Object-Relational Mapping Reconsidered

A Quantitative Study on the Impact of Database Technology on O/R Mapping Strategies

Martin Lorenz
martin.lorenz@hpi.de

Günter Hesse
guenter.hesse@hpi.de

Jan-Peer Rudloph
jan-peer.rudloph@hpi.de

Matthias Uflacker
matthias.uflacker@hpi.de

Hasso Plattner
hasso.plattner@hpi.de

Hasso Plattner Institute, University of Potsdam
August-Bebel-Str. 88
Potsdam, Germany

Abstract

Object-oriented applications often achieve persistence by using relational database systems. In such setup, object-relational mapping is used to link objects to tables. Due to fundamental differences between object-orientation and relational algebra, the definition of a mapping is a considerably difficult task. Today, there are only informal guidelines that support engineers in choosing the best mapping strategy. However, guidelines do not provide a quantification of actual impact and trade-off between different strategies. Thus, the decision on which mapping strategy should be implemented relies on a large portion of gut feeling.

In this paper, we propose a framework and conduct a quantitative study of the impact of object-relational mapping strategies on selected non-functional system characteristics. Our study creates awareness for consequences of using different mapping designs and persistence technologies. This allows developers to make distinctive and informed decisions, based on quantified results rather than gut feeling.

1 Introduction

Enterprise applications are often developed using object-oriented (OO) technologies and achieve persistence using relational database management systems (RDBMS). Modern applications incorporate so called object-relational (O/R) mapping layer or O/R middleware (ORM) that manage the linking between objects and tables. The main purpose of such a layer is to automate the mapping process and to create an abstraction that hides technical details of the mapping implementation. This includes mapping of data types and relationships. To some extent, this mapping process is fairly straight forward. However, there is one type of relationship that requires special consideration - object-oriented class inheritance, also known as generalization or sub-typing.

Whereas association and aggregation can be mapped directly to relational concepts, e.g., foreign keys, the mapping of inheritance is more complex [7]. Class inheritance is not a concept known to RDBMS. Hence, there exists no conceptual pendant in a relational data model that reflects semantics of an inheritance relationship. Instead, there exist different strategies, which can be used to implement the semantic of an inheritance relationship. These strategies have been defined and documented in a number of publications ([4, 12, 15, 13]) and are widely accepted. In their essence, they differ in the number, structure, and relationship of tables that are used to store objects of the domain model. Depending on the chosen strategy, an application has varying non-functional system characteristics [13], e.g., efficiency, usability, and maintainability. Today, there is no cost model, method, or tool chain, which helps software engineers to make an informed decision for or against a particular mapping strategy. Instead, they can only base their decision on informal guidelines. However, guidelines do not provide quantification of the impact and trade-off between mapping strategies. This is problematic for two reasons.

First, missing quantification neither allows anticipating the consequences of a mapping decision, nor does it allow effectively comparing mapping strategies for a given scenario. Second, guidelines do not take different database technologies into account. Especially in the enterprise information systems domain, the database market becomes increasingly diverse [21]. Apart from traditional disk-based row-oriented databases, specialized database systems increase in market share. One class of RDBMS that gets increasing attention are in-memory column-stores. SAP, one of the largest vendors of enterprise software, is moving their entire business suite onto an in-memory column-store platform [20]. Performance characteristics of such RDBMS are significantly different from classical disk-based row-stores [3]. The most interesting characteristics include, but are not limited to, read and write performance, memory consumption, join and scan processing. Since mapping strategies differ primarily in data model structure, these database characteristics have a direct impact on the characteristics of the mapping strategy, the O/R middleware and consequently the application. Given the informal nature of guidelines and their inability to quantify the impact of mapping strategies makes the decision making process error-prone and non-transparent. Consequently, the decision on which mapping strategy should be implemented relies on a large portion of gut feeling and is not based on quantified facts.

In this paper, we present a quantitative research study to make the impact of different O/R mapping strategies on non-functional system characteristics transparent. We propose a framework that allows to quantify the impact of a

URI: http://hdl.handle.net/10125/41754
ISBN: 978-0-9981331-0-2

mapping strategy, which provides a degree of accuracy and comparability that cannot be achieved with guidelines. It will be applicable for all types of databases with an SQL interface. A prototypical implementation of the framework is available online and our code is published on GitHub. Experiments presented in this paper can be reproduced using this prototype. Furthermore, we would encourage the use of our framework in industry projects to validate design decisions in the context of O/R middleware implementations.

The remainder of the paper is structured as follows: Section 2 describes the background of our work and motivates the topic. Section 3 gives a brief introduction to mapping strategy semantics. Section 4 describes the methodology the paper is based on and elaborates on the design of our framework. Sections 5 and 6 present a study and evaluation of a sample inheritance hierarchy. Section 7 discusses threats to validity of our framework design and measurements. Section 8 introduces related work and Section 9 presents conclusion and future work.

2 Is this really a Problem?

During our research, we have been confronted with the statement that the problem of object-relational mapping has been solved. To some extent, we agree with that. Especially mapping semantics, implementation patterns, and solutions for mapping process automation have been introduced and seem to be accepted by industry. However, we see two reasons why we believe this topic is worth investigating.

2.1 Missing Awareness

Looking at publications which discuss the topic of object-relational mapping strategies, we recognize a missing awareness of how large the impact and trade-off between strategies actually is. Relevant literature confines definitions of mapping strategies to implementation patterns and best practices [4, 12, 15, 14, 6]. None of these publications sheds light on the question to which degree a mapping strategy impacts non-functional system requirements. A first attempt to provide a structured approach to effectively analyze and compare mapping strategies has been introduced by Holder et. al. in [13]. Based on the ISO/IEC standard for software quality, they define a mapping between high-level non-functional (quality) software system characteristics (efficiency, usability, maintainability) and O/R mapping layer aspects (object creation, object retrieval, query complexity, etc.). Unfortunately, their work focuses on the definition of the mapping rather than on the definition of metrics to quantify the impact of mapping layer implementation, i.e., mapping strategies. However, Holder et. al. reveal the structure and complexity that have to be considered when proposing solutions to the problem of choosing an appropriate O/R mapping layer design.

ORM frameworks such as Hibernate[1] do a great job at hiding the mapping process complexity and provide process automation. However, such tools do not provide automated selection of inheritance mapping strategies. Regardless of whether using an ORM framework or implementing custom object-relational mapping logic is chosen, the decision for a mapping strategy has to be made by the developer. But how does a developer know what consequences the selection has? What part of the application is impacted by a particular mapping strategy? What is the trade-off to an alternative mapping? How big is the penalty of making a wrong choice? None of these questions can be answered sufficiently using state-of-the-art best practices, guidelines or ORM tools.

