ISSN 1744-1986

The Open University

Technical Report N° 2007/10

Object-Relational Implementation Constructs: The role of a Structured User Defined Type in the resolution of an Object-Relational Impedance Mismatch

Christopher J Ireland

29th June 2007
Student Research Proposal

Department of Computing
Faculty of Mathematics and Computing
The Open University
Walton Hall,
Milton Keynes
MK7 6AA
United Kingdom

http://computing.open.ac.uk

ISSN 1744-1986

The Open University

Technical Report N° 2007/10

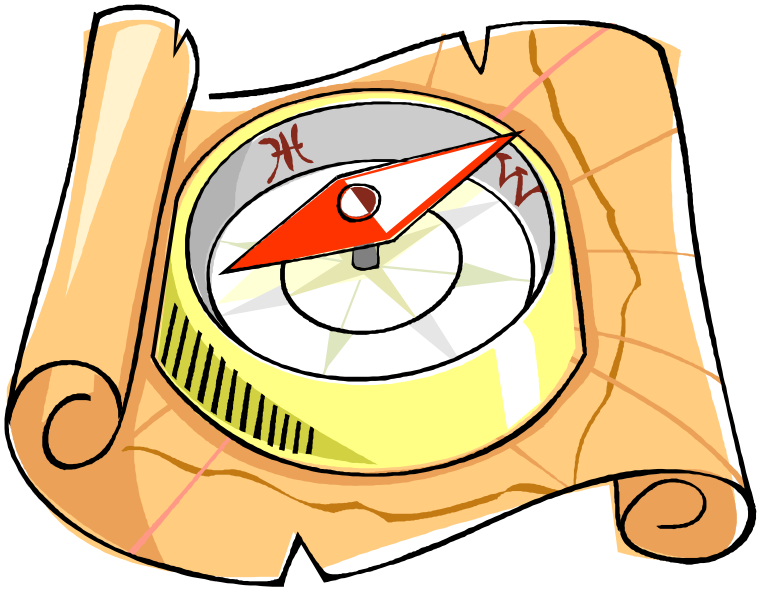
Problem-oriented software engineering

Christopher J Ireland

29th June 2007

Department of Computing
Faculty of Mathematics and Computing
The Open University
Walton Hall,
Milton Keynes
MK7 6AA
United Kingdom

http://computing.open.ac.uk



Object-Relational Implementation Constructs:
The role of a Structured User Defined Type
in the resolution of an Object-Relational Impedance Mismatch

Prepared by: Mr Chris Ireland

For: Dr Leonor Barroca
Dr Neil Smith

Cc: Mr Mike Newton
Dr David Bowers
Dr Kevin Waugh

Date: 15th March 2007

Version: 1.3 – Post Assessment Amendments

Amendment History

Date	Version	Who	Details
5th April 2007	1.1	CJI	Changes to section 4 to reflect feedback from assessment. Specifically to include an improved definition of scope.
18th April 2007	1.2	CJI	Section 4 feedback from Mike Newton.
25th April 2007	1.3	CJI	Section 4 emphasise possible complexities of a SUDT based implementation.

Table of Contents

Table of Figures

Table of Tables

1. Introduction

1.1 Purpose of the paper

The goal of the document is to demonstrate my competence to complete a PhD successfully[1]:

- my ability to comprehend the research discourse, including the ability to read well and critically and to make sense of those sources;
- my ability to write in an appropriate academic style;
- my ability to frame and situate a compelling research question – including demonstrating my ability to articulate why the question is worth pursuing;
- my ability to design a feasible, viable and interesting programme of research, including:
    - choosing and justifying an appropriate research methodology;
    - understanding how my research will contribute to the existing body of theory and knowledge;
    - planning a sequence of research activities, allowing for contingencies, and managing the research process via a plan of work;
    - understanding the limitations of the work I have planned;
- my ability to take some part of that research to completion; and
- my ability to achieve rigour, and to take care to justify or substantiate my choices, assumptions, and assertions.

1.2 Intended Audience

The intended audience of this proposal are:

1. Assessors with a view to making recommendations on its suitability as a viable PhD research project;
2. Supervisors with a view to assessing progress; project dependencies and other organisational issues; and

[1] These goals are taken from http://phdskills.open.ac.uk/static_page.php?page=33.

3. Peers with a view to assessing relevance to their work and opportunities for synergy.

1.3 Structure

The proposal is structured as follows:

Section 2: I introduce my field of research, contextualising research at the juncture of number of significant issues in modelling; programming; and relational databases.
Section 3: I critically assess current research in the field and identify a significant gap which is worthy of further research;
Section 4: I state my research question which aims to addresses the gap; and
Section 5: I present a research plan which will deliver an understanding which is necessary to close the gap.

2. The Research Context

The translation of data about a real-world phenomenon between different representations is a prevalent problem in software development. I refer to this problem as impedance mismatch.

The problem occurs at all stages of software development including the representation of: user requirements as a conceptual model; a conceptual model as application code; and program data within a database. Finding a resolution to the problem adds time and effort to a software development project.

I refer to data about some real-world phenomenon as a concept. Concepts are collectively represented on a conceptual model. When a concept is described using a particular language I call this a representation. A concept and its many representations are collectively abstractions of a real world phenomenon.

A concept must be represented within a software system if that system is to be able to process data about the phenomenon it represents. The correct and accurate description of a concept is fundamental to the success any software system but depends on a number of factors (Figure 1).

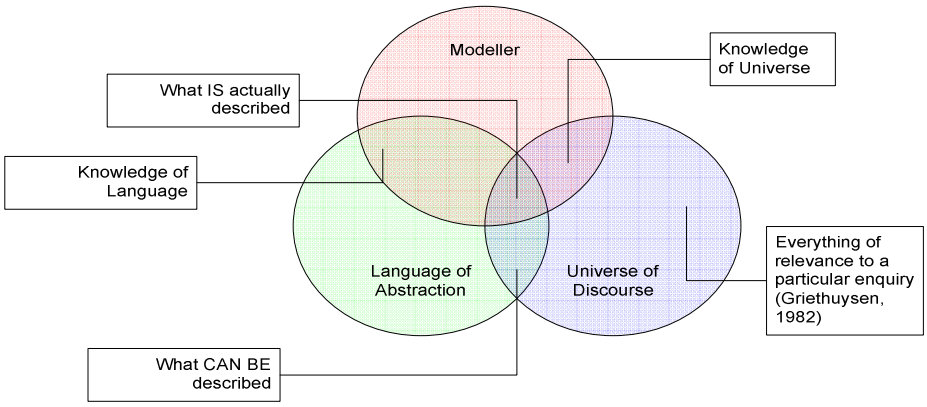


Figure 1 - The Semantic Potential of an Abstraction

2.1 The Importance of Impedance Mismatch

Impedance mismatch is a situation which occurs when there are two different representations of the same concept.

There are many reasons why two representations would differ. Each representation may have been developed at a different time; using a different language; or with a different insight or purpose on the part of its creator. Company standards and the use of 3rd party packages and libraries may also dictate the use of a certain representation.

