2009 First International Conference on Advances in Databases, Knowledge, and Data Applications

A Classification of Object-Relational Impedance Mismatch

Christopher Ireland, David Bowers, Michael Newton, and Kevin Waugh

Department of Maths and Computing

The Open University, Milton Keynes, UK

{cji26@student.open.ac.uk, D.S.Bowers@open.ac.uk, M.A.Newton@open.ac.uk, K.G.Waugh@open.ac.uk }

Abstract

Object and relational technologies are grounded in different paradigms. Each technology mandates that those who use it take a particular view of a universe of discourse. Incompatibilities between these views manifest as problems of an object-relational impedance mismatch. In this paper we propose a conceptual framework for the problem space of object-relational impedance mismatch and consequently distinguish four kinds of impedance mismatch. We show that each kind of impedance mismatch needs to be addressed using a different object-relational mapping strategy. Our framework provides a mechanism to explore issues of fidelity, integrity and completeness in the design and implementation of existing and new strategy choices. Our framework will be of benefit to standards bodies, tool vendors, designers and programmers as it will provide them with new insights into how to address problems of an object-relational impedance mismatch both at the most appropriate levels of abstraction and in the most appropriate way.

1. Introduction

A paradigm is a particular way of viewing a universe of discourse. Each paradigm comes with its own particular abstractions, organising principles and prejudices. There are a number of different paradigms in computing. Each paradigm influences both the process and the artefacts of software design and development.

Combining technologies based on different paradigms presents a set of problems for those responsible for the design and implementation of an application. We refer to each such problem as an impedance mismatch problem. However people are inventive and proponents of one paradigm may believe that they have solved an impedance mismatch problem. Such a solution will, typically, involve using a subset of concepts from one paradigm to represent a concept in the other. It then becomes received wisdom within a community both that there is a solution to a problem and that the solution is understood by all concerned.

The relational paradigm has proven popular for the development of databases whilst at the same time the object paradigm has underpinned a number of programming languages and software development methods. The popularity of technologies which embody the different paradigms in these two separate but essential aspects of software development means that inevitably they will be used together. Differences of abstraction, focus and language lead to problems when these technologies are combined in a single application.

An object-relational application combines artefacts from both object and relational paradigms. Essentially an object-relational application is one in which a program written using an object-oriented language uses a relational database for storage. A programmer must address object-relational impedance mismatch ("impedance mismatch") problems during the production of an object-relational application. For some authors however there is no impedance mismatch [1]. This is true for those developing an entire application using a single programming language such as Visual Basic, C++, Java or SQL-92 ("SQL") because each language is based on a single paradigm. Those who combine object and relational technologies, and must work across paradigms, have a different experience [2], [3]. The received wisdom is that impedance mismatch problems are both well understood and resolved by current solutions. However, for each such impedance mismatch problem there is a choice of solutions that we refer to collectively as object-relational mapping (ORM). Each solution addresses problems of an impedance mismatch in a different way.

During the development of an object-relational application the resolution of impedance mismatch problems involves many different software development roles, and takes time and effort to achieve [3]. Twenty five to fifty percent of object-

978-0-7695-3550-0/09 $25.00 © 2009 IEEE
DOI 10.1109/DBKDA.2009.11

relational application code deals with impedance mismatch problems [4]. The problem of impedance mismatch has been labelled "the Vietnam of Computer Science" [3] because initial quick wins based on the received wisdom are rapidly replaced by a quagmire of issues. The popularity of object and relational technologies, the different interpretations of ORM, the plethora of approaches and technologies for the resolution of an impedance mismatch, and the existence of guidelines [5] and metrics [6] for selecting a strategy also suggest that problems of an impedance mismatch are neither uncommon nor trivial.

Understanding the contradiction between the received wisdom and the reality of impedance mismatch provides the motivation for this paper. In this paper we start to expose the contradiction by understanding the nature of the problem space and the different ways ORM is used to address it. To that end, we provide both a conceptual framework for the problem space and a classification of impedance mismatch which highlights a number of different ORM strategies.