We see that there is a missing awareness of the impact of choosing a mapping strategy. Based on this observation, we think that this missing awareness and the abstraction of the problem by using ORMs constitute the general opinion that the problem of mapping strategy selection is solved.

This paper presents the results of a quantitative research study with the goal of making the impact of different mapping strategies transparent. To provide this transparency we propose a framework that extends the approach presented by Holder et. al. For every extension point we define metrics, which can be used to quantify the impact of a mapping strategy on various non-functional system characteristics. With this framework it is possible to analyze and compare different mapping strategies based on quantifiable metrics in order to create awareness. We show that the penalty of choosing a wrong mapping strategy can result in a performance decrease up to an order of magnitude.

2.2 A New Dimension to the Problem

In enterprise context, an increasing demand in flexibility, specialization and complex ad hoc analytics allows new specialized database systems to grab market share from traditional disk-based row-store systems [21]. In-memory column-stores are one particularly interesting type of RDBMS. SAP's next generation enterprise platform is running on SAP HANA[1], an example for such an in-memory column store [20]. The combination of in-memory computing, column-orientation, compression, and multi-core processing induces performance and memory consumption characteristics that are quite different from disk-based row-stores. Consequently, the performance profile of an application is impacted. Object-relational mapping proofs to be a suitable concept, where this impact can be observed. In this paper, we demonstrate that database technology has a major influence on characteristics of mapping strategies and O/R middleware. Consequently, developers who design and configure O/R middleware will make wrong decisions if they rely on experiences with traditional disk-based row-stores.

3 Mapping Strategy Semantics

In this section, we briefly explain the semantics of existing O/R mapping strategies. We start by explaining *Single Table Inheritance* approach in Section 3.1, followed by *Class Table Inheritance* and *Concrete Class Inheritance* approaches in Sections 3.2 and 3.3, respectively. For a more elaborate introduction to mapping strategy semantics, we refer to [4, 12]. To exemplify our descriptions, we introduce the sample hierarchy depicted in Figure 1.

3.1 Single Table Inheritance

The *Single Table Inheritance (ST)* approach is the simplest form of mapping an inheritance hierarchy to a relational

[1]http://hibernate.org

[1]https://hana.sap.com/abouthana.html

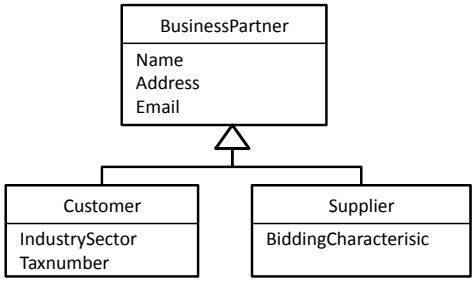

Figure 1: Class Inheritance Example

database. This approach uses one table to store all classes of a hierarchy. Every member variable of a class that is part of the hierarchy has to be represented by a column in that table. Additionally, we need a column to determine the actual type of the record, i.e., to define what class it belongs to. Figure 2 depicts the data model corresponding to the example from Figure 1.

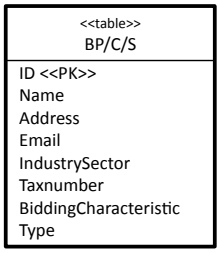

Figure 2: Database Schema for *ST* Approach

3.2 Table per Class Inheritance

The *Table per Class Inheritance (TPC)* approach is conceptually the mapping that is closest to the inheritance structure of the actual class model. As the name implies, it defines one table for every class in the hierarchy. The inheritance relationship is represented by foreign keys constraints (indicated as FK in Figure 3 and 4). Each table only provides attribute columns for member variables introduced by the class that corresponds to that table. Variations of that strategy propose to introduce an additional column, indicating the type of the class in every table (similar to the *Single Table Inheritance* approach). That way it is possible to determine the corresponding class of a record by looking at the type column. However, this is more or less a performance tweak for some types of queries. Since inheritance conceptually implies an "is-a" relationship, it is a matter of argumentation and data access pattern, whether or not to introduce the type column. Figure 3 depicts the database schema corresponding to the example in Figure 1.

3.3 Table per Concrete Class Inheritance

The *Table per Concrete Class Inheritance (TPCC)* approach suppresses the concept of inheritance structures and treats each member of a hierarchy as autonomous classes. Thus, the data model defined by this approach introduces a table

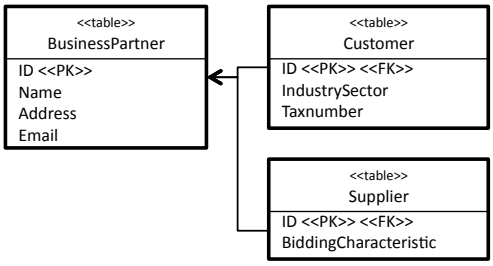

Figure 3: Database Schema for *TPC* Approach

for every concrete class in the hierarchy. Each table provides columns for all member variables of the corresponding class as well as for all inherited members. Figure 4 depicts the database schema corresponding to the example shown in Figure 1.

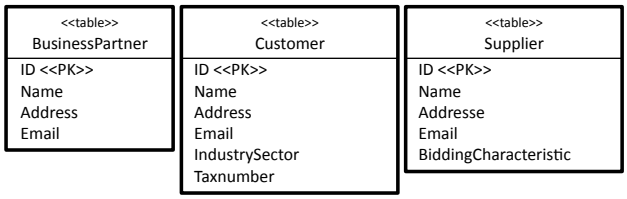

Figure 4: Database Schema for *TPCC* Approach

4 Methodology

The aim of this paper is to propose a quantitative approach for measuring the impact of object-relational mapping strategies on non-functional system characteristics. Thus, we provide transparency and create awareness for the impact and trade-off between different mapping strategy alternatives. The quantitative approach should enable developers to make more informed decisions regarding design and configuration of O/R middleware solutions. In the remainder of this paper we answer the following questions.

1. Q1: What needs to be measured to quantify the impact of an O/R mapping strategy on non-functional system characteristics?
2. Q2: How can one measure, present and compare an O/R mapping strategy's impact on non-functional system characteristics?
3. Q3: Does our framework provide enough information to effectively compare O/R mapping strategies?
4. Q4: Does database technology have an impact on characteristics of O/R mapping strategies?

To answer these questions, we use the following methodology. First, we introduce measurable non-functional system characteristics, which are influenced by the mapping strategy. Second, we describe idea, architecture, and prototypical implementation of a framework that allows analyzing object models and database technologies to quantify their impact on non-functional system characteristics. Finally, we define the study subject, which is used for measuring and evaluation. Each intermediate step of our methodology is described in detail in Sections 4.1 through 4.3.