The many representations reflect the different needs and perspectives of the stakeholders involved in a software project. Stakeholders determine what is important and include analysts, database administrators, programmers and other database users.

The problem is exacerbated by the wide variety of languages which are used to express a concept. Examples of the many languages from the domain of software development include the Unified Modelling Language (UML), Java, C++, SQL and XML. Each language provides an opportunity to produce a different representation. Each additional representation increases the effort required for reconciliation.

Impedance mismatch is not a new problem in the field of software engineering. It will continue to be a problem whilst there is more than one way to describe a concept.

2.2 A Question of Paradigm

At the heart of an impedance mismatch problem is a difference of paradigm. A paradigm is a particular way of viewing the world. The use of different paradigms leads to different representations of the same concept. Object-orientation and relational are two important software development paradigms.

The object-oriented paradigm considers the world as a network of interacting objects. Each object represents data about a concept. An object is an instance of a single class.

A class describes the characteristics of one or more objects. A characteristic may be an attribute which represents some property of a concept; or some processing in the form of a method which characterises some behaviour of that same concept.

A class may be part of a hierarchy with other classes. A subclass has at most one parent. A subclass is important both in conceptual modelling and program design but it may have many possible semantics (Brachman 1983). These possibilities create further potential for an impedance mismatch problem.

A subclass forms an important aspect of an object-oriented representation. On a conceptual model a subclass provides benefits such as intellectual manageability (Smith and Smith 1977), p105); a clearer specification of the semantics of a phenomenon; and incremental design. In an object-oriented program it provides for amongst other things software re-use and incremental development. The representation of a subclass will form an important aspect of my research.

The relational paradigm is based on a view of the world as a mathematical relation (Codd 1970). A relation is a set of n-tuples where n is the degree of the relation. A relation can be used to represent a concept. Often a relation is represented as a table with columns for each tuple. A row in such a table represents data about a concept.

The relational paradigm is grounded on the semantics of a set. This contrasts with the network and hierarchical views of the object-oriented paradigm. A concept must fit into a representation using relations (An, Borgida et al. 2006), p23), (Hainaut 2006), p109, section 5)). Whilst the relational model has a defined mathematical basis, it has been noted that such an approach leads to an obfuscation of semantics (Schmid and Swenson 1975), p220).

The semantics of a software development language are grounded within a particular paradigm. The UML and Java are languages used to describe a representation within the object-oriented paradigm. SQL is a language used to describe a representation (or schema) within the relational paradigm.

2.3 An Issue of Transformation

The reconciliation of a difference between two representations will involve some form of transformation.

Whilst transformations are necessary to address an impedance mismatch problem, they are not a new field of study (Lammel and Meijer 2006). They also form a cornerstone of the Model Driven Architecture (Djuric, Gasevic et al. 2006).

A transformation is a function which takes a representation expressed in a source language and produces another representation in a target language.

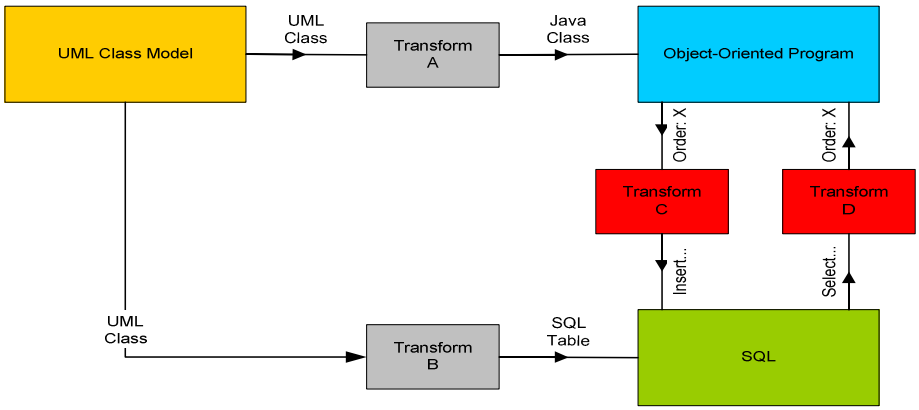


Figure 2 - The Transformation of a Representation

Figure 2 provides an example of a number of transformations. Transform A takes a UML class as input and produces a Java representation of that same class. Transform B produces a different representation from the same class using SQL.

A transformation is necessary when a representation described using one language must be converted into a representation in another language. There are many examples of transformations within software development including the transformation of: an entity on an entity-relationship model into an SQL table; a UML class into a Java class (Figure 2, Transform A); and a relationship on an entity-relationship model into a primary/foreign key pair.

In Figure 2, data about a real-world Order "X" is shown as being represented both as a Java object (Order:X[2]) and as a row in an SQL table (ORDER[3]). The impedance mismatch problem is a direct consequence of these two different representations of a UML class. Transforms C and D represent the program code required to maintain data about an object as it passes between the representations produced by Transforms A and B.

At their most basic, transformations C and D will involve a translation of the languages used to describe a concept or the names of attributes and columns. They can be far more complex ((Hainaut 2006), p109, section 5).

A transformation must also possess other characteristics: it must be repeatable over time although what is transformed may change; in order to avoid misinterpretation it must preserve the semantic integrity of the source representation in the target; and it must therefore be a non-loss transformation.

A *faithful transformation* is a transformation which preserves the semantic integrity of a source representation in a target representation. Two representations exhibit semantic integrity when they support exactly the same facts about some real world phenomenon.

A transformation must be faithful if those involved in software development are to correctly interpret a representation and both avoid an impedance mismatch problem and accurately define one or more data integrity rules. The definition of a faithful transformation is therefore an important aspect of software engineering.

A more stringent requirement, for a *viable transformation,* is that it must not only be faithful but also addresses an impedance mismatch problem. It does so by producing a

[2] For convenience and clarity in text we have used the UML notation for naming an object. This name is of the form Class:Object. Java would use a reference to an Order object.

[3] In order to distinguish from any other use, we will use capital letters for the name of an SQL table.

representation which minimises, if not removes the effort required to implement transforms C and D on Figure 2.

2.4 Objects and Relations

The separation of the management of data from its use within a program provides many opportunities for an impedance mismatch problem. The problem is typified by the support for complex abstractions provided by object-oriented programming languages but not by SQL.

There have been a number of different approaches to addressing an impedance mismatch problem between an object-oriented program and a relational database. The resolution has typically involved additional software layers and third party tools in which mappings are performed (Hohenstein 1996).

Impedance mismatch may be resolved by addressing a difference of paradigm between two languages. If both languages support the same representation then no transformation will be necessary. One solution is a completely new database language.

Object-oriented databases (OODB) were an attempt to remove an impedance mismatch problem for object-oriented applications. Although suited to applications involving complex data structures (Zhang, Ritter et al. 2001), OODB lacked the ability to handle the high volume transaction throughput now provided by their relational counterparts (Stonebraker, Rowe et al. 1990), p31).

