc reporting since only a single table needs to be accessed [9]. The changeability of the object model is reduced compared to the 'one class – one table' mapping strategy, however, because any object schema modification forces a modification of the sole inheritance table, which may already contain data.

Finally, if objects are stored using this mapping strategy, all relational fields that are not needed to store an object must contain null values. This especially applies to

relational fields corresponding to attributes of subclasses since these fields are not used when storing objects from other subclasses.

2.3 One Inheritance Path – One Table

The 'one inheritance path - one table' mapping strategy only maps each *concrete* class to a table. Thereby, all inherited and non-inherited attributes of a class are mapped to the same table. Each table then only contains instances of its corresponding concrete class.

Under this mapping strategy non-polymorphic queries run as quickly as they do under the 'one inheritance tree – one table' mapping strategy because only one table needs to be accessed. In contrast, a polymorphic query against a class needs to access the table that corresponds to this class and all tables that correspond to its subclasses, which could result in a significant performance overhead. Moreover, this mapping strategy implies a duplication of relational fields, which results in multiple updates of these fields if the corresponding class attribute is changed.

The three inheritance mapping strategies just described are restrictive in that they are only applicable to entire inheritance hierarchies. In the next section, we introduce more finely-grained inheritance mapping strategies that can be used in combination on elements of an inheritance hierarchy in order to produce an optimal mapping.

3 Semantics of Mapping Strategies for Individual Inheritance Relationships

In practice, current O/R middleware products such as Hibernate [6] support the mixing of different inheritance mapping strategies for one inheritance hierarchy, thereby providing a finer level of granularity in the mapping strategy selection than the basic mapping strategies described in Section 2. However, the opportunity to mix inheritance mapping strategies means that the developer must decide what mix of mapping strategies will be optimum for a given object model structure. This requirement strengthens the likely usefulness of a set of mapping metrics that reflect this finer granularity. Hence, the ability to use such a suite of metrics to inform this decision, as proposed by this paper, should ease the developer's effort and result in a better quality decision.

Before discussing the development of these metrics a method of clearly representing the semantics of individual inheritance mapping strategies is needed. The following notation will be used and follows the inheritance mapping model definitions suggested by [4]. In the following definitions, P denotes the superclass (parent class) at the superclass-end of an example inheritance relationship while C denotes the subclass (child class) at the subclass-end of this inheritance relationship.

Union superclass: If U_C denotes the set of classes that are reachable from C (including C) via 'union superclass' inheritance relationships, the attributes defined by all classes in U_C are mapped to a table corresponding to the most general class in U_C. Moreover, a type indicator field is needed in this table in order to determine the class type of each row.

Joined subclass: The attributes defined by C as well as the primary key attributes inherited from P or another superclass are mapped to an own table T. The primary key fields of T contain a foreign key constraint to the same primary key fields in the table to which P is mapped.

Union subclass: If *C* is abstract and all inheritance relationships to its direct subclasses are mapped with 'union subclass' (or *C* does not have any subclasses), then no corresponding table is created for *C*. Otherwise, the attributes of *C* and the attributes of all superclasses of *C* are mapped to an own table.

Figure 1, adapted from an example in [4], shows a sample mapping that represents the above definitions. In it, white boxes denote classes and grey boxes denote tables. The mapping from classes to tables is indicated with black arrows and inheritance relationships (white arrows) are labeled with the mapping strategies applied to them.

Having introduced the semantics of mapping strategies for individual inheritance relationships, in the next section we describe the derivation of measurement goals relevant to the assessment of the impact of these mapping strategies.