2. The Impedance Mismatch Problem Space

In the literature and in practice we find many examples of approaches to resolving the problem of an impedance mismatch ([7],[8],[9],[10], and [11]). The general term used to refer to such approaches is object-relational mapping (ORM). Essentially, ORM is how we address the problem of impedance mismatch but in research and practice the term ORM is used to refer to a number of different things: for Fussell [8] it is a transformation process; for others ORM is something defined in the configuration of a mapping tool such as Hibernate [12]; whilst to others it is a pattern [10] or canonical mapping [11] used as the basis for a design transformation. Practitioners recognise ORM as both a process and a mechanism ([2], p225) by which an impedance mismatch is addressed. As a process, ORM is the act of determining how objects and their relationships are made persistent in a relational database: in essence the selection of one or more patterns [10]. As a mechanism ORM forms the definition of correspondence necessary for the successful implementation of a particular pattern as one or more mappings. A mapping is described between two representations in different implementation languages. In order to address an impedance mismatch, this mapping is codified as one or more transformations within some part of an application. For the developer of an object-oriented application that must use a relational database for persistence, ORM may be all of these: impedance mismatch is also a fact of life [10]. We observe from this variety that there is no consensus on a single strategy for ORM and by implication how we address impedance mismatch. Indeed, each of the current solutions addresses a different aspect of impedance mismatch. Some strategies focus on equivalence between a class and a table [10] (what to map) whilst others propose a unified query language [13] (how to map) or software architecture [8] (where to map). Evidently when these writers use the term "impedance mismatch" they are not talking about the same thing. We require some form of organising principle which goes beyond received wisdom to facilitate the comparison of strategies, and which also recognises an essential aspect of the problem: that there are different levels of abstraction.

3. Problems of an Object-Relational Impedance Mismatch

In the context of object-relational application development, one objective of ORM is to isolate a programmer using an object-oriented programming language (OOPL) from the need to understand the SQL language, the schema of an SQL database, and its implied semantics. A programmer need not focus on how to store an object but on what to store and when to store it, or on what to retrieve and when to retrieve it. Such a strategy is typical of ORM products such as Hibernate [12] and Oracle TopLink. They provide a programmer with a virtual object database; presenting data in a relational database as if it were a collection of objects. However, ORM does not isolate a relational database from an object-oriented program. The design of a relational database must address issues such as data redundancy, data integrity, data volumes, access control, concurrency, performance and auditing. Impedance mismatch problems occur when these requirements are at odds with the design of an object-oriented program. These problems have been described by [3] and [2]. In Table 1 we have catalogued the issues emerging from their work as impedance mismatch problems.

It is evident from Table 1 that impedance mismatch is not a single, well-defined problem. The different interpretations of ORM also indicate that there is no single, well-defined solution. If we are to understand impedance mismatch, we must understand the nature of these different problems and how they are addressed by different approaches to ORM. At a detailed level, this understanding provides the motivation for our conceptual framework and classification.

Table 1 - A catalogue of impedance mismatch problems

Problem	*Description of the problem and typical questions raised*
Structure	A class has both an arbitrary structure and an arbitrary semantics defined through methods. A class may also be part of a class hierarchy. SQL-92 does not provide an analogy for a class hierarchy or support repeating groups within a column. How then do we best represent the structure of a class using SQL-92?
Instance	To conform to relational theory, a row is a statement of truth about some universe of discourse, but an object is an instance of a class and may have an arbitrary structure. How does a row correspond to an object and where is the canonical copy of state located? Essentially, how much of an object must we maintain in a database?
Encapsu-lation	The state of an object is accessed via methods. The state of a row has no such protection and may be modified by other applications. How do we ensure consistency of state between an object and a row?
Identity	An object has an identity independent of its state. This in-memory identity will be different between two executions of a program. Within the same execution, two objects with the same state are different if they have a different identity. The primary key of a row is part of the state of that row. How do we uniquely identify a collection of data values across both object and relational representations?
Process-ing Model	An object model is a network of interacting discreet objects and access is based on navigation. The relational model is declarative and access is set-based. The object and relational models represent references in different directions. A transaction may not require all the data about an object. How do we represent in, maintain in, and retrieve from a database a sufficient set of in-memory objects?
Owner-ship	A class model is owned by a programming team; a relational schema is ultimately owned by a database team, it may hold legacy data and may also be used by other applications. When things change how do we maintain the necessary correspondence between a class model and a database schema?