Another solution involved a multi-paradigm approach. Object-Relational SQL (OR-SQL) was introduced in SQL: 1999 (ISO 2003). It is a language influenced by both the object and relational paradigms.

The cornerstone of the object-model employed within OR-SQL is the structured user defined type (SUDT). A SUDT has a lot in common with a class. A SUDT has attributes and methods and may be combined with other SUDT into a type hierarchy. For clarity, I will use the term SUDT (or type) when referring to SQL. In the context of the UML and Java I will use the term class.

A SUDT extends the catalogue of data types available within a relational database. As such a SUDT may be used as the basis for the definition of a column or a typed table. A parallel may be drawn between the rows of a typed table and the extent of a class.

2.5 Object-Relational Implementation Constructs

The field of object-relational implementation constructs (ORIC) is concerned with resolving the problem of an impedance mismatch between object and relational technologies (Figure 3). My research aims to achieve this by understanding and describing faithful and viable transformations.

The UML is important because it provides a standard notation (Juric 1998) for the description of a conceptual model, and a common basis for program and database design. Java is a popular object-oriented programming language. A significant question is whether or not OR-SQL is part of the solution to the problem (Figure 3)?

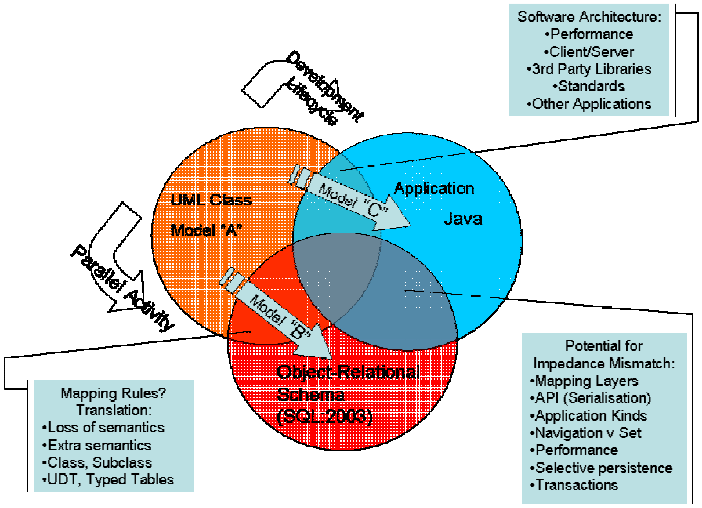


Figure 3 - The Potential for an Impedance Mismatch in the Field of Object Relational Implementation Constructs

In the context of a software development lifecycle, a conceptual model (Figure 3, Model A) expressed using the UML is transformed into two representations: a Java program (Model C) and an object-relational database schema (Model B). Given that both are object-based representations one might expect there to be no impedance mismatch problem.

The structures used in the Java program and the OR-SQL schema address different issues. The program will include subclasses for which no persistence is required such as network I/O, user interface and analytics. The OR-SQL schema may have a different structure (Mok and Paper 2001); the potential for different semantics; and possibly a design influenced by other applications with a different emphasis.

Other important questions include what aspects of a UML class model must be represented in an SQL schema? How should each aspect be represented; and what are the implications of a given representation for the resolution of an impedance mismatch problem?

The faithfulness and viability of a transformation directly influences whether or not the problem of an impedance mismatch is resolved. ORIC is therefore an important field of study for the viability of an object-relational application.

2.6 Summary

Impedance mismatch is a problem for those developing object-oriented applications against a relational database.

Research in the field of ORIC is concerned with the resolution of any impedance mismatch between an application developed using an object-oriented programming language and OR-SQL.

At the heart of the problem is a difference of paradigm. Overcoming the difference will involve some form of transformation between two representations: one based on an object; the other based on a relation.

A faithful transformation preserves the semantic integrity of a source representation in a target. A viable transformation resolves an object-relational impedance mismatch problem. Together they reconcile the requirements of a number of key stakeholders including programmers, analysts, database administrators, and other database users.

An important contribution of research in the field of ORIC is to the understanding and description of what constitutes a faithful and viable transformation.

In the next section I critically evaluate research in the field. I also highlight a specific issue which lies at the heart of one attempt to resolve an impedance mismatch problem.

3. Related Work

The single most important contribution of OR-SQL to the description of a relational database schema is the type (or SUDT), with possible subtypes. The SUDT is the foundation of the OR-SQL object model and provides a parallel with a UML class. The key to maximising the benefit of OR-SQL and addressing one source of an impedance mismatch problem is in understanding how best to make use of this support for subtype in the context of a UML design.

In this review of the literature there is therefore a useful distinction to be made between research work on object-relational database design and research work on impedance mismatch problems. Given the importance and influence of UML on the field it is important first to examine research in that area.

3.1 The Semantics of UML

Many authors have examined the semantics of the UML. Some writers (Juric 1998) have focused on English language interpretations of the underlying meta-model whilst others (Morris and Spanoudakis 2001) have examined the physical symbols used to represent a class and a subclass in diagrammatic form.

The semantics of the UML determine *what can be represented* but they are at a different level from those of a real world phenomenon as understood by a modeller. The semantics of the UML provide few clues as to the semantics intended by a modeller. Consider that each modeller will conceptualise the real-world in different ways and thus the same phenomenon may be represented in a number of different ways on a UML class model. What *is* described on a UML class model is also only a subset of what *can* be described (Figure 1).

It is important to recognise that the purpose of a UML class model is contextual. At various stages of a software development lifecycle it provides an understanding of a particular universe; a structure for an object-oriented program; and a model of data.

The reconciliation of these many roles will have a significant influence on the viability of a transformation.

There are a number of options for representing subclasses in the UML which have lead to different interpretations and confusion. Researchers in the field have addressed this from a number of angles:

- (Eichelberger 2003) proposes a set of aesthetic criteria for a class diagram, but stops short of calling for changes to the standard and relies instead on spatial relationships and the size of diagram components to carry semantic information; and
- (Purchase, Colpoys et al. 2001) examined four notations used to represent the specialisation relationship[4] on a UML class diagram in order to assess which forms best suited human comprehension.

In order to validate a UML class diagrams and overcome the imprecise semantic information provided by a graphical notation, others have attempted formal specifications of the UML using Z ((Shroff and France 1997), (Evans 1998)) or Hierarchical Predicate Transition Nets (He 2000). An understanding of such formal representations of the underlying rules of UML is typically out of the reach of IT practitioners. How would they then assess the completeness and validity of a design? Such a formal representation would form a valuable completeness check perhaps augmenting UML diagram support in a CASE tool.

(Evans 1998) present a set of rules for transforming a UML class diagram from one form to another. Using these rules they attempt to derive some conclusions about the system being modelled. They conclude that UML class models "do not provide the precision that is often required when reasoning with software systems", one notable point being the lack of any diagrammatic indication of dependencies between attributes and classes.

[4] Specialisation is the name we use for the relationship between a class and a subclass.