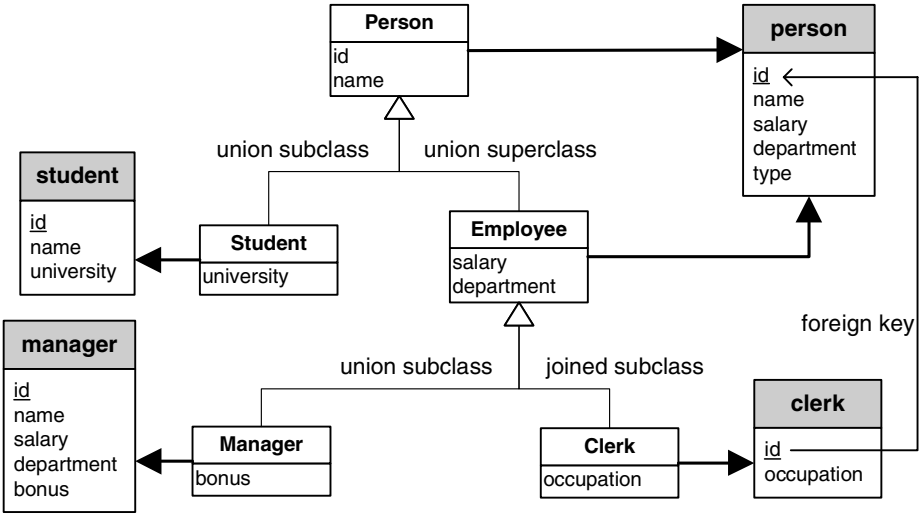

Fig. 1. Example of mixed inheritance mapping strategies

4 Measurement Goals

Bearing in mind that different mapping strategies have different impacts in terms of the non-functional characteristics of applications, our intent in this section is to identify measures that would be useful in guiding the developer's choice of mapping. In order to identify appropriate metrics, we follow a simplified variant of the commonly employed Goal/Question/Metric (GQM) approach [2, 3]. The GQM approach defines a framework for identifying metrics by defining goals, asking questions related to how these goals could be achieved and defining metrics that are intended to answer the posed questions.

In the first step of identifying relevant measurement goals, we consider software quality (non-functional) characteristics that are influenced by O/R mappings. The result of our top-down approach to identifying software quality characteristics for O/R mappings is depicted in Figure 2.

