Information and Software Technology 82 (2017) 1–18

Contents lists available at ScienceDirect

Information and Software Technology

journal homepage: www.elsevier.com/locate/infsof

Twenty years of object-relational mapping: A survey on patterns, solutions, and their implications on application design

Alexandre Torres[a,*], Renata Galante[a], Marcelo S. Pimenta[a], Alexandre Jonatan B. Martins[b]

[a] Instituto de Informática - Universidade Federal do Rio Grande do Sul (UFRGS), Av. Bento Gonçalves, 9500, Porto Alegre, RS, 91501-970, Brazil
[b] TAGMATEC, Rodovia SC-401, 600, ParqTEC Alfa, CE Alfama, sala 405, Florianópolis, SC 88030-911, Brazil

ARTICLE INFO

Article history:
Received 30 December 2015
Revised 23 September 2016
Accepted 27 September 2016
Available online 28 September 2016

Keywords:
Object-relational mapping
Design patterns
Impedance mismatch problem
Enterprise patterns
Class models

ABSTRACT

Context: Almost twenty years after the first release of TopLink for Java, Object-Relational Mapping Solutions (ORMSs) are available at every popular development platform, providing useful tools for developers to deal with the impedance mismatch problem. However, no matter how ubiquitous these solutions are, this essential problem remains as challenging as ever. Different solutions, each with a particular vocabulary, are difficult to learn, and make the impedance problem looks deceptively simpler than it really is.

Objective: The objective of this paper is to identify, discuss, and organize the knowledge concerning ORMSs, helping designers towards making better informed decisions about designing and implementing their models, focusing at the static view of persistence mapping.

Method: This paper presents a survey with nine ORMSs, selected from the top ten development platforms in popularity. Each ORMS was assessed, by documentation review and experience, in relation to architectural and structural patterns, selected from literature, and its characteristics and implementation options, including platform specific particularities.

Results: We found out that all studied ORMSs followed architectural and structural patterns in the literature, but often with distinct nomenclature, and some singularities. Many decisions, depending on how patterns are implemented and configured, affect how class models should be adapted, in order to create practical mappings to the database.

Conclusion: This survey identified what structural patterns each ORMS followed, highlighting major structural decisions a designer must take, and its consequences, in order to turn analysis models into object oriented systems. It also offers a pattern based set of characteristics that developers can use as a baseline to make their own assessments of ORMSs.

1. Introduction

Relational Databases (RDBs) and Object-Oriented Programming Languages (OOPLs) are based upon distinct paradigms, containing technical, conceptual, and cultural incompatibilities. The obstacles of dealing with such incompatibilities are commonly referred to as the object-relational Impedance Mismatch Problem (IMP) [1–3].

If system analysis aims at the recognition of the business problems, sketching platform independent solutions, such as identifying and applying analysis patterns [4], the design activity is focused in fitting the analysis to the limitations of the chosen platform. At the design, the IMP can turn analysis models into something difficult to understand, maintain, evolve, and even track back to the original idea.

For example, two basic concepts on OO models are *inheritance* and *many-to-many* relationships, both extensively used on analysis models, such as analysis patterns [4], and both are absent on RDBs. Primary and foreign keys are important concepts for good database design, and both are absent on OOPLs. Bridging these conceptual mismatches, without losing the tracking among all artifacts, is the challenge of the IMP.

In many projects it is possible to avoid the IMP, by not adhering to Object-Oriented (OO) practices, or not using a RDB. Sometimes it is feasible to place all domain logic within stored procedures. Another way is using *NoSQL* persistence, such as Document-Oriented Databases, and deal with other kind of mis-

* Corresponding author.
E-mail address: atorres@inf.ufrgs.br (A. Torres).

http://dx.doi.org/10.1016/j.infsof.2016.09.009
0950-5849/

matches related to Object-Document Mapping [5]. Nevertheless, relational databases (RDBs) continue to be the backbone of information systems, and nobody knows if this will ever change [6].

From the developer standpoint, Object-Relational Mapping (ORM) encompasses solutions for mapping business objects to relational data, by separating persistence concerns on a *persistence layer* [7]. Contrary to popular belief, ORM is not trivial nor fully automatic: mappings are a kind of *updatable view*, and its automation is undecidable [8]. Moreover, ORM relies on experienced developers being able to trade-off between database design concerns, such as data normalization, primary keys, and relationships, and the forces of OO design, such as pursuing high cohesion and low coupling [3,9].

Design patterns are a great way to organize knowledge and teach good practices, and this also applies to ORM patterns [7,10–12]. For quite some time, Object-Relational Mapping Solutions (ORMSs) were expensive and limited, and these patterns were an important reference when developing the ORM from scratch. Nowadays ORMSs are ubiquitous: every OO platform has at least one ORMS, sometimes with an "official" API. This popularization made ORM looks simpler than it really is [13].