The lack of precision raises questions about whether a formal representation of the semantics of the UML will help in assessing the semantics of a class diagram. The UML language specification (OMG 2004) specifies the syntax of the language but not the semantics of a particular class diagram. These semantics depend on a number of factors (Figure 1). In order to understand the faithful transformation of a subclass it is necessary to understand how and why a subclass is used.

It is not clear how a subclass may be used. The subclass relationship is referred to ((Brachman 1983), (Marcos and Cavero 2002) and ((Hainaut 2006), p117)) as is-a, a-kind-of, subclass, generalisation, specialisation, subset and inheritance hierarchy. Each has its own semantics. In the field of semantic relationships it has been noted that such a lack of precision is detrimental to the progress of science (Siau 2004). Certainly, these different terms do not help efforts to produce a formal basis for the faithful transformation of a subclass.

Even if a modeller expresses their understanding clearly, the UML itself may limit the precision of that expression. (Atkinson and Kuhne 2002) examined deficiencies in the meta-model underlying the UML, and proposed a separation of the logical and physical dimensions of the meta-model. They emphasise that it is important to preserve what they call “strict meta-modelling”. This research is essential for ensuring that a model is consistent. It is also a prerequisite for any potential transformation. The consistent representation of a concept will help to produce a consistent understanding.

It is evident that the semantics of a UML subclass are not precisely defined. Little consideration is given to the actual understanding expressed by a modeller. A subclass may be introduced at any time for any one of a number of reasons. This flexibility is both a strength and a weakness of the UML.

The lack of precision poses a threat to the implementation of a UML design. In understanding and defining a faithful transformation from a UML class model into an OR-SQL database schema, it is important to understand the role of a subclass.

3.2 Object-Relational Database Design

The UML has proved popular for the modelling of IT systems. Its popularity has lead researchers to propose extensions which allow for the design and representation of object-relational databases.

(Naiburg and Maksimchuk 2001) presents a case study on the use of UML to support database design but they do not consider options for translating the resulting model into SQL or OR-SQL. This contribution is valuable if only for its contextualisation of the database design process using UML. It highlights the contribution a number of stakeholders identified in section 2.

(Marcos, Vela et al. 2001) propose extensions to the UML in support of object-relational database design as distinct from the design of a relational database. These extensions are in the form of stereotypes, tagged values and constraints. They allow a designer to indicate how aspects of a class model should be implemented in an object-relational schema. They do not provide a rationale for why a subclass should be transformed in such a way.

In a subsequent paper (Marcos, Vela et al. 2003) propose a methodology for database design in which they suggest a mapping from a UML class model to an OR-SQL schema.

Although the Oracle ORDBMS is used as an example, in neither paper do Marcos, Vela, et al. consider:

1. what the options are for transforming a subclass on a UML class model into an OR-SQL schema; to what degree is semantic integrity preserved; how does one choose between them; and what are the consequences of any given choice; and

2. in what circumstances a traditional SQL approach may be more appropriate than OR-SQL approach or where they may be used in combination to provide a more viable and faithful representation of a subclass.

So far the transformations presented in the literature have been informal; primarily suggestions of mechanisms for *emphasising* rather than selecting between object-relational transformations.

(Mok and Paper 2001) were the first to suggest an approach to ORDB design which focussed on the capabilities of OR-SQL. They provide one option for transforming a UML class model with subclasses to an OR-SQL schema.

They propose two algorithms: the first (Mok and Paper 2001), p3) for removing ambiguity[5] from a UML class diagram. The second, which generates a nested normal form (Mok, Ng et al. 1996), recognising functional dependencies.
(Grant, Chennamaneni et al. 2006) argue that this approach:

1. Does not preserve the structure of the original UML class model; and
2. Is more rigorous than those presented before but requires developers to have a "good" mathematical background in order to comprehend the nature and implications of a transformation.

A UML class model forms the basis of both an application design and a database design. Any transformation which moves the structure of a database schema away from a UML class model has potential to exacerbate an impedance mismatch problem.

A faithful transformation must maintain the semantic integrity of a UML subclass in an OR-SQL representation. The removal of ambiguity proposed by (Mok and Paper

[5] The ambiguity is in the form of semantically overloaded elements. A semantically overloaded element is one which has many different roles on a model. An example from Mok, W. Y. and D. P. Paper (2001). On Transformations from UML Models to Object-Relational Databases. 34th International Conference on System Sciences, Hawaii IEEE.,p4) is that of a person being both a member and the manager of a department.

2001) provides a useful contribution to an understanding of a faithful transformation. Further work is necessary to determine if making a UML class model fit into a representation using a nested normal form also produces a viable transformation.

Whist rigour is important; a viable transformation must avoid the mismatch problem faced by software developers working with previous generations of SQL (Neward 2006)**,** (An, Borgida et al. 2006), p23), (Hainaut 2006), p109, section 5). A failure to do so will necessitate the adoption of costly compensation strategies in an application.

Unlike previous versions of the SQL standard, OR-SQL allows a schema designer to define behavioural aspects of a subtype. These behavioural aspects are implemented in the form of methods.

(Mok and Paper 2001) suggest that UML state chart diagrams may be used to generate database triggers. Whilst this is one transformation option they omit to consider or contrast this with other possible options using a method or database procedure. If one is to produce a faithful transformation, it is important to understand the implications of these other possible implementations of the behaviour of a subtype.

Other researchers have considered the transformation of a UML model to an object-relational databases schema but have not focussed on subclass.

(Soutou 2005) describes the practical options for implementing referential integrity within an OR-SQL implementation (notably Oracle 9i and 10g). This work will benefit those trying to implement an OR-SQL model as it describes a range of options but it does not consider the role of a subtype. It is also of little benefit to those wishing to make faithful transformation choices in the first instance.

(Soutou 2001) earlier examined options for implementing a number of different kinds of conceptual relationship using OR-SQL. Their focus was primarily on the impact of association cardinalities (1:1, 1-N; N-M) on the selection of a transformation.

They highlight a number of additional options provided by OR-SQL over SQL and suggest a generic solution for representing cardinalities. They do not make recommendations for selecting between different transformations in the context of a UML class model; or examine the impact of a subclass on these relationships.

In their work on relationship transformation (Soutou 2001) did not consider specialisation. This omission may be justified on the basis that specialisation is a structural relationship between classes and not one between instances. The focus of the (Soutou 2001) paper was on relationship semantics at the instance level.

3.3 Impedance Mismatch

There are a number of approaches for resolving the practical issue of an impedance mismatch particularly with respect to a subclass. Typically the solution involves some form of mapping between two representations: the one used in an object-oriented program and the one used in a relational database.

When a mapping is performed in an application, the application is exposed to any changes in a relational representation. The application (and there may be many of them) must carry the overhead of maintaining the mapping both to *and* from a relational database.

When a mapping is performed in middleware an application is less exposed to the underlying data storage. The introduction of another layer of software increases complexity; may lead to a reduction performance; *and* a mapping still needs to be maintained. Depending on the nature of a change to a data structure, the change to a mapping will be more or less trivial.