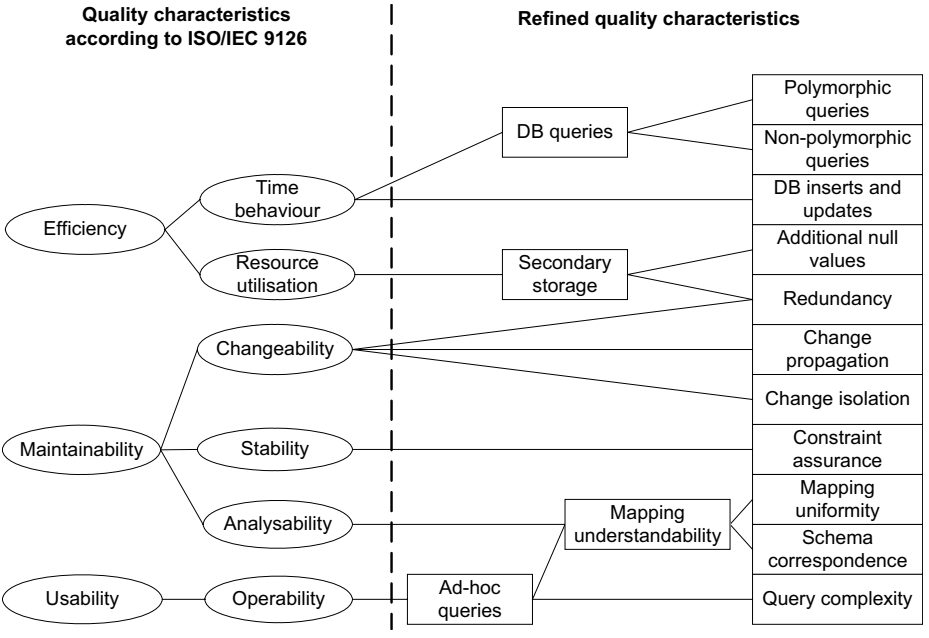

Fig. 2. Quality characteristics of O/R mappings

Table 1. Description of quality characteristics

Quality characteristic	Description
Time behavior of polymorphic database queries	Mainly depends on the number of tables that need to be accessed / the number of joins that need to be performed. Higher redundancy can improve the time behavior of polymorphic database queries but can negatively affect the time behavior of database inserts and updates [9].
Time behavior of non-polymorphic database queries	
Time behavior of database inserts and updates	
Additional null values	Null values that solely result from the applied mapping strategy
Redundancy	The degree of redundancy that is caused by the applied mapping strategy
Change propagation	Extent to which it is necessary to adapt the relational schema and the O/R mappings to changes in the object model
Change isolation	Depends on whether existing tables need to be modified for adding/deleting classes
Constraint assurance	Depends on the ability of the relational schema to enforce integrity constraints
Mapping uniformity	Uniformity of applied mapping strategies. Refers to the overall mapping and is not applicable to individual mapping strategies
Schema correspondence	Extent to which the object model resembles the relational schema
Query complexity	Effort required to formulate ad-hoc queries

This network of quality characteristics subsumes and extends the classifications of O/R mapping impacts suggested by [9, 11]. It starts with the quality characteristics efficiency, maintainability, and usability, which are a subset of the software quality characteristics defined by the ISO/IEC 9126-1 standard [7]. The other high-level software quality characteristics of the ISO/IEC 9126-1 standard – functionality, reliability, and portability – are not considered to be significantly influenced by O/R mappings, and so are not included here.

Efficiency, maintainability, and usability are then split into quality sub-characteristics, also defined by the ISO/IEC 9126-1 standard. Finally, these sub-characteristics are refined into specific quality characteristics relevant to O/R mappings. In Section 5, we propose metrics to measure these specific quality characteristics.

In Table 1, the specific quality characteristics are listed and explained. This builds on the works of Keller [9] and Philippi [11] by considering additional characteristics, namely Redundancy, Change isolation, Constraint assurance, and Mapping uniformity, and we further extend their work by proposing metrics for some of these. The next section describes and justifies the metrics developed for these quality characteristics and provides examples of their use.

5 Metrics for Inheritance Mappings

Metrics have been suggested for object-oriented design [5] and for relational database systems [12] as well as for object-relational database systems [1]. These metrics, while useful, are considered insufficient for measuring the impacts of O/R mappings for two reasons. First, the available metrics for relational schemas do not sufficiently cover the suggested network of quality characteristics (see Section 4). Second, the metrics focus on either object-oriented design or relational schemas but not on the mapping between them. Therefore, the metrics suite suggested here, comprising four metrics at the level of individual classes, is complementary to those described elsewhere, in that it explicitly addresses the measurement of O/R mappings.

5.1 Table Accesses for Type Identification (TATI)

Polymorphic queries against a class *C* return objects whose most specific class is *C* or one of its subclasses. Before an object can be completely retrieved, it is necessary to identify the most specific class of this object. It should be noted that identifying the most specific class is equivalent to identifying the tables that need to be queried in order to retrieve the requested object. In contrast, for non-polymorphic queries, the most specific class of the requested object is the same class against which the query is issued.

There are two strategies to identify the most specific class: either each possible table is queried individually and the search is stopped as soon as the most specific class is identified, or all possible tables are queried with one query. While the former strategy means that the search is completed as soon as the most specific class is found, the latter strategy allows the database system to query the tables in parallel. The latter strategy can be accomplished by sending multiple queries to the database at the same time or by using the SQL UNION clause. The maximum number of table accesses measured with this metric is therefore only relevant if the former strategy is applied.

Definition: For queries that are issued against a class *C*, TATI(*C*) equals the maximum number of tables that have to be accessed in order to identify the most specific class of the requested object.