4.1 Non-functional Characteristics

The primary driver for the development of a software system are functional requirements. Architecture and design decisions for the implementation of this functionality are driven by non-functional requirements. Selecting an object-relational mapping strategy impacts non-functional characteristics. Consequently, an engineer needs to understand the impact of a design decision in order to conform to the desired overall system characteristics. Holder et al. [13] propose an organization of non-functional characteristics which are influenced by O/R mapping strategies. Their organization is derived from the ISO/IEC standard for software quality, ISO/IEC 9126 [1]. Starting from the high-level characteristics efficiency, maintainability, and usability, they derive more concrete quality characteristics. Figure 5 depicts their organization. We see the definition of quality characteris-

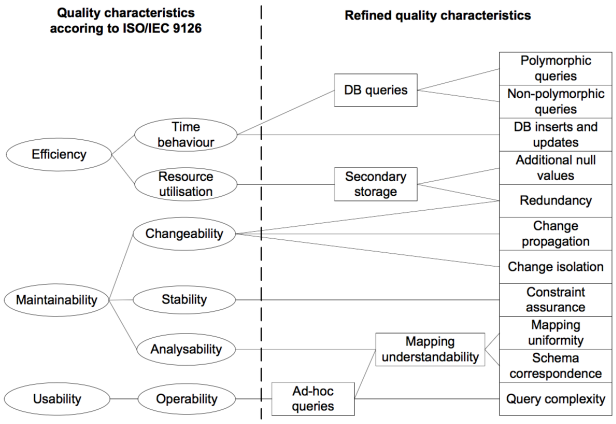

Figure 5: O/R Quality Characteristics defined by Holder et al. [13]

tics as an important step towards measurable aspects of a mapping strategy. However, we propose further refinement of these characteristics to provide measurable metrics. Sections 4.1.1 through 4.1.5 elaborate on each of Holder et al.'s quality characteristics. We discuss their idea and propose refinements where we see the necessity. Q1 is answered in these sections.

4.1.1 DB inserts and updates. This characteristic aims at capturing a strategy's ability to store new objects in the database and to update existing ones. Surprisingly, Holder et al. do not consider the retrieval and deletion of objects. We propose to add the two operations *Lookup* (read single object using identifier) and *Delete* (remove object using identifier). With that, all operations of the well known CRUD (Create, Read, Update, Delete) pattern are considered. The metric used to measure these aspects is query execution time.

4.1.2 Polymorphic and non-polymorphic queries. The aspect of polymorphic queries is derived from the polymorphic nature of classes in an inheritance hierarchy. Polymorphism is induced by the semantic of an "is-a"-relationship, i.e., each specialization is also an instance of all of its parent classes. Looking at our example from Figure 1, every *Customer* is a *BusinessPartner*. In the context of a database query, this semantic needs to be considered. A polymorphic query includes all database records of all specializations of the class the query was executed on. A non-polymorphic query is only restricted to records of one particular class.

We consider this characteristic of high importance, because it addresses the most dominant interaction pattern in enterprise workloads. Krüger et al. [16] have analyzed database workloads from different enterprise application systems. Their findings show that close to 90% of all database queries are read operations. These operations are triggered by tasks like the presentation of filtered object lists, or ad hoc analytics. With regards to these findings, we propose more diverse considerations of (non-)polymorphic queries. It is unrealistic to define and implement metrics that capture the efficiency of all possible types of database queries. However, we propose to distinguish between so-called range selects and table scans. The semantic of these operations is very similar. Both are set operations, which return a number of records that apply to some filter criterion. The only difference is that range select queries filter on an indexed attribute and table scans on non-indexed attributes. These two query types can be seen as atomic parts for any complex database query, i.e. aggregations or joins. Consequently, they can be used as a rough estimate on how different set operations would perform on the selected mapping strategy. The metric used to measure these aspects is query execution time.

4.1.3 Additional null values. Null values are only an issue for the *ST* strategy, because all objects are stored in the same table. In the end, the motivation to investigate the impact of null values is to understand the strategy's efficiency in terms of memory consumption. That is why we consider memory consumption instead of null values for our quantitative study. Thus, we are able to effectively compare all three strategies. To measure memory consumption, we aggregate the size of all tables that are used to store objects of the inheritance hierarchy. The metric used to measure these aspects is utilized memory space.

4.1.4 Change propagation. This characteristic reflects the complexity induced by a strategy to adapt the relational schema to changes in the object model. An example for such an adaptation is the change of an attribute's data type, the adding of a completely new attribute, or the deletion of an existing one. Holder et al. confined their investigation to the change of complexity, i.e., how many tables have to be touched in order to adapt the data model to the object model. We believe that this is not enough, because the refactoring of tables reflects only one aspect of change propagation. We want to include an investigation on the execution efficiency of such a change, i.e., how efficient can the database execute the needed operations to refactor the data schema. Especially with a changing database technology (data layout, row- vs. column) we expect significant differences in performance. The metric used to measure these aspects is query execution time.

4.1.5 Change isolation. This characteristic analyzes a strategy's ability to isolate changes that result from adding or deleting classes. It is very similar to change propagation, but the resulting changes are a lot more complex and invasive to the data schema. Change isolation is very close

related to the issue of database schema evolution, which is a research topic by itself. Apart from performance aspects, schema evolution discusses many issues that are difficult or impossible to measure, e.g., database governance, schema ownership, etc. Since our goal is to propose an empirical approach, we are looking for measurable system aspects. That is why we decided to look at execution time of data definition statements (e.g., CREATE, ALTER, and DELETE TABLE ...), which need to be executed to implement the changes induced by changing the object model. We decided to use this metric, because it reflects the database's ability to perform data rearrangements that result from changing the data schema.

Constraint assurance, mapping uniformity, schema correspondence, and query complexity are conceptual characteristic and do not need to be measured. Thus, we do not consider them for our study.

4.2 Framework

This section describes the architecture and design of our proposed framework. We start with an abstract definition of the idea of a framework. Later on, we describe the prototypical implementation of our proposed framework, which we provide as open source code on GitHub under `https://github.com/MartyLore/Mapping-Strategy-Analyzer`. It can be used to reproduce the measurements presented in this paper.