4. A Conceptual Framework for Impedance Mismatch

In this section we consider how we might organise the different views of ORM. Other models such as [8] and [14] focus on client/server software architectures. Essentially, they help inform where one might perform a mapping. Hohenstein [9] considers programming language issues and, for a C++ programmer, helps inform how to perform a mapping.

Our catalogue of impedance mismatch problems (Table 1) defines a problem space for ORM. Our goal is to understand how ORM approaches address this problem space. We propose a conceptual framework as an abstraction of this problem space. We describe four kinds of ORM strategy: reconciliation, pattern, mapping, and transformation. Each strategy has different implications for the resolution of an impedance mismatch and helps to inform the kind of mapping necessary. Our framework comprises four levels of abstraction common to both object and relational technologies. Within each level there are both object and relational contributions. We summarise the four levels in Table 2.

Table 2 - Our conceptual framework of impedance mismatch

Level	ORM is concerned with…
Paradigm	Issues relating to incompatibilities between the two different views of a concept from a universe of discourse: one as a network of interacting objects; and the other as a set of relations.
Language	Issues relating to the incompatibility of data structures between object and relational based languages. Lammel ([11], p182) refer to this as a canonical mapping. In this paper we will use the term pattern in the context of Keller [10], as an outline description of a solution.
Schema	Issues relating to the maintenance of two representations of a particular concept described in different languages.
Instance	Issues relating to the storage and retrieval of an object in the context of an object-relational application.

The level names are terms that may have alternative interpretations and therefore require clarification. A paradigm is one particular way of viewing a universe of discourse ([15], pA-6). A language is used to produce an abstract description of a universe of discourse. We consider a concept to be some identifiable collection of things from a universe of discourse. A schema is a description (representation) of some concept from a universe of discourse, expressed using a particular language. We consider program source code the schema for an executing program just as an SQL script is the schema for a relational database. Finally, an instance is data about some thing from the universe of discourse set within a particular schema.

The relationship between the levels of our conceptual framework (Table 2) is that of context. A paradigm sets the context for the semantics of a language. There are many possible programming languages. A language provides data and processing structures for describing the semantics of a universe of discourse in the form of a schema. There are many possible schemata. A schema sets the structure into

which data about some thing from a universe of discourse must fit. Conversely a schema sets constraints on what it is we can represent about some thing from a universe of discourse.

This contextualisation has implications for choices made during the development of an object-relational application. A development language implies not only a choice of paradigm but also a set of structures and patterns that may be used ([2], [9], and [10]). The maintenance of a legacy application may dictate the use of a particular language. Choices made during the design of an object model and an SQL schema dictate the content of a mapping schema. During program and database schema development choices may be made by different teams with different agendas. In particular, a programmer has many technologies and algorithms from which she may choose in order to implement a transformation: for Java alone there is a choice of JDBC, Hibernate, JDO and TopLink to name but a few. All levels of our conceptual framework have influence on the work a programmer must do in order to resolve an impedance mismatch. When we use the term ORM we must recognise that an impedance mismatch problem can have its source at any of these levels, and understand how and at what level(s) a problem is best addressed.