The maximum number of tables that need to be accessed for a query issued against a class *C* equals the number of different tables that correspond to *C* and all of its subclasses. In Figure 1, TATI(*Person*) = 4 since *Person* is the root class of the inheritance hierarchy and the inheritance hierarchy is mapped to 4 tables altogether. Similarly, TATI(*Employee*) = 3 since *Employee* and its subclasses *Manager* and *Clerk* are mapped to the 3 tables *person*, *manager*, and *clerk*.

5.2 Number of Corresponding Tables (NCT)

In contrast to polymorphic queries, the most specific class of a queried object in a non-polymorphic query is known. Therefore, no queries are necessary in this case to determine the most specific class. The performance of non-polymorphic queries therefore mainly depends on the number of tables that contain data of the requested object. For polymorphic queries, it is possible to retrieve the complete object data while querying tables in order to identify the most specific class of the requested object.

We therefore propose the metric NCT(*C*), which equals the number of tables that contain data from instances of a class *C*. This number depends on the inheritance mapping strategies that are used for the inheritance relationships on the path of class *C* to the root class of the inheritance hierarchy. In particular, the application of the 'joined subclass' strategy results in increasing values of NCT.

As already indicated, this metric is a measure of object retrieval performance. In addition, this metric is a measure of query complexity in the context of ad-hoc queries since we consider the number of involved tables to be an appropriate measure of the user's effort in formulating a query. However, using the number of tables to measure ad-hoc queries assumes that no views are employed to ease query formulation.

Definition: NCT is formally defined by equation (1), where the function p(*C*) returns the direct superclass of *C*.

$$NCT(C) = \begin{cases} 1 & \text{if } C \text{ is the root class or} \\ & \uparrow(\mathrm{p}(C), C) \text{ is mapped via 'union subclass'} \\ NCT(p(C)) & \text{if } C \text{ is mapped to the same table as its superclass} \\ NCT(p(C)) + 1 & \text{if } C \text{ is mapped to its own table} \end{cases} \quad (1)$$

In Figure 1, NCT(*Clerk*) = 2 because the tables *clerk* and *person* contain data that are necessary to assemble objects of *Clerk*. In contrast NCT(*Manager*) = 1, as all defined and inherited attributes of the class *Manager* are mapped to relational fields of the table *manager*.

5.3 Number of Corresponding Relational Fields (NCRF)

The Number of Corresponding Relational Fields (NCRF) gives a measure for the degree of change propagation for a given O/R mapping. More specifically, this metric reflects the effort required to adapt the relational schema after inserting, modifying, or deleting a class attribute. This effort is mainly influenced by the application of the 'union subclass'

mapping strategy since applying this mapping strategy typically results in the duplication of relational fields (see Section 2.3). Because of these duplications, changes in the object model result in multiple changes to the relational schema. In contrast, the duplication of relational fields does not occur when the 'joined subclass' or the 'union superclass' mapping strategies are applied. (Note: primary key fields are not considered by this metric because they should be resistant to changes.)

Definition: For a class *C*, NCRF(*C*) equals the number of relational fields in all tables that correspond to each non-inherited non-key attribute of *C*. If *C* does not have any non-inherited non-key class attributes, NCRF(*C*) equals the number of relational fields to which each non-inherited non-key class attribute of *C would* be mapped.

In Figure 1, NCRF(Person) = 3 since the class attribute *Person.name* is mapped to the three relational fields *person.name*, *student.name*, and *manager.name*. NCRF(*Employee*) = 2 since each of the attributes *Employee.salary* and *Employee.department* is mapped to one relational field in the table *person* and one relational field in the table *manager*. Finally, NCRF(*Student*) = NCRF(*Manager*) = NCRF(*Clerk*) = 1 since the only non-inherited class attribute of each of these three classes (*Student.university*, *Manager.bonus*, *Clerk.occupation*) is mapped to 1 relational field.

5.4 Additional Null Values (ANV)

ANV measures additional storage space in terms of null values that result when different classes are stored together in the same table using the 'union superclass' mapping strategy (see Section 2.2).