4.2.1 Architecture proposal. This section answers Q2. The basis for our proposal are the considerations of Section 4.1. The idea is to encapsulate each of the mapping aspects in dedicated test cases that can be executed, evaluated, and displayed automatically for a given object model. Figure 6 depicts the framework idea. Central part of our framework is a configuration file, which reflects the object model. Configuration options include structural aspects such as classes and specialization relationships between classes. Every class is defined by a *key*, *className*, an attribute called *instanceCount*, and a list of *fields*, which represents member variables of a class. The *instanceCount* attribute is used for configuring the number of objects of a class. Every field of a class is defined by *attName*, and *attType*.

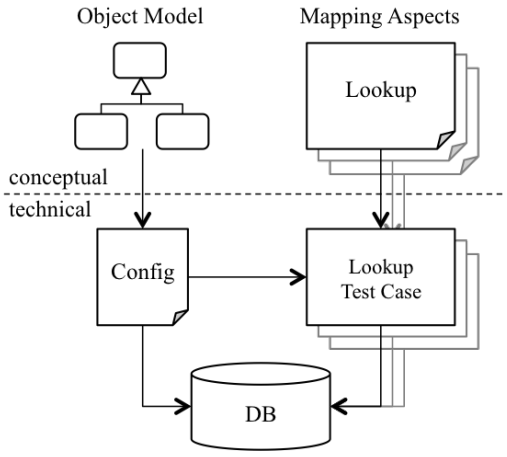

Figure 6: Framework Proposal

Based on this configuration, it is possible to implement logic that constructs SQL statements, which generate data models that correspond to the mapping strategy semantics explained in Section 3. Subsequently, the data model needs to be populated with sample data. The logic to generate sample data takes the configuration as input and produces SQL statements, which insert data into the data models. At that point, the framework automatically produced three data models that correspond to the three mapping strategy semantics introduced in Section 3 and populated them with identical data. Based on these three data models, it is now possible to quantify mapping strategy aspects and compare their results.

As mentioned in the beginning, each mapping strategy aspect is implemented in a separate test case. Each test case consists of a preparation, an execution, and a tear-down phase. The preparation and tear-down phase are optional. They can be implemented to execute any preparation or cleanup work that is needed for the test but that should not be part of the actual measurement. A good example of a preparation and cleanup would be to add and remove an index to analyze the efficiency of a range-select compared to a table scan for a particular attribute. Every test case has the constraint that schema structure and data need to be the same before and after the test. That constraint guarantees that all tests can be executed under the same conditions.

The encapsulation of test logic in a test case has three advantages.

1. It allows separation of concerns. Every test case targets the quantification of one particular mapping aspect.

2. It provides an easy extension mechanism. The investigation of new mapping aspects can be added simply by implementing a new test case that contains the execution logic.

3. It allows to adapt to different database systems. Although SQL is a standardized interface, many database vendors incorporate slightly different versions of SQL dialect. We provide test case implementations that produce standard SQL, which conforms to the SQL:2011 standard [2]. However, if a database requires a different SQL dialect, it is possible to inherit from our default test case implementation and override the execution logic. In our prototype, an inherited test case must declare what database it is designed for. This declaration refers to the database driver. Before test execution, our prototype checks what database the test should be executed for and selects the appropriate implementation.

The number of consecutive test executions is configurable. After successful execution, every test case returns the minimum, maximum, and average value for all test runs. All test cases are executed one after another, to guarantee measurements free of side-effects.

4.2.2 Prototypical implementation. The conceptual design of our framework in Section 4.2.1 is supposed to highlight our main idea. To perform the actual measurements, we implemented a prototype. The overall architecture of our prototype is depicted in Figure 7.

As programming language, we decided to use Java. This decision was taken, because we wanted a web-based solution and because most major database vendors provide a JDBC

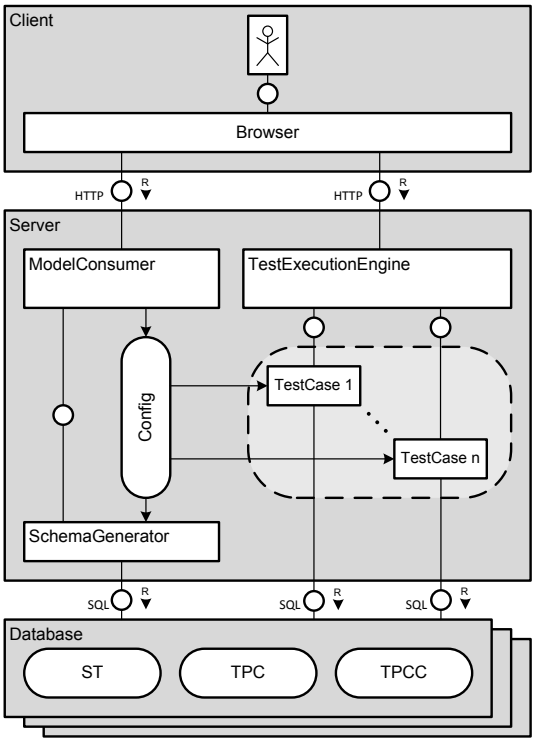

Figure 7: Framework Architecture

version of their database driver. The interface of our prototype consists of a visual editor to define the object model, a list to select the available database systems, and a list to select the test cases that should be executed on the selected database. A screenshot of the prototype's user interface is shown in Figure 8. To set up the configuration, we provide a visual editor based on GoJS[2]. All configuration parameters can be set using this editor. We provide possibilities to save and load existing models. Internally, the configuration is represented in JSON format. The interface of our web client is connected to the two server endpoints *ModelConsumer* and *TestExecutionEngine*. The *ModelConsumer* takes the object model configuration and a list of selected databases as input parameters. The configuration is stored internally. The *ModelConsumer* triggers the *SchemaGenerator*, which produces the SQL statements to generate and populate the data model according to the configuration. To

[2]http://gojs.net/

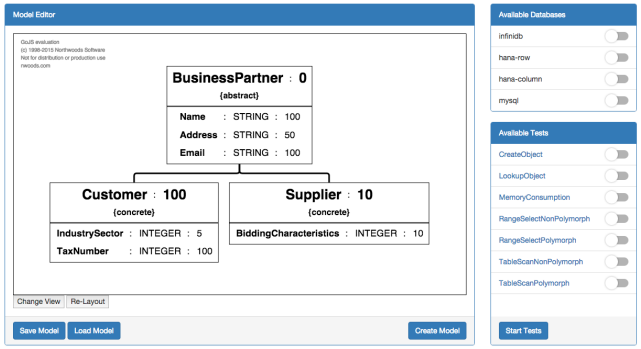

Figure 8: Prototype Interface

execute the tests, our interface connects to the *TestExecutionEngine*, which takes a list of selected databases and test cases as input. Based on the configuration, the test case implementations generate SQL statements, whose execution time can be measured. To be more concrete, we measure round trip time, which includes network communication. After a test finishes, the result is returned to the web interface.