5. A Classification of Impedance Mismatch

Previously we stated that ORM is the term used to classify approaches to impedance mismatch. In this section we analyse the relationship between the impedance mismatch problems described in Table 1 and our conceptual framework (Table 2). Our analysis leads to four kinds of impedance mismatch which are addressed by separate ORM strategies: reconciliation, pattern, mapping, and transformation.

Our classification provides a new way to think about the problem of impedance mismatch and how we might go about resolving it. Each strategy represents a particular way of conceptualising an impedance mismatch problem and a particular set of options for addressing it. For each strategy we provide illustrations from the literature.

We will base our examples on an extract of a class model of a School (Figure 1) that represents the following concepts: a person must have an address; and Student and Teacher are two kinds of person. Any attributes required to support an example will be added when appropriate. Even with such a simple class model there are a number of options for its representation as a relational model and as an SQL database schema (see [9] and [10]).

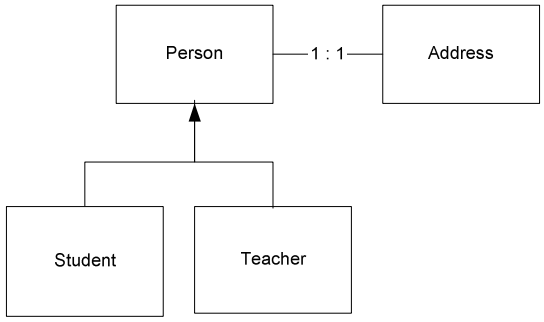

Figure 1 - A conceptual model of a school using a class model

5.1. Paradigm

An ORM strategy at the paradigm level involves the reconciliation of the different perspectives of a universe provided by the object and relational paradigms. Different aspects of an object-relational application are grounded in each paradigm. Typically the object paradigm influences program design and the relational paradigm database design. ORM in this context is the act of bridging the differences between these two paradigms. This is the essence of the impedance mismatch problem. It is important to understand the nature of these differences.

There is no consensus of terminology. Each paradigm uses its own set of building blocks to describe a universe of discourse. Although there is no single agreed definition of an object-based representation (the UML is one attempt but there are others [16]), such a representation will typically include concepts such as class, subclass, object, attribute, and association. There is, however, a single definition of what constitutes a relational representation [17], which will include concepts such as relation, tuple and domain.

There is some correspondence between the building blocks. The relational paradigm does not prescribe the domains that may be used. Neither does an object paradigm prescribe the objects that may be used. A relation represents an assertion (a predicate) about a universe of discourse involving one or more domains; and a tuple of a relation is, formally, a statement of truth about that universe. There is no equivalent representation in the object paradigm. An object is not a representation of a statement of truth about a universe of discourse. For data about a person to also be represented as data about a student, the relational model requires two tuples. An object model based on Figure 1 would necessitate only one object of the class Student. This is because the semantics of the sub-class relationship state that an object of a sub-class Student is also a member of its parent class Person. Furthermore an object has identity and

encapsulates its state whereas a tuple does not. A class defines the allowable attributes and behaviour of an object but its definition is not based on predicate logic. Whereas the relational model is concerned with statements of truth, an object has arbitrary semantics. The behaviour of an object is defined using methods and a valid state is defined using a constraint. In this respect a tuple may be considered inert in so far as it has no intrinsic behaviour. Instead a tuple may be the target of a relational operator such as project, restrict or union. A lack of correspondence between two perspectives on a universe of discourse materialises as an impedance mismatch. We label this kind of impedance mismatch a *conceptual mismatch* and it is addressed using an ORM reconciliation strategy.

A reconciliation strategy must address differences in perspective, terminology and semantics. The designer of an object representation and the designer of a relational representation view and describe aspects of a universe of discourse in different ways. The designer of an object-relational application must identify correspondence and reconcile differences between these two perspectives.