For a definition of ANV(*C*), the following is considered. Let A_C be the set of non-inherited attributes of class *C* and let F_C be the set of corresponding relational fields in the shared table. Applied to a particular class *C,* the aim of ANV(*C*) is to give a measure for the number of null values that occur at the relational fields F_C. An important observation is that null values at the relational fields F_C occur if and only if instances of classes different from *C* and different from subclasses of *C* are stored in the shared table. More precisely, if instances of two distinct classes *B* and *C* are stored together in a shared table and *B* is not a subclass of *C,* then each row in the shared table that represents an instance of *B* contains a null value at each relational field in F_C. ANV(*C*) is therefore higher the more classes different from *C* and different from subclasses of *C* are mapped to the same table as *C*.

The number of null values depends on the number of instances of each class, something that may be unknown at the stage of mapping specification. In order to give an approximation of additional null values, it is assumed that there is the same number of instances for all (concrete) classes. Furthermore, ANV is normalized by assuming that there is only one instance of each class. This assumption also ensures that ANV only depends on a given object model and is thus in line with the previously described metrics. Note, however, that this metric could easily be generalized to take the number of instances per class into account if this is known.

Definition: ANV(*C*) equals the number of non-inherited attributes in *C* multiplied by the number of concrete classes that are mapped to *T*, excluding *C* and all of its subclasses.

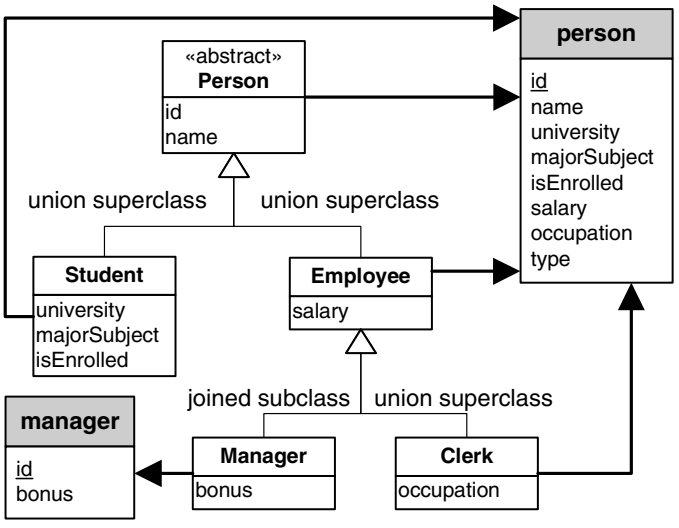

Fig. 3. Example mapping to illustrate the ANV metric

For the example mapping shown in Figure 3, ANV(*Student*) is calculated as follows. Concrete classes that are mapped to the same table as *Student* are *Employee* and *Clerk*. These two classes do not inherit the attributes declared by *Student*; therefore, additional null values are contained by rows that correspond to instances of *Employee* and *Clerk*. If one instance for each class was stored, in total there would be $2 \cdot 3 = 6$ null values in the rows of the table *person* at the fields *university*, *majorSubject*, and *isEnrolled*. Thus, ANV(*Student*) = 6.

6 Metric Coverage

Table 2 shows the coverage of the defined quality characteristics by the proposed metrics. This table shows that more metrics are needed in order to more fully measure the relevant quality characteristics of mapping strategies.

Table 2. Metric coverage for inheritance mapping strategies

Quality characteristic	TATI	NCT	NCRF	ANV
Polymorphic queries	X	X		
Non-polymorphic queries		X		
DB inserts and updates		X		
Additional null values				X
Change propagation			X	
Change isolation				
Constraint assurance				
Mapping uniformity				
Schema correspondence				
Query complexity		X		

The quality characteristic Redundancy is not included in the table since this characteristic is only applicable to association mapping strategies and not to inheritance mapping strategies [10]. It should be noted that although the 'one class – one inheritance tree' mapping strategy leads to a duplication of relational fields, it does not imply redundancy.