(Hohenstein 1996) propose a mechanism for mapping a relational schema to a C++ class model. Theirs is an example of the first approach.

They reverse engineer a relational database schema into C++ class representation. Taking an *existing* relational database as given they attempt to infer semantic information. They use this information to produce a number of C++ classes which

have the equivalent semantics and which are used to map between an application and a database. Inference is necessary because the semantic information within a conceptual model is dispersed around an SQL schema (Lammel and Meijer 2006), p197), (Hohenstein 1996), p406)).

(Hohenstein 1996) is one of many approaches (Keim, Kriegel et al. 1993), (Keller 1997), (Grehan 1998), (Sutherland, Pope et al. 1993) and (Snyder and O'Connor 2005) which attempt to combine object-oriented applications with relational databases. Although this and other work (for example (Ambler 1995) and (Cabibbo and Porcelli 2003)) may be beneficial for integration with legacy databases (Ambler 1995) and (Hohenstein 1996) could not concern themselves with the opportunities provided by OR-SQL as their work pre-dates the object-relational standard. OR-SQL is an evolution of SQL and so this work may continue to have some relevance depending upon the nature of a viable and faithful transformation.

The adoption of an object paradigm suggests that OR-SQL has the *potential* to reduce the impedance gap between an object-oriented program and a relational database. (Zhang W. 2001) describes what they still see as “a considerable gap” between the object-*relational* paradigm and the object-*oriented* paradigm. They highlight that a key problem is the lack of a single object-model between application and database but they do not provide a solution.

So far we have considered research relating to the intent (structure) of a class model. There are further issues when we consider the extent (the representation of an object).

Examining the integration of an object-oriented application and an OR-SQL schema, (Zhang and Ritter 2001) identify that there are still two different representations of an object. If nothing else data about an object in an object-oriented application is still separate from data about that object in an OR-SQL database. The fact that there are two representations of an object lies at the heart of an impedance mismatch problem.

An object within an executing object-oriented application *is a different object* to that in an object-relational database. This may be contrasted with an OODB in which an application has a proxy for an object in a database. One way of overcoming the problem in a relational database is to use serialisation. Serialisation results in three separate representations[6] of data about the same object. If it were the same object then a copy mechanism such as serialisation would be unnecessary.

Without a common basis for the design of these two representations it is possible that each will promote different semantics. A UML class diagram provides such a basis but that must fulfil a number of roles.

A viable transformation of a UML subclass produces an OR-SQL representation which is equivalent or at least not contradictory to that used in an object-oriented program. An assessment of the degree to which a transformation addresses the problem of an impedance mismatch is essential for an understanding of the viability of a transformation.

3.4 Summary - A Focus on Subtype

The introduction of OR-SQL provided an opportunity to bridge the paradigm gap between an object-oriented application and a relational database. This opportunity to address the problem of an impedance mismatch does not appear to have been taken.

A SUDT forms the keystone of the OR-SQL object model. My research so far has identified that there is no published research on the contribution of a SUDT to the resolution of an object-relational impedance mismatch.

It is not yet clear the degree to which the semantic integrity of a UML subclass and a SUDT are similar. There are questions as to how a SUDT may be used most effectively in an OR-SQL schema. It is also not clear how to faithfully transform a

[6] The three representations are: (1) the source object structure; (2) the target object structure; and (3) the serialised object structure.

UML subclass to a SUDT in an OR-SQL schema in order to avoid an impedance mismatch.

A UML class model provides a description of both data and processing. This description forms the basis of both a Java program and an OR-SQL schema. In order to realise the potential of a SUDT in OR-SQL it is therefore necessary to understand the faithful and viable transformation of a UML subclass. Such understanding will provide an opportunity to resolve an object-relational impedance mismatch; and a sound foundation for the use of a model driven approach to the development of object-relational systems.

4. The Research Question

In this section I provide a definition of my research question including: its scope; the method I will use to achieve a viable outcome; and how I believe it contributes to the sum of knowledge.

4.1 Definition of Scope

The concept of an impedance mismatch provides focus for the exploration of the capability and usability of object-relational features. Specifically my research examines the contribution of SUDTs to addressing the problem of an object-relational impedance mismatch.

An impedance mismatch occurs when a mapping between two representations is made difficult for a programmer due to conceptual differences. These differences exist because of an unclear; ambiguous; or non-reversible mapping between two representations.

An object-relational impedance mismatch occurs as a result of the different representations of data used within an object-oriented program and a relational database schema. An object-relational impedance mismatch is not an insurmountable problem. To date, solutions have lead to obfuscation of semantics which increases project risk due to misunderstanding; and extra work for a programmer which extends project timescales. My research is concerned with investigating one possible solution to the problem of an object-relational impedance mismatch. Such a solution would reduce both project risk and timescales.

In practice approaches to the problem of an object-relational impedance mismatch have compromised the representation used within a program or a relational database schema. The use of a mapping layer such as Hibernate within a Java program is one example of an approach used within a program.

A mapping layer provides an object-oriented program with object-based interface into a relational database. A programmer is insulated from the need to understand both

how an object is stored and the SQL language used to maintain that representation. The details of how an object is represented in a relational schema are moved to a mapping layer, but the production of a mapping is still a necessary part of software development. This approach is particularly useful when a Java program must be developed against a legacy relational schema. In such a situation it is not possible to compromise the database structure without affecting other applications.

If the design of a relational schema is not constrained by a legacy data structure, changes may be made to an SQL representation to bring it closer to an object-oriented representation (Hohenstein 1996). In one approach a table is created for each leaf class in a class hierarchy; in another approach a single table is used to represent an entire class hierarchy. Neither approach removes the problems of an object-relational impedance mismatch and may cause other problems due to obscure semantics and potentially introduce data integrity issues. A table-per-leaf-class approach results in columns representing parent class attributes being duplicated across tables. This increases the likelihood of data duplication and the potential scope of maintenance activities. A table-per-hierarchy approach results in a NULL value for a column for an object which does not have a particular attribute. It is also necessary to introduce shadow information (Ambler 2006) such as an additional column to classify a row.

None of the above solutions removes the need to change either the Java or SQL representation. The result is more work for those developing an object-relational application and increases the risk of error due to misunderstanding.

With the introduction of SUDTs the designer of an SQL schema can now produce a structure which appears to parallel that of a class model used in an object-oriented program. SUDTs may be used in many ways to provide a faithful representation of a UML class model, but they increase the complexity of a relational database schema which leads to difficulties deciding the most appropriate way in which to use them.

Contrasting the approaches of (Mok and Paper 2001) and (Marcos, Vela et al. 2001) to the design of an object-relational schema; each suggests a translation to a different

SUDT based representation from the same UML class model. The approach of (Mok and Paper 2001) suggests a structure involving the nesting of SUDTs and in which the database representation of an object has no identity. The approach of (Marcos, Vela et al. 2001) suggests a structure involving references between objects as rows in one or more typed tables. In this case a database object has an identity but this may be different from that in an object-oriented program. There is therefore no single mapping to a SUDT based representation from a UML class structure. The use of SUDTs increases the complexity of an object relational database schema and leads to a range of implementation options. The range of options means that the use of SUDTs is a particularly rich area for investigation.