One example of the reconciliation of object and relational semantics is provided by Date [18]. He emphasises that relational theory is not at odds with the ideas of object-orientation. Just as the semantics of a class are arbitrary, the relational model does not prescribe the data types that may be defined. Consider that a relation STUDENT based on the class Student (Figure 1) may include a domain ADDRESS that is both a complex data type and the implementation of a class. There is therefore scope for addressing a conceptual mismatch.

5.2. Language

An ORM strategy at the language level is concerned with identifying general patterns of correspondence between the data structures available in an OOPL such as Java, and those structures available in SQL.

Each language reflects the paradigm on which it is based. The outline structure of a Java program is a collection of classes. A class may be viewed as a template for the creation of an object at run-time. An SQL schema is a description of a collection of tables. A table corresponds to a relation. Whereas, formally, a tuple is a statement of truth, the semantics of a row are less strict. A row represents data about some thing from a universe of discourse. Each row corresponds to a tuple.

A significant difference between Java and SQL is the extensibility of their type systems. Whereas a class is an essential part of the extensibility of the Java type system there is no equivalent extensibility in the SQL type system. Implementing a representation of a relation in an OOPL [19] or an SQL like syntax within an OOPL [13] may move the primary focus of ORM activities to the schema level, but it does not address this extensibility issue or remove the need for an ORM strategy.

Generally a class is part of the type system of an object-oriented program. A Java program based on Figure 1 could define a variable of type Address. Using SQL a class is something that may be represented; it is not an extension of the type system and also not a first-class citizen. In an SQL schema therefore a representation of Address may not be used to define the type of a column. This is the essence of a structure problem (Table 1) and there exist a number of patterns to help resolve this [10]. Aspects of an object-oriented design such as a class and an object fit into a *representation* that must be described using the existing features of SQL. A column is a scalar value and cannot adopt the type of a class represented in such a way. This representation is limited as it may be used to only store the state and not the behaviour of an object. In an object-oriented application at run-time an object has a unique identity independent of its state. The value of a primary key is part of the state of a row. This is the essence of the identity problem. The mismatch between two descriptions of a concept materialises as an impedance mismatch. We label this kind of impedance mismatch a *representation mismatch* and it is addressed using an ORM pattern strategy.

A pattern provides a way to describe correspondence between data structures. SQL provides an approximation of the data structures mandated by the relational paradigm, just as Java provides an approximation of those mandated by the object paradigm. The syntax and grammar for SQL is defined by standard and is implemented in vendor-specific languages such as Oracle and SQLServer. None of these languages is a pure implementation of SQL but nevertheless may be classified as a relational language. In practice we must address not only differences between languages as defined by their respective standards but also differences between vendor implementations. A pattern strategy must provide one or more patterns (such as [10]) that address issues of structure and identity.

5.3. Schema

An ORM strategy at the schema level will produce a mapping between two representations of a concept. Our emphasis here is on design issues. The description of a concept within an object-relational application will involve at least two schemas: one based on class and the other based on table. These two representations of a concept are different not just

because they are phrased in a different language, but because the purpose of those designing a class model is different from the purpose of those designing an SQL schema. Whilst those designing a class will focus on the cohesive representation of a network of interacting objects; the focus of those designing a SQL schema is typically data volume, data integrity, and notably the removal of redundant data. Figure 1 may be familiar to only one part of a development team: the programmers. Database designers will conceive a different solution based on tables which may not have a one-to-one correspondence to that class model. This difference of focus is the essence of the ownership problem and produces a kind of impedance mismatch that we label an *emphasis mismatch*. An emphasis mismatch is addressed using an ORM mapping strategy.