Results are presented in the form of a table and as a radar chart. The table contains measurements for every combination of test case, database, mapping strategy and object model class. This presentation allows a fine granular analysis, which provides detailed information about how mapping aspects perform on a particular database using a certain strategy. Table 2 depicts an example of the result presentation in form of a table. Depending on the complexity of the object model (number of classes), selected databases, and test cases, the table can get confusing. For an engineer, an important information is how a mapping strategy compares to another. In other words, what is the trade-off between strategies? This information is difficult to present using a large table. That is why we add a radar chart. This chart provides a useful visualization, which lets an engineer see what profile a mapping strategy shows on what database. An example of our radar chart visualization is shown in Figure 11. Our current prototype supports four types of database systems, which can be distinguished by storage type (in-memory vs. disk) and data layout (row- vs. column-orientation). Table 1 depicts our selection. By

	Disk	**In-Memory**
Row-store	MySQL	SAP HANA
Column-Store	InfiniDB	SAP HANA

Table 1: Supported Database Types

providing one representative from each quadrant, we cover the major share of RDBMS that are used in modern enterprise information systems. Fortunately, SAP HANA provides an in-memory column- as well as a row-store. Thus, we can cover two quadrants with one database. InfiniDB[2] is basically a MySQL[3] database with a disk-based columnar storage engine.

The hardware infrastructure of our prototype comprises of two virtual machines on the same local network (10GBit). One machine hosts an Apache Tomcat server where our prototype is deployed. The other machine hosts our database servers. A diagram of our experiment setup is depicted in Figure 9.

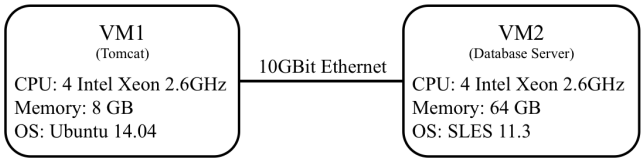

Figure 9: Experiment Setup

[2]http://infinidb.co/
[3]https://www.mysql.com/

4.3 Study Subject - Object Model

To answer Q3 and Q4, we need to define an object model, which is used as a subject in our quantitative study. We decided to use the BUCKY object model proposed by Carey et al. [10]. It was defined for a query-oriented benchmark that tests key features offered by object-relational systems. An alternative could be the OO7 benchmark [9], also proposed by Carey et al. Although OO7 is considered the standard for qualitative and quantitative comparison, it is not representative for most common operations in typical transactional/enterprise applications. OO7 focuses on extensive traversals of hierarchical structures. It was created to test the kind of specialized applications for which object-oriented databases were designed [8]. We decided to use the object model defined in [10], because it is query-oriented, better documented, and its structure is easy to comprehend. It includes an inheritance hierarchy derived from an HR system in a university context. We make one adjustment from the original model. The BUCKY model defines multi-inheritance relationships. We deliberately decided to disregard multi-inheritance from our study. This decision was made for the following reasons.

First, the majority of object-oriented programming languages does not support multi-inheritance. Second, it would add significant complexity to the framework design. Third, adding multi-inheritance would not invalidate our findings for single-inheritance. Adding support for multi-inheritance models to the framework can be a task for future work. The adapted object model is depicted in Figure 10. Ac-

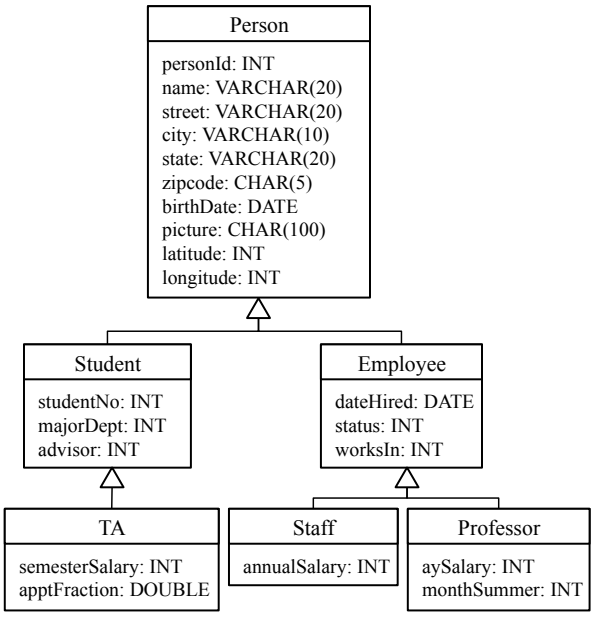

Figure 10: Adapted Version of BUCKY [10] Benchmark Model

cording to the benchmark description in [10], the system contains 50,000 objects of type *Student*, 25,000 *TA* (teaching assistants), 25,000 *Staff* members, and 25,000 objects of type *Professor*. *Employee* and *Person* are abstract classes. Our prototype contains a pre-configured version of this data model, which can be loaded on demand.

5 Measurements

The ideas behind Q3 and Q4 aim at the usefulness of an empirical framework for comparing mapping strategies. In this section, we present the measurements executed by our prototype. These measurements can be used to answer Q3 and Q4. Due to space restrictions, we only present measurements for selected mapping aspects. A more comprehensive study, incorporating all mapping aspects, can be executed using our prototype. For this paper, we select *Lookup*, *Memory Consumption*, *RangeSelectPolymorph*, *TableScanPolymorph*, and *AddAttribute*. We use 100 iterations for each test run. Table 2 shows the results of the BUCKY model analysis for the databases MySQL, InfiniDB, and SAP HANA (row- and column-store). The table shows measurements for all combinations of test case, strategy, database, and model class. Additionally, we provide aggregated values (AGG) for one strategy over all classes of the hierarchy. Query execution times are aggregated to average, *Memory Consumption* is aggregated to the sum of the individual measurements. The strategy that performs best for a particular test case is highlighted in green. These aggregated values allow an overall rating that can be used to determine what strategy is optimal for a particular test case and mapping aspect respectively.