Potentially a SUDT provides a mechanism to resolve a difference of representation between a class used in an object-oriented program and the representation used in an SQL schema. The potential reduction in an impedance mismatch provides one possible focus for examining the many ways in which an SUDT may be used (Figure 4). It also allows me to assess the use of SUDTs independently of product implementation concerns such as efficiency.

At the heart of my research is an assessment of whether an SQL representation involving a SUDT provides a solution to the problem of an object-relational impedance mismatch by removing a difference of representation. As a consequence my research will exclude any transformation which does not produce a representation involving a SUDT but it may include other OR-SQL features. In order to assess the improvements offered by a SUDT it will still be necessary to produce non-SUDT based representations for comparison.

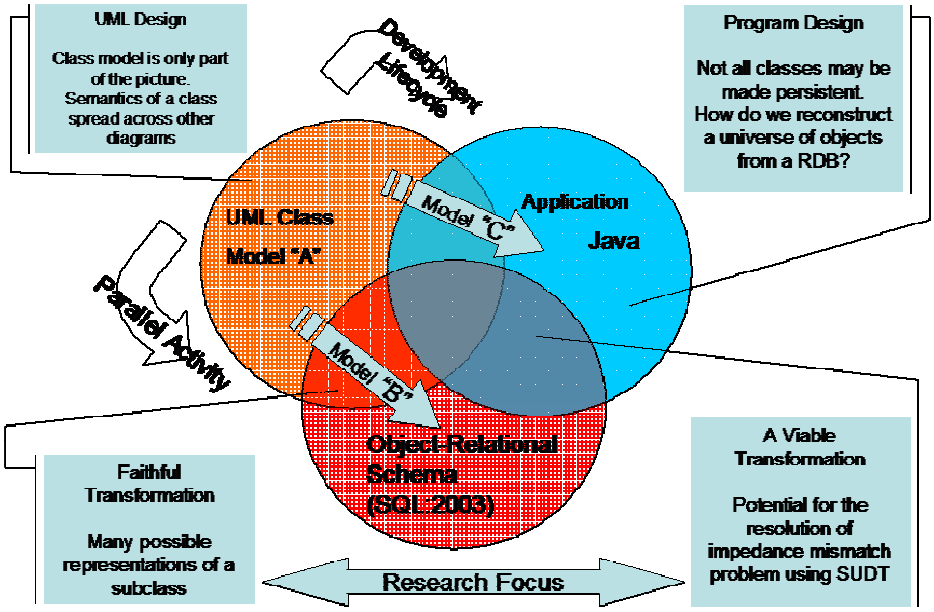


Figure 4 - The Scope of my Research Question

A UML class model provides the starting point for an object-oriented program, an OR-SQL database schema and therefore my analysis.

The UML supports a representation which allows for a subclass to have multiple parent classes. The Java and OR-SQL representations allow only a single parent. My focus is on one solution to the problem of an impedance mismatch between a Java program and an OR-SQL schema. As a result I will represent a case study using a UML class model on which a subclass has at most one parent. Support for multiple parents will not be considered.

An understanding of the many possible semantics of a subclass on a UML class model is essential to assessing its role in the delivery of a Java program and an OR-SQL schema. A complete UML design also includes other models such as the state chart which augment the semantics of a subclass. Although a UML class model forms the starting point for a transformation, my research is not focussed on a UML class model itself. Rather a UML class model forms the basis for a Java program and an OR-SQL schema. My research focus is the contribution of a SUDT to the resolution of an

object-relational impedance mismatch, and therefore a UML class model need only contain sufficient information to test that contribution.

The design of a Java program and an OR-SQL schema present different demands on the structure of a UML class model. A program may require classes for which no persistence is needed and an OR-SQL schema may use a different structure for efficiency and data integrity. In assessing the contribution of a SUDT it is important that I understand these requirements.

I will develop transformations based the published OR-SQL standard (SQL: 2003). This will remove commercial emphasis and generalise the results of my research. Any further translations that may be necessary for implementation; and how best to use those options available in a specific ORDBMS are out of the scope of my research.

My research is not concerned with issues arising from a vendor specific implementation of the SQL standard. As a consequence efficiency issues such as the performance of a particular piece of SQL code are out of scope. Efficiency of representation is a concern in so far as it affects programmer comprehension.

4.2 Contribution to Knowledge

The UML and OR-SQL are both current and popular technologies. There are no formal guidelines for the transformation of a subclass from UML to a faithful OR-SQL representation using a SUDT. Furthermore there is no way to assess the contribution of an SUDT to resolving the problem of an object-relational impedance mismatch.

Although trumpeted as the “Next Great Wave” (Stonebroker 1998), in practice it would appear that little use is being made of the object-based features introduced in OR-SQL. This contrasts sharply with the popularity of object-oriented software development methods, modelling notations and programming languages.

I have identified a number of barriers to adoption including:

1. Anecdotal evidence (Appendix A) which suggests why practitioners are not making use of the support for subtype in OR-SQL;
2. Database vendors no longer emphasise support for an object-relational model; and
3. UML based software development processes have not addressed transformation issues. For one of the main texts, persistence issues are "beyond the scope of this book" (Jacobson, Booch et al. 2005), p257). Another recommends "begin with the lowest common denominator and build from there" (Naiburg, p135).

My research will make a positive contribution by providing an understanding of the contribution of a SUDT to the removal of an object-relational impedance mismatch. This will be based on an understanding of the faithful transformation of a subclass from a UML class model to an OR-SQL representation.

My proposal involves a number of elements which address gaps in the academic field:

1. A review of the current literature regarding the way a subclass may be used in UML and Java and a SUDT used in OR-SQL;
2. An understanding of the possible transformations of the many possible uses of a UML subclass into an OR-SQL representation using a SUDT;
3. The formulation of criteria for the faithful transformation of a UML subclass to an OR-SQL representation using a SUDT;
4. The formulation of criteria for the assessment of object-relational impedance mismatch specifically relating to the viability of a transformation of a UML subclass to an OR-SQL representation using a SUDT in the context of a Java program; and
5. The application of these criteria to a set of subclass transformations in order to provide guidance on the use of a SUDT in the context of a Java program to remove or avoid an object-relational impedance mismatch.

The results of my research have wider implications; (Table 1) presents the benefits of my research to other stakeholders.