Here we provide two simple examples of a mapping strategy based on Figure 1. The mappings we use are based on the patterns described in [10]: (1) a Java program defines the class Person that includes a non-scalar attribute phoneNumber that is a list of the telephone numbers by which a person may be contacted. In order to preserve this attribute in an SQL schema an additional table PHONE_NUMBER would be created with a row for each phone number for each object of the class Person; and (2) a Java program defines the class Student that is a sub-class of Person. In order to remove redundant data in the SQL schema, a single table PERSON is created and each attribute of Student is rolled-up and represented by a column in this table. In order to differentiate different kinds of person (of which a student is just one) a new column PERSON_TYPE is added to the table PERSON. A value of "S" for PERSON_TYPE indicates that a row represents data about an object of the class Student.

In the first case a mapping must indicate that for an object of the class Person, each value of the attribute phoneNumber is stored in (and retrieved from) a separate row in the table PHONE_NUMBER. The mapping must also specify the join between the table holding the rest of the Person class details and the table PHONE_NUMBER. In the second case a mapping must indicate that data representing the state of an object of class Student is stored in a row of the table PERSON, and that the PERSON_TYPE for that row must be "S". Less obvious is that the column PERSON_TYPE corresponds to the sub-class relationship between class Student and class Person. Each additional table and column adds to a structure problem and makes a mapping more complex for those who must implement it.

A mapping strategy is concerned with correspondence between two different descriptions of a concept. For example data for an object of class Student (Figure 1) may be sourced from rows in two SQL tables: STUDENT and ADDRESS. In order to address the ownership problem, this correspondence must be documented, published and implemented as we demonstrated using the phoneNumber example earlier. Although the detail is application specific, a mapping strategy will generally provide a mechanism for identifying, documenting, and implementing the correspondence of structure and identity between specific entries in a class model and entries in an SQL schema. Hibernate [12] uses XML whilst [20] make use of meta-data stored in SQL tables. This information forms an important part of the design of an object-relational application.

5.4. Instance

One issue that lies at the heart of ORM practice is the treatment of an object. The problem is that an object is conceptualised as an atomic unit when in practice it has a number of subdivisions. A Java object has subdivisions of structure, state and behaviour. The schema and instance levels of our conceptual framework show how these subdivisions are fragmented (Figure 2). The structure of an object is defined both in a class and an SQL schema (the ownership problem); the behaviour of an object is defined in a class; and a valid state of an object must be maintained and enforced both in-memory and across one or more rows in one or more tables, giving rise to encapsulation and identity problems.

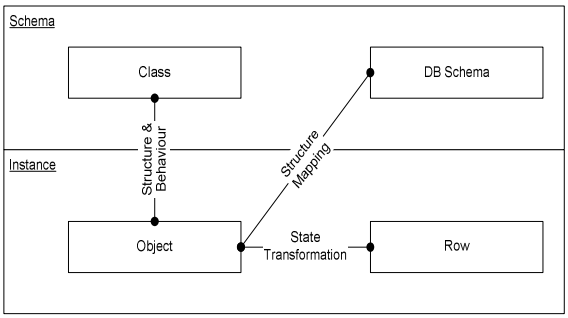

Figure 2 - Fragmentation of the subdivisions of an object

In practice fragmentation is addressed using a transformation but there are problems. Data about an object may not transform cleanly to a row of a table or an individual slot ([18],p3) (the instance problem), as we highlighted with the Student phoneNumber example. The structure of an object may not transform to a single table (the structure problem). The SQL-92 standard does not support the behavioural aspects of an object and so the behaviour of an object must be implemented within a Java class

at the schema level. The later introduction of persistent stored modules in SQL provided an opportunity for the fragmentation of behaviour. At run-time it may not be necessary to retrieve all the data about an object for a user to complete a transaction. This combined with fragmentation of the universe of objects required to complete a transaction, is the essence of the processing model problem and provides another driver for ORM transformation activities.

A programmer must reconcile fragmentation when developing an object-relational application. Fragmentation in the implementation of an object is a significant characteristic of an impedance mismatch. We label this kind of impedance mismatch an *instance mismatch* and it is addressed using an ORM transformation strategy.