7 Conclusions and Further Research

The application of metrics in O/R mapping tools provides significant potential for supporting the developer in the considerably difficult task of mapping specification. Developers would benefit from a facility that supports the manual specification of O/R mappings by giving feedback on the impacts of these mapping strategies. Since the impacts of mapping strategies strongly depend on the concrete characteristics of the object model, metrics are considered an appropriate means to convey these schema characteristics. Moreover, the adoption of O/R metrics in model-driven generation of O/R mappings should enable a more appropriate selection of mapping strategies that leads to better fulfillment of non-functional requirements. As mapping impacts differ particularly for inheritance mappings, we have focused on developing a set of metrics for inheritance mapping strategies. This metrics suite is based on a novel inheritance mapping model that supports the mixing of inheritance mapping strategies in inheritance hierarchies.

We plan to empirically evaluate the suggested metrics in terms of their utility in giving feedback about the impacts of mapping strategies to developers. We will extend this further by investigating algorithms that automatically determine the appropriate mapping strategy based on the requirements of the developer. To achieve this goal, normalization of the metrics will be required to ensure that metric values can be weighted and compared appropriately.

Finally, as object-relational database management systems (ORDBMSs) become increasingly prevalent, support for mapping from object-oriented programming languages to ORDBMS-schema also becomes more important. We will therefore further investigate how mapping specification for ORDBMS-schemas can be supported by the application of metrics.

References

1. Baroni, A.L., Calero, C., Piattini, M., Abreu, F.B.: A Formal Definition for Object-Relational Database Metrics. In: 7th International Conference on Enterprise Information Systems (ICEIS 2005), Miami, pp. 334–339 (2005)
2. Basili, V.R., Caldiera, G., Rombach, H.D.: The Goal Question Metric approach. Encyclopedia of Software Engineering 1, 528–532 (1994)
3. Basili, V.R., Rombach, H.D.: The TAME project: Towards Improvement-Oriented Software Environments. IEEE Transactions on Software Engineering 14(6), 758–773 (1988)
4. Cabibbo, L., Carosi, A.: Managing Inheritance Hierarchies in Object/Relational Mapping Tools. In: Pastor, Ó., Falcão e Cunha, J. (eds.) CAiSE 2005. LNCS, vol. 3520, pp. 135–150. Springer, Heidelberg (2005)

5. Chidamber, S.R., Kemerer, C.F.: A Metrics Suite for Object Oriented design. IEEE Transactions on Software Engineering 20(6), 476–493 (1994)
6. Hibernate Open Source Software, http://www.hibernate.org
7. ISO/IEC 9126-1:2001 Software Engineering - Product Quality - Part 1: Quality Model, http://www.iso.org
8. Keller, A.M., Jensen, R., Keene, C.: Persistence Software: Bridging Object-Oriented Programming and Relational Databases. In: Proc. of the 1993 ACM SIGMOD Int. Conf. on Management of Data, Washington, D.C, pp. 523–528 (1993)
9. Keller, W.: Mapping Objects to Tables: A Pattern Language. In: Proc. of the 1997 European Conf. on Pattern Languages of Programming (EuroPLoP 1997), Irrsee, Germany (1997)
10. Oertly, F., Schiller, G.: Evolutionary database design. In: 5th International Conference on Data Engineering (ICDE), Los Angeles, pp. 618–624 (1989)
11. Philippi, S.: Model Driven Generation and Testing of Object-Relational Mappings. Journal of Systems and Software 77(2), 193–207 (2005)
12. Piattini, M., Calero, C., Genero, M.: Table Oriented Metrics for Relational Databases. Software Quality Journal 9(2), 79–97 (2001)
13. Rumbaugh, J., Blaha, M.R., Premerlani, W., Eddy, F., Lorensen, W.: Object-Oriented Modelling and Design. Prentice-Hall, Englewood Cliffs (1991)