Stakeholder	Underlying Issue…	Benefit…
Analyst	Whilst a UML design may be transformed into a Java program, a number of steps are necessary to produce a database schema. A relational schema may bear little resemblance to the UML class model making validation and verification difficult.	A theory for the selection of a faithful transformation of a UML subclass to an OR-SQL representation using a SUDT will aid traceability. These criteria and transformations may be incorporated into a CASE tool.
Software Developer	The resolution of an object-relational impedance mismatch problem involves extra development effort which adds no significant business value to the end-product. Mappings must also be maintained in response to changes in data structure.	To be viable a transformation must maintain compatibility between the structure used in an application and that used in a database. Understanding the faithful and viable transformation of a UML subclass to an OR-SQL representation using a SUDT, will help those developing an object-relational application to avoid impedance mismatch problems.
Database Administrator (DBA)	Implementing a subclass structure in an RDB typically obfuscates data semantics. This impacts the maintenance of data integrity.	Data integrity will be supported by a sound theoretical basis for the use of SUDT. A SUDT may also provide benefits such as the enforcement of standards and data access optimisations.
Database User	Implementing a subclass structure in an RDB typically obfuscates data semantics. This makes querying and understanding the data structure difficult and time consuming.	A SUDT makes the context far more explicit: the type of a row may be easily determined; joins are not necessary to reconstruct an object; and rows of a sub-table are automatically included in a super-table with no user intervention.

Table 1 - The Benefits of my Research for Stakeholders

4.3 Research Hypothesis

In order to understand how a SUDT can contribute to the resolution of an object-relational impedance mismatch I propose to test the following hypothesis:

In the development of an object-relational application using Java, the faithful transformation of a UML class to a representation involving a SUDT provides a viable solution to the problem of an object-relational impedance mismatch.

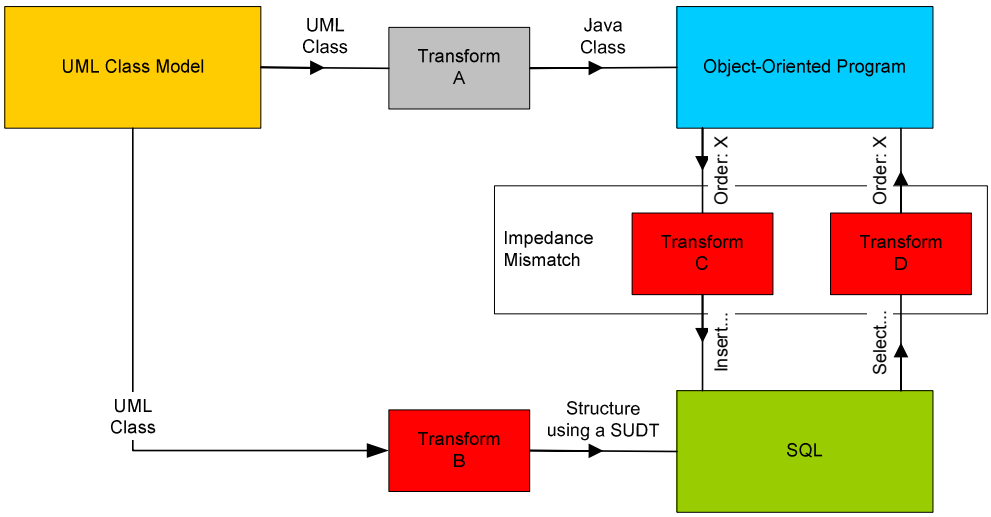


Figure 5 - The Focus of my Research Question

My hypothesis will test the degree to which a faithful transformation of a UML class to an OR-SQL representation involving a SUDT (Figure 5 – Transform B), addresses the problem of an impedance mismatch faced by a software developer (Figure 5 – Transforms C and D). Appendix C provides one example of an impedance mismatch problem.

In order to test this hypothesis a number of other questions must be asked ((Phillips and Pugh 2005), p48) including, but not limited to:

1. What are the semantics of a subclass and how can they be expressed in a UML class model, a Java program and using a SUDT on an OR-SQL schema. What are the similarities and essential differences;
2. What can, and cannot be expressed using a subclass in a UML class model, a Java program and using a SUDT in an OR-SQL schema;
3. What other UML representations are necessary to support a transformation choice;

4. What are the options for translating a UML subclass into an OR-SQL representation using SUDT;
5. How do I decide which is a faithful transformation and if there are options, how do I choose between them;
6. How do I assess the viability of a representation in the context of a Java program; and
7. How do I balance the need to resolve an object-relational impedance mismatch with the need to maintain semantic data integrity?

Points 5 and 6 above require particular attention since they are influential in ensuring that the outcomes of my analysis are measurable and recognisable, and contribute something meaningful to my research question.

4.4 The Assessment of Faithfulness and Viability

A *faithful transformation* is a transformation which preserves the semantic integrity of a source representation in a target representation. The preservation of data integrity motivates the need for faithfulness with respect to an OR-SQL representation.

The concept of faithfulness will be refined further within my research project. A precursor to the definition of faithfulness is an understanding of the many potential interpretations of a subclass highlighted by (Brachman 1983), (Meyer 1996), and (Marcos and Cavero 2002). My initial understanding of faithfulness is based on the categories of model quality assessed by (Moody, Sindre et al. 2003), section 2.2). The categories of model quality are syntactic correctness, validity, completeness and comprehension.

Syntactic correctness is fundamental to the success of a transformation. I relate the concepts of validity and completeness to my understanding of faithfulness:

1. Validity in so far as a target representation only includes statements which are correct and relevant about its source; and

2. Completeness in so far as a target representation includes all statements necessary to fully replicate the semantics of its source.

In order to assess faithfulness, each transformation to an OR-SQL representation using a SUDT will be evaluated against these criteria. Essentially an SUDT based representation is a faithful representation of a UML class if it is both valid and complete with respect to the definition of that class.

I relate comprehension to my definition of a viable transformation. A *viable transformation* is one which addresses the problem of an object-relational impedance mismatch, by reducing the time and effort spent developing a mapping for a programmer competent with Java and SQL. Such a transformation may for example ensure the consistent naming of a structure between a program and a database; or ensure a one-to-one mapping between two representations.

In order to assess the contribution of a SUDT to the resolution of object-relational impedance mismatch I must be able to quantify its impact on a Java program. A review of the literature highlights that there is no published method to quantify the effects of an object-relational impedance mismatch and therefore assess the viability of a representation. The production of measures to quantify the effects of an object-relational impedance mismatch and therefore assess viability forms a significant contribution of my research.

Although a greater understanding will be developed within the research project, my initial understanding is that a quantification of viability will include factors such as:

1. the number of database accesses required to both extract and save the details of an object;
2. the number of program statements required to map the details of an object between representations;

3. the number of shadow attributes (Ambler 2006) required to map the details of an object between representations. Surrogate database identifiers and type indicators are just two examples; and
4. The degree to which a programmer can comprehend and implement a mapping.

Point 4 requires further clarification. An SUDT represents one possible solution to an object-relational impedance mismatch. Whilst there is also no published work on comprehension with regard to an SUDT, I believe it is important to first understand and assess the contribution of an SUDT. My research will provide the understanding necessary to inform a future field study into how an SUDT based representation aids programmer comprehension.

A conceptual gap exists when there is no direct correspondence between two representations. A conceptual gap inhibits programmer comprehension by obscuring the semantics of a mapping between two representations. One (initial) indirect measure of comprehension is the degree of a mapping relationship between a Java and an SUDT based representation. A degree of one represents correspondence and does not inhibit comprehension. Such a degree may be possible for example, if the transformation of a class hierarchy results in a separate SUDT for each class. Current approaches (Hohenstein 1996) typically result in a degree greater than one[7] and therefore lack such correspondence.