The degree of state fragmentation that is characteristic of an instance mismatch is influenced by the ORM mapping strategy employed to produce the structure of an SQL schema. The design of that SQL schema is influenced by the ORM pattern strategy chosen to address a representation mismatch. Choices made within a level of our framework therefore have consequences in other levels.

An instance mismatch transformation strategy must address the consequences of fragmentation in behaviour and state. Such a strategy must deal with the processing model, encapsulation, and instance problems. The SQL standard does not provide support for the definition of behaviour within an SQL schema although relational database vendors have provided such facilities for some years. The valid state of object data may be enforced by rules defined within a class method or as a database constraint. Ambler ([2],p228) describes shadow information that is one strategy for the fragmentation of state, and scaffolding attributes that are one strategy for the fragmentation of structure.

6. Summary and Future Work

Our conceptual framework of the problem space of impedance mismatch has allowed us to recognise four different kinds of impedance mismatch. Their relationship is summarised in Figure 3. We have shown that although problems of an impedance mismatch are rooted in particular levels of our classification, through the contextualisation relationship between the levels they materialise within the instance level as problems a programmer must address in order to make an object-relational application work.

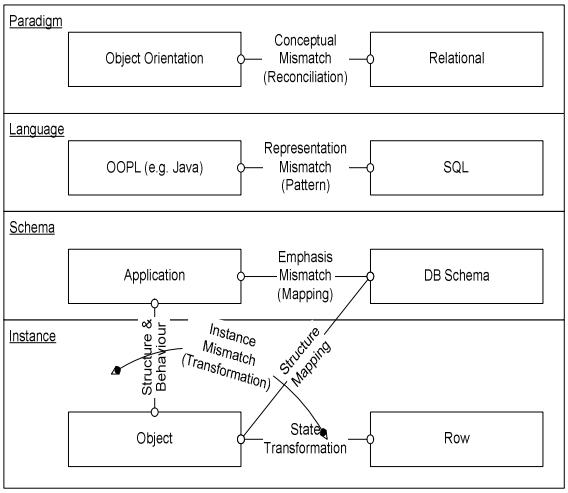

Figure 3 - Our conceptual framework of impedance mismatch incorporating ORM strategies

Our classification also distinguishes four kinds of ORM strategy within the levels of our conceptual framework:

- a conceptual mismatch provides the impetus for a reconciliation strategy at the paradigm level;
- a representation mismatch provides the impetus for a pattern strategy at the language level;
- an emphasis mismatch provides the impetus for a mapping strategy at the schema level; and
- an instance mismatch provides the impetus for a transformation strategy at the instance level.

Although choices may have consequences within other levels of our framework, each ORM strategy must only deal with issues at the corresponding level of abstraction. An emphasis mismatch strategy for example must only address issues relating to the representation of a concept. Such a strategy must work between the representations available in two languages. In order to successfully resolve an impedance mismatch problem it may be necessary therefore to address the problem at more than one level of our framework. As a result a solution to an impedance mismatch problem may involve using more than one ORM strategy. This is not the same as a single solution which crosses levels of abstraction or considers object or relational issues in isolation (see [8]). Such a solution lacks the clarity provided by our approach.

Our framework and classification provide an organising mechanism which allows us to explore issues of fidelity, integrity and completeness in the design and implementation of existing and new ORM strategy choices.

The novel perspective provided by our framework and classification produces new insights in areas such as how to exploit the strategic options available when translating a concept between paradigms; and latent issues in solutions which cross levels of abstraction. In providing an understanding of the nature of impedance mismatch, our framework will provide standards bodies, tool vendors, designers and programmers with new insights into how to address the problems of impedance mismatch both at the most appropriate levels of abstraction and in the most appropriate way.

The literature has not recognised that there are different kinds of impedance mismatch each of which is addressed using a different ORM strategy. How can we be sure therefore that ORM technologies such as JDO, Microsoft LINQ [13], and changes to the SQL standard [21] adequately address the whole problem at the correct level of abstraction and in the most appropriate way? Our framework and classification provide a means to understand the potential and consequences of these ORM technologies.