6 Evaluation

To answer Q3, we look at the results from one particular database. At its core, Q3 reflects the engineers desire to understand how a mapping strategy performs in comparison to other strategies. This question has two parts. First, it is important to know what strategy performs best for a particular mapping aspect and second, how big is the penalty on that mapping aspects if a different strategy is selected. Both parts of the question can be answered with our empirical analysis. As an example, we select the MySQL data from Table 2, which represents disk-based row-stores. Our analysis allows to determine the mapping strategy that performs best for every test case and mapping aspect respectively. Furthermore, we can directly see how big the penalty for that mapping aspect would be for selecting a different strategy. Looking at *Memory Consumption*, the best strategy is *TPCC*. Choosing *ST* over *TPCC* would result in an increased memory footprint of 2%. However, choosing *TPC*, the increase in memory consumption would be 47%. In case of *AddAttribute*, *TPC* outperforms *TPCC* by 49% and *ST* by 222%. As mentioned in Section 4.2.2, representing such large amounts of data in a table tends to become confusing. Hence, we provide a visualization of normalized aggregated data in our radar chart. The normalization is done by assigning the value 1 to the best performing strategy. The values for the other two strategies are calculated by dividing the aggregated value of the best by their aggregated values. Figure 11 depicts the strategy profiles for the data of HANA (Column).

6.1 Finding 1

Given the information from our analysis, an engineer is able to anticipate the impact of a mapping strategy on non-functional system characteristics. Because we provide a

		Lookup (ms)			MemoryCons. (kB)			RangeSelectPoly. (ms)			TableScanPoly. (ms)			AddAttribute (ms)		
		ST	TPC	TPCC	ST	TPC	TPCC	ST	TPC	TPCC	ST	TPC	TPCC	ST	TPC	TPCC
MySQL	Person	n/a	n/a	n/a	17968	14864	0	0.24	0.20	0.26	34	28	24	769	549	858
	Student	0.25	0.29	0.21	0	3600	6672	0.22	0.18	0.26	31.23	29.06	14.35	740	264	454
	TA	0.29	0.26	0.21	0	1552	3600	0.23	0.17	0.18	32.43	29.02	5.61	797	134	177
	Employee	n/a	n/a	n/a	0	2576	0	0.22	0.20	0.22	30.21	27.34	10.03	752	268	475
	Staff	0.27	0.26	0.21	0	1552	3600	0.21	0.20	0.18	30.21	27.12	5.08	975	153	159
	Prof.	0.24	0.25	0.21	0	1552	3600	0.18	0.20	0.18	30.12	28.11	4.21	734	112	193
	AGG	0.26	0.26	0.21	17968	25696	17472	0.22	0.19	0.21	31.45	28.11	11.12	795	247	386
InfiniDB	Person	n/a	n/a	n/a	15426	13168	0	32.3	22	41.41	31.21	21.72	40.40	247	166	178
	Student	0.88	0.91	0.81	0	2896	5529	29.80	21.31	24.40	30.2	22.9	24.40	245	85	107
	TA	0.88	0.74	0.90	0	829	3051	31.93	21.83	6.21	30.9	23.7	6.50	182	34	38
	Employee	n/a	n/a	n/a	0	1924	0	29.22	23.24	15.62	28.9	21.7	15.70	185	62	71
	Staff	0.86	1.02	0.81	0	741	2856	33.61	22.70	5.72	30.4	22.6	5.40	183	27	34
	Prof.	0.87	0.93	0.82	0	926	2944	28.91	23.61	6.20	28.7	24.5	5.50	182	34	38
	AGG	0.87	0.93	0.83	15426	20484	14380	30.82	22.41	16.67	30.11	22.92	16.3	204	68	78
HANA (Row)	Person	n/a	n/a	n/a	14944	6960	0	0.33	0.32	0.33	41.7	41.6	6.91	8.17	4.83	18.82
	Student	0.19	0.31	0.18	0	3008	3584	0.32	0.26	0.25	29.12	27.31	3.77	5.81	4.23	7.51
	TA	0.19	0.32	0.17	0	1024	2208	0.31	0.25	0.27	18.21	17.72	1.42	5.32	4.42	3.92
	Employee	n/a	n/a	n/a	0	2000	0	0.31	0.28	0.23	27.52	27.21	2.62	5.48	4.12	6.71
	Staff	0.18	0.32	0.17	0	800	2000	0.33	0.24	0.21	18.70	17.98	1.42	5.63	4.03	4.07
	Prof.	0.18	0.32	0.17	0	1008	2000	0.29	0.24	0.21	18.32	17.92	1.43	5.52	4.11	3.97
	AGG	0.19	0.32	0.18	14944	14800	9792	0.31	0.26	0.23	25.52	29.94	2.91	5.92	4.21	7.05
HANA (Column)	Person	n/a	n/a	n/a	5925	4327	0	0.48	0.44	3.61	0.29	0.28	3.12	11.84	4.64	27.21
	Student	0.45	0.56	0.37	0	1658	2477	0.47	0.44	2.36	0.28	0.28	2.31	10.56	4.82	8.34
	TA	0.41	0.54	0.38	0	563	1559	0.44	0.37	0.25	0.32	0.26	0.24	5.11	9.42	10.61
	Employee	n/a	n/a	n/a	0	1120	0	0.43	0.35	2.62	0.24	0.24	2.03	10.41	4.11	18.33
	Staff	0.39	0.53	0.37	0	539	1490	0.37	0.32	0.25	0.23	0.22	0.24	5.18	10.39	7.31
	Prof.	0.41	0.53	0.42	0	563	1508	0.36	0.38	0.24	0.23	0.23	0.23	11.03	10.37	13.50
	AGG	0.41	0.54	0.38	5925	8770	7034	0.42	0.38	1.55	0.27	0.25	1.29	9.01	7.29	14.17

Table 2: Measurement Results

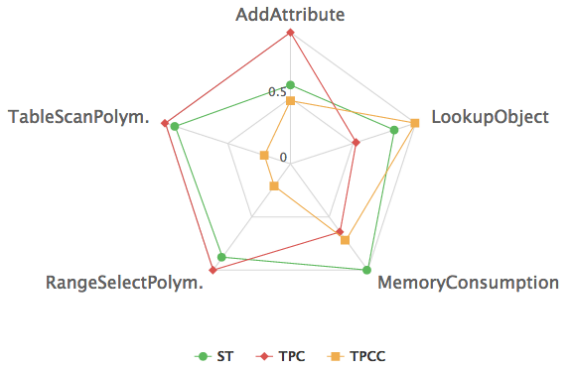

Figure 11: Strategy Comparision HANA (Column)

quantification of the impact, the engineer is also aware of the penalty for selecting an alternative strategy.

This is not possible with guidelines. Furthermore, our quantification allows a separate consideration for different model classes. Thus, an engineer can make a differentiated decision based on the information on how a strategy performs on a certain class. A good example for such differentiation is the *TableScanPolymorph* for HANA (Column). Comparing *TPC* with *TPCC*, we see that *TPCC* shows equal or better performance for classes, which are leafs in the inheritance hierarchy tree (*TA, Staff, Professor*). However, for more general classes (*Person*, *Employee*), the strategy performs much worse. In case of *Person*, the execution time of a polymorphic table scan in *TPCC* is more than an order of magnitude worse compared to *TPC*.