Each ORMS has a different terminology, with different names to concepts that have similar, or even the *same,* meaning. For example, *embedded values* are also referred as *complex types, composite columns,* and *aggregated values,* all of them representing the same basic pattern. ORMSs named after patterns, such as *Data Mapper* and *Active Record*, in reality aggregates a combination of different pattern implementations, what may confuse developers about their propose and scope. Moreover, each OOPL has particularities that must be taken into account when dealing with ORM, such as if it has static or dynamic types, or if it supports multiple inheritance. This all contributes to categorizing IMP as the *Vietnam of computer science* [14].

This paper pursues answers to developers and designers about what compromises must be taken when dealing with the IMP, and how to assess the strengths and weakness of ORMSs. It presents a critical survey in the scenario of ORMSs, relating its characteristics with the established design/architectural patterns in the literature. In order to be relevant to the widest public, within a limited survey, the ORMSs were selected to represent the most popular OOPLs.

The scope of this study covers the characteristics of ORMSs related to the static view of modeling [15], including all data structure concerns, as well as the organization of operations on the data, describing behavioral entities as discrete modeling elements. This includes the presence of *Create, Read, Update and Delete* (CRUD) basic persistence operations, but not the design of their behavior; includes the way a relationship can be fetched and the impact of each decision, but not the way queries are designed to fetch related data. The design of queries in ORMSs, using SQL, or other proprietary object query languages, is out of the proposed scope. Internal implementation mechanisms of ORMSs were left out, such as *Unit of Work* and *Repository* patterns, because they are difficult to assess, and should not interfere with structural design.

The contribution of this paper is to provide a common terminology for the assessment of ORMSs, and a better understanding on ORM patterns and how they are applied across distinct platforms and frameworks. It summarizes a list of important decisions that a developer will have to make when using ORMSs. This information helps the designer in the anticipation of project limitations, the documentation of design decisions due to ORMS characteristics, and a better traceability between implementation artifacts and conceptual elements. This paper is not intended to point out what is the best of the ORMSs, but rather to help understand how to compare them.

The remaining of this paper is organized as follows: in Section 2 we discuss the related work that drives our study, with a brief historical introduction of the impedance mismatch problem; in Section 3 we present the methodology used in the survey; in Section 4 we present the survey of ORMSs relating patterns, implementations, and implications for the application design; in Section 5 we present conclusions and future work.

2. Historical background and related work

Initial research on the object-relational IMP may be traced back to the practical need of storing *Smalltalk* objects in persistent databases [1]. At that time, most studies were focused on the upcoming new generation of OO database systems, discussing issues later employed for ORM such as inheritance, associations, and polymorphism [16,17].

Patterns are recurring solutions identified by experience and documented as practical guides to software design [18]. In the 90s, due to the long takeoff of OO databases, impedance mismatch research shifted to ORM. The early impedance solutions were organized as an easy to access "buffet" of design patterns for system designers and developers. Patterns for each persistence layer component, and approach, covered several strategies to overcome the IMP [7,10,19].

TopLink released the first known commercial ORMS for *Smalltalk* in 1994, and two years later for *Java* [20]. The central patterns of interest in this paper (ORM patterns) are those focused on the mapping of objects to tables, since they establish the common ways of designing classes that represent database objects and *vice versa* [12].

In the twenty-first century the OO databases adoption was near stagnant [21]. The set of persistence patterns became part of a so called enterprise pattern set [11], as the impedance mismatch problem was recognized as much more than a technical concern, encompassing conceptual and cultural problems [3].

The dominance of OO languages and the inexpensive/free ORMSs disseminated the use of persistence layers and a *Domain Model* approach, where domain classes centralized behavior and data. Successful ORMSs such as *TopLink* (now *EclipseLink*) and *Hibernate* contributed to, and received contribution from, standards such as *JDO* and *JPA*, both influenced by the object data standard [22]. Failure in adopting ORMSs is often related to SQL and performance anti-patterns [23,24].

The impedance mismatch problem attracted some attention beyond the pattern scope in recent publications. Model management fits the IMP as one type of engineered mapping problem, solved by schema evolution operations [25]. Mappings are also subject of study for the general impedance mismatch case [26]. Both works deal with impedance mismatch beyond the OO domain, including XML, *Cobol* and Data warehousing mismatch problems.

Another classification for data mappings was proposed in a general top-down way: a conceptual framework based on concerns such as paradigm, language, schema and instance; each concern influencing the subsequent choices [27]. Our classification has a bottom-up approach, and deals with a smaller set of representative frameworks, focusing more on language, schema and instance options for ORMSs.

3. Methodology

This survey is based on bibliographic research, aiming at the relationships between the ORMS information and the pattern literature. The sources of ORMSs information are obtained at the descriptions, online manuals, published papers (when they exist), books, and practical experience found on the internet validated by the personal experiences of the authors. The ORM patterns were