7. References

[1] E. Meijer, "There is No Impedance Mismatch (Language Integrated Query in Visual Basic 9)," presented at OOPSLA, Portland, Oregon, 2006.

[2] S. W. Ambler, *Agile Database Techniques - Effective Strategies for the Agile Software Developer*: Wiley, 2003.

[3] T. Neward, "The Vietnam of Computer Science" http://blogs.tedneward.com/2006/06/26/The+Vietnam+Of+Computer+Science.aspx on 6th February 2007

[4] A. M. Keller, R. Jensen, and S. Agarwal, "Persistence Software: Bridging Object-Oriented Programming and Relational Databases," presented at ACM SIGMOD international conference on management of data, Washington, D.C, 1993.

[5] F. Marguerie, "Choosing an object-relational mapping tool" http://weblogs.asp.net/fmarguerie/archive/2005/02/21/377443.aspx on 14th November, 2007

[6] S. Holder, J. Buchan, and S. G. MacDonell, "Towards a Metrics Suite for Object-Relational Mappings," *COMMUNICATIONS IN COMPUTER AND INFORMATION SCIENCE*, vol. 8, pp. 43-54, 2008.

[7] R. Biswas and E. Ort, "The Java Persistence API - A Simpler Programming Model for Entity Persistence" http://java.sun.com/developer/technicalArticles/J2EE/jpa/index.html on 25th September 2007

[8] M. L. Fussell, "Foundations of Object Relational Mapping" http://www.chimu.com/publications/objectRelational/ on 25th September 2007

[9] U. Hohenstein, "Bridging the Gap between C++ and Relational Databases," presented at European Conference on Object-Oriented Programming, 1996.

[10] W. Keller, "Mapping Objects to Tables: A Pattern Language," presented at European Conference on Pattern Languages of Programming Conference (EuroPLoP), Irsee, Germany, 1997.

[11] R. Lammel and E. Meijer, "Mappings Make Data Processing Go 'Round: An Inter-paradigmatic Mapping Tutorial," *Lecture Notes in Computer Science*, vol. 4143, pp. 169-218, 2006.

[12] Hibernate www.hibernate.org

[13] J. Schwartz and M. Desmond, "Looking to LINQ" http://reddevnews.com/features/print.aspx?editorialsid=707 on 23rd October 2007

[14] S. Ambler, "The Design of a Robust Persistence Layer for Relational Databases" http://www.ambysoft.com/downloads/persistenceLayer.pdf on 10th May 2007

[15] J. J. v. Griethuysen, "Concepts and Terminology for the Conceptual Schema and the Information Base," in *ISO/TC97/SC5/WG3*, ISO, Ed. New York: ISO, 1982

[16] P. Coad and E. Yourdon, *Object Oriented Analysis*: Yourdon Press, 1990.

[17] E. F. Codd, "A relational model of data for large shared data banks," *Communications of the ACM*, vol. 13, pp. 377-387, 1970.

[18] D. Kalman, "Moving forward with relational: looking for objects in the relational model, Chris Date finds they were there all the time," in *DBMS*, vol. 7, 1994, pp. 62(6)

[19] E. Meijer and W. Schulte, "Unifying Tables, Objects, and Documents" http://research.microsoft.com/~emeijer/Papers/XS.pdf on 21st August 2007

[20] J. Sutherland, M. Pope, and K. Rugg, "The Hybrid Object-Relational Architecture (HORA): an integration of object-oriented and relational technology," in *ACM/SIGAPP symposium on Applied computing: states of the art and practice*. Indianapolis, Indiana, United States: ACM Press, 1993

[21] A. Eisenberg and J. Melton, "SQL: 1999, formerly known as SQL3," *SIGMOD Record*, vol. 28, pp. 119-126, 1999.