To answer Q4, we look at Table 2 again. We start out by comparing the aggregated values across all strategies. It is not surprising that there are differences between the absolute numbers. This can be explained by the different storage mechanisms (in-memory vs disk) and data layouts (row- vs. column-orientation). However, we can also see different rankings for individual mapping aspects. Looking at *RangeSelectPolymorph*, we can see that there is not much difference between *ST*, *TPC* and, *TPCC* for MySQL. If any, *TPC* seems to have a slight advantage over *ST* and *TPCC*. The same can be said for HANA (Row), only with the exception that *TPCC* seems to be slightly better compared to *TPC* and *ST*. A different behavior can be observed for InfiniDB and HANA (Column). For InfiniDB, the *TPCC* strategy outperforms *ST* almost by a factor of two and *TPC* by 34%. In case of HANA (Column) *TPC* outperforms *ST* by 10% and *TPCC* by a factor of four. The same observation can be made for *Memory Consumption* and *TableScanPolymorph*. Similar to the discussion of Q3, a comparison based on the numbers in Table 2 is a tedious and error prone task. To get a quick impression of the differences between mapping strategies, a comparison of their profiles using the radar chart visualization helps. Figure 12 depicts a comparison of the aggregated normalized values of the analysis from Table 2.

6.2 Finding 2

Based on our observations, we can say that database technology (storage type and data layout) has significant impact on object-relational mapping strategies. Today's guidelines do not consider different database technologies. They have been developed based on experiences and best practices with classical disk-based row-stores. Looking at these results, the validity of existing guidelines for databases other than disk-based row-stores has to be questioned.

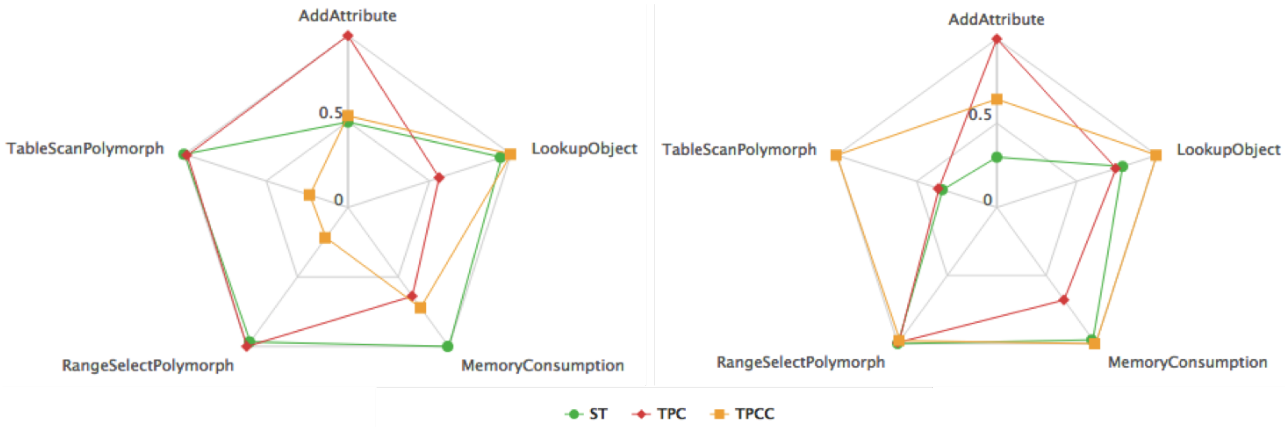

Figure 12: Strategy Comparison HANA-Column (left), MySQL (right)

7 Threats to Validity

To consider validity threats of our study, we review experiment design and results according to the validity types proposed in [11]. A distinct feature of our work is that there is no baseline to compare to. Our work is motivated by the fact that current state-of-the-art to provide information about the impact of mapping strategies are guidelines. By nature, these guidelines do not provide metrics. Hence, they are not quantifiable. Our proposed framework provides a novel approach to the decision making process in the context of object-relational mapping. To the best of our knowledge, there exists no other study, framework, or tool that provides quantification of the impact of mapping strategies on non-functional system characteristics. That is why we discuss only those validity threat types, which are applicable to our study.

Confirmability: Our framework proposes the use of test cases. Thus, the results of our study are shaped by the quality of our implementation. We have put a strong focus on implementing the test cases according to the descriptions and semantics defined in Section 4.1. All code is made available online and can be reviewed. We have designed a framework that is open to extensions and adaptations. Other researchers are able to reproduce results based on our implementations, they can adapt our implementations, and they can provide new test cases to quantify mapping strategy aspects, which we have not yet considered. Also, confirmability is impacted by the choice of databases, which we incorporated into our study. Primary focus for this study was to select one database from every quadrant of Table 1. Even within the same quadrant, different RDBMS may show different characteristics. Our framework design supports integration of new types of RDBMS with little effort. Hence, a possible successor study could analyze the differences between RDBMS in one of the four quadrants.

External Validity, Transferability: The generalization of our results is critical at this point. As explained in Section 2, we want to create awareness for the significance of having transparency in the decision making process for designing O/R middleware. For this particular study, we selected only a single object model as a study subject. Thus, the findings in this paper are hardly generalizable at this point. However, we are able to show that database technology has a significant impact on the performance of O/R middleware. This substantiates our initial hypothesis that existing guidelines and best practices have to be questioned when designing O/R middleware for different types of databases.

Construct Validity, Internal Validity: The central question is if different strategy profiles are caused by changing database technology or if there may be other effects that influence our results? Since we use the same test case implementation logic for all four RDBMS, the only parameter that has changed in the study of Q4 is the database technology. Consequently, there is no other effect that could impact our measurements.

8 Related Work

Research in the area of ORM has been subject to various discussions [7, 19, 17]. Regarding the mapping of class inheritance hierarchies, research can be divided in two main areas: the description of strategy semantics and the definition of methods for mapping generation and selection.

Strategy description. Starting in the 1990s, a number of works proposing mapping strategy semantics have been published. Barsalou and Wiederhold [5] first discussed the semantics of object-oriented class inheritance in the context of relational databases. Keller et al. [14] investigated possibilities to map single inheritance in C++ to database tables. In [15], Keller proposes a pattern language to map objects to tables. The most comprehensive discussions of inheritance mapping strategies can be found in textbooks by Fowler [12] and Ambler [4]. Both books also include explicit guidelines that should aid software developers in choosing appropriate mapping strategies.