In order to assess the contribution of an SUDT, each Java code fragment will be assessed against these and other criteria and compared with the Java code required to maintain an SQL representation which does not make use of an SUDT. Whilst validating my choice of metrics, this comparison will also provide evidence of the contribution of a SUDT to the resolution of an object-relational impedance mismatch.

[7] A Java object may be represented by a row(s) in two or more relational tables. A table may also store the data of objects from many different classes.

4.5 Research Method

I propose to test the hypothesis identified in the previous section using the method presented in Figure 6. This method provides for the analysis of a series of transformations using a set of criteria in order to test the validity of my hypothesis.

Using the method I will test the contribution of a SUDT to the viability of a transformation. I will do so by assessing the contribution of the result of each transformation to the problem of an impedance mismatch. In this case the dependent variable is viability. The independent variable is a faithful representation using a SUDT.

A key element of the method is the definition of a control group. The control will be used to assess the impact of a SUDT representation and provide internal validity ((Oates 2006), p131).

The control group will be a collection of faithful transformations which make no use of a SUDT. These transformations will be based on established techniques ((Hohenstein 1996), p406) and (Keller 1997)) for the implementation of a subtype using SQL. The control group provides a baseline for comparison which when combined with my metrics for viability, is essential for assessing the impact of an SUDT based representation. The use of established techniques provides method triangulation which enforces the validity of my research outcomes.

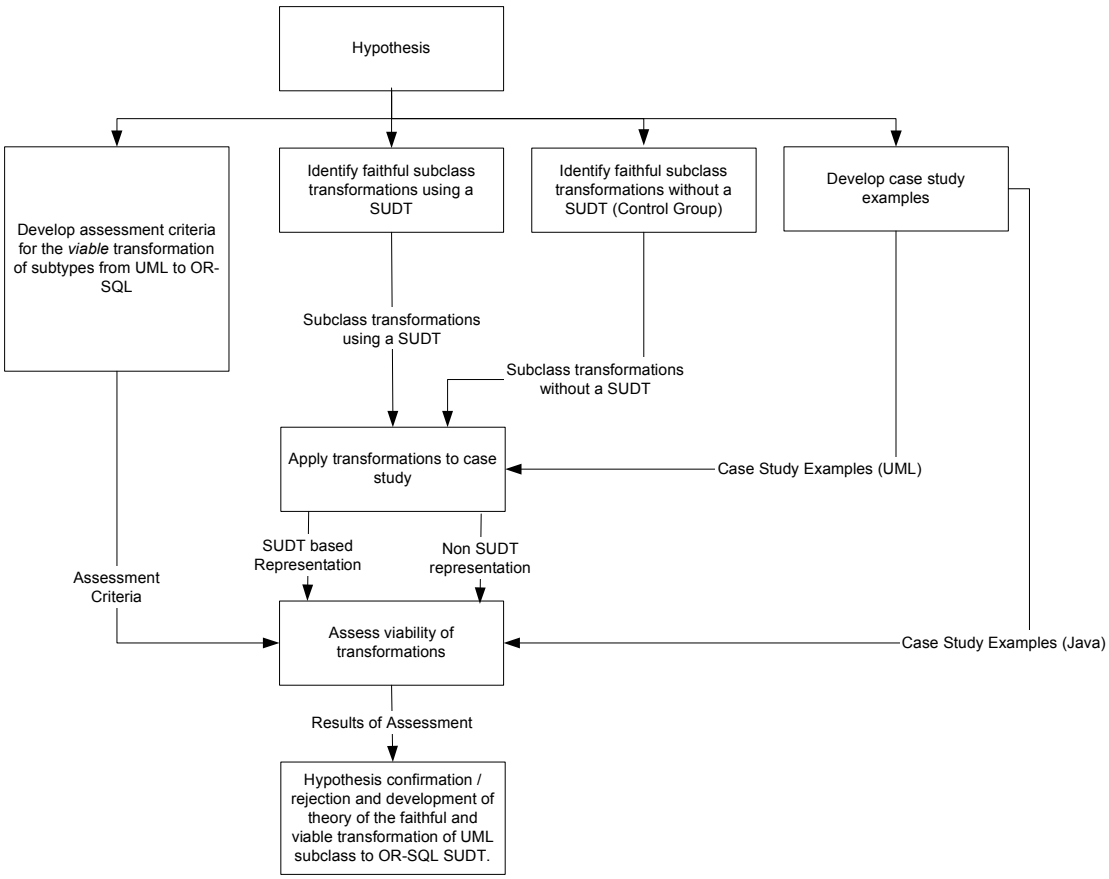


Figure 6 - Research Method

In order to avoid a completely artificial situation I will use a case study based on practical experience in a leading City of London financial institution (Appendix B). This will provide a basis for exploring possibilities and testing transformations. Where possible I also plan to network with fellow IT practitioners, the OMG and ORDB vendors to identify the ways in which both a subclass and a SUDT are used in real world projects.

One significant consequence of this approach is its lack of impartiality. The criteria for viability are developed in parallel with UML class model case study examples by the same researcher. There is a risk that: the case study examples will serve to justify a transformation; and a transformation will only work with a particular example. I aim to counter this criticism through: the influence of literature such as (Brachman 1983) which will provide a third party framework for an understanding of the potential semantics of a subclass; and an assessment of the strengths and weaknesses of each

transformation in the context of a Java program and a control group from the same UML class model.

The method provides scope for the definition of assessment criteria for the viability of a transformation beyond that presented above. The method also includes activities for the further refinement of the case study and the production of a set of faithful transformations. Although the activities shown in Figure 6 will be time-boxed, it is my expectation that the assessment will be complete when each subclass transformation has been applied to the case study and assessed using these criteria.

The success of my research project rests on providing understanding of how an SUDT based representation provides a solution to the problem of object-relational impedance mismatch. One possible outcome is that an SUDT provides no contribution. In this case my research will provide evidence for this and highlight both the weaknesses of an SUDT based approach and changes which will improve the situation. Should time permit the scope of my research may be widened to include other OR-SQL structures which may address some of the weaknesses.

5. The Research Plan

This section presents the plan for the PhD research project to address the question identified in section 4.3.

5.1 Activities, Milestones and Timescales

The initial timeframe for the project is three calendar years from probation (Figure 7). The main activities are taken from the research method (Figure 6). As a part-time student this allows about a year for contingency. This contingency is 25% of the overall time remaining (post probation) for part-time registration. It reflects the need to maintain a realistic perspective on key activities including refinement, consolidation of knowledge and the final write-up. Two important points are:

1. The literature review and writing up are continuous processes. New material must be consolidated with my research in order to ground the results of my work;
2. The activities in Phase 2 are iterative and time boxed. This reflects the need to both move forward and learn from the application of a transformation.