Mapping generation and selection. Cabibbo [6] propose a mapping model that allows to automatically generate mappings for ORM tools. Their work is restricted to the automatic generation of mappings. It neither provides an automatic selection, nor does it discuss possible selection criteria. Philippi [18] suggests a model driven approach to automatically generate object-relational mappings based on non-functional software requirements. His approach assigns ratings for non-functional system characteristics on mapping strategies. However, it remains unclear what metrics or reasoning is used to come up with ratings.

To the best of our knowledge, there is no cost model, method, or tool chain, which would allow an exact comparison of the strategies regarding their impact on the non-functional characteristics.

9 Conclusion and Future Work

State-of-the-art for supporting decision making for designing mapping functionality for object-oriented class inheritance hierarchies are guidelines. Guidelines have two deficiencies. First, they do not provide a quantification of the impact of a mapping strategy. Hence, it is not possible to compare mapping strategies effectively. Secondly, guidelines do not take different RDBMS technologies into account.

In this paper, we propose an empirical approach to quantify the impact of O/R mapping strategies on non-functional system characteristics. This approach is superior to guidelines, because quantification allows anticipating the consequences of choosing a mapping strategy. Leveraging our framework increases transparency and accuracy of the decision making process. A comparison based on empirical data allows to make differentiated decisions regarding the choice for a mapping strategy. This is particularly helpful for designing new systems, but it is equally important for performance optimizations of existing systems. Furthermore, our evaluations show that the RDBMS technology influences the characteristics of mapping strategies significantly. This influence has not been discussed so far. Consequently, it has not been considered by state-of-the-art guidelines.

The goal of our research is to understand the influences and relations between structural aspects, e.g., the class hierarchy, runtime aspects, e.g., data model population, mapping strategy, and database technology. As a first step, we want to focus on the investigation of runtime behavior. Especially in the context of enterprise applications (standard software), the ways how a software is used vary. Looking at our class hierarchy in Figure 10, universities with different sizes and internal structures might populate the exact same domain/data model differently. We want to understand how runtime characteristics, e.g., number of instances per class, impact aspects of O/R mapping layer.

10 References

[1] ISO/IEC 9126-1:2001 software engineering – product quality – part 1: Quality model, 2001.

[2] ISO/IEC 9075:2011 information technology - database languages - sql, 2011.

[3] D. J. Abadi, S. R. Madden, and N. Hachem. Column-stores vs. row-stores: How different are they really? In *Proc. of the ACM SIGMOD Int. Conf. on Management of Data*, pages 967–980, New York, NY, USA, 2008.

[4] S. Ambler. *Agile Database Techniques: Effective Strategies for the Agile Software Developer*, chapter 14 - Mapping Objects to Relational Databases, pages 231–244. John Wiley & Sons, Inc., New York, NY, USA, 2003.

[5] T. Barsalou and G. Wiederhold. Complex objects for relational databases. *Computer-Aided Design*, 22(8):458–468, 1990.

[6] L. Cabibbo and A. Carosi. Managing inheritance hierarchies in object/relational mapping tools. In *Proc. of the 17th Int. Conf. on Advanced Information Systems Engineering (CAiSE)*, pages 135–150, Porto, Portugal, 2005.

[7] M. J. Carey and D. J. DeWitt. Of objects and databases: A decade of turmoil. In *Proc. of the 22th Int. Conf. on Very Large Data Bases*, pages 3–14, San Francisco, CA, USA, 1996.

[8] M. J. Carey, D. J. DeWitt, C. Kant, and J. F. Naughton. A status report on the oo7 OODBMS benchmarking effort. In *OOPSLA'94, Proceedings of the Ninth Annual Conference on Object-Oriented Programming Systems, Languages, and Applications, Portland, Oregon, USA, October 23-27, 1994.*, pages 414–426, 1994.

[9] M. J. Carey, D. J. DeWitt, and J. F. Naughton. The 007 benchmark. In *Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data*, SIGMOD '93, pages 12–21, New York, NY, USA, 1993. ACM.

[10] M. J. Carey, D. J. DeWitt, J. F. Naughton, M. Asgarian, P. Brown, J. E. Gehrke, and D. N. Shah. The bucky object-relational benchmark. In *Proceedings of the 1997 ACM SIGMOD International Conference on Management of Data*, SIGMOD '97, pages 135–146, New York, NY, USA, 1997. ACM.

[11] R. Feldt and A. Magazinius. Validity threats in empirical software engineering research - an initial survey. In *Proceedings of the 22nd International Conference on Software Engineering & Knowledge Engineering (SEKE'2010), Redwood City, San Francisco Bay, CA, USA, July 1 - July 3, 2010*, pages 374–379, 2010.

[12] M. Fowler. *Patterns of Enterprise Application Architecture*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2002.

[13] S. Holder, J. Buchan, and S. G. MacDonell. Towards a metrics suite for object-relational mappings. In *Proc. of the 1st Int. Workshop on Model-Based Software and Data Integration (MBSDI)*, pages 43–54, Berlin, Germany, 2008.

[14] A. M. Keller, R. Jensen, and S. Agarwal. Persistence software: Bridging object-oriented programming and relational databases. In *Proc. of the ACM SIGMOD Int. Conf. on Management of Data*, pages 523–528, New York, NY, USA, 1993.

[15] W. Keller. Mapping objects to tables - a pattern language. In *Proc. of the European Conf. on Pattern Languages of Programming Conf. (EuroPLOP)âĂŹ*, Berlin, Germany, 1997.

[16] J. Krüger, M. Grund, A. Zeier, and H. Plattner. Enterprise application-specific data management. In *Proc. of the 14th IEEE Int. Enterprise Distributed Object Computing Conf. (EDOC)*, pages 131–140, Vitória, Brazil, 2010.

[17] T. Neward. The vietnam of computer science, 2006. [Accessed: September 20, 2015].

[18] S. Philippi. Model driven generation and testing of object-relational mappings. *J. Syst. Softw.*, 77:193–207, 2005.

[19] C. Russell. Bridging the object-relational divide. *Queue*, 6(3):18–28, May 2008.

[20] V. Sikka, F. Färber, A. K. Goel, and W. Lehner. SAP HANA: the evolution from a modern main-memory data platform to an enterprise application platform. *PVLDB*, 6(11):1184–1185, 2013.

[21] M. Stonebraker and U. Cetintemel. "one size fits all": An idea whose time has come and gone. In *Proceedings of the 21st International Conference on Data Engineering*, ICDE '05, pages 2–11, Washington, DC, USA, 2005. IEEE Computer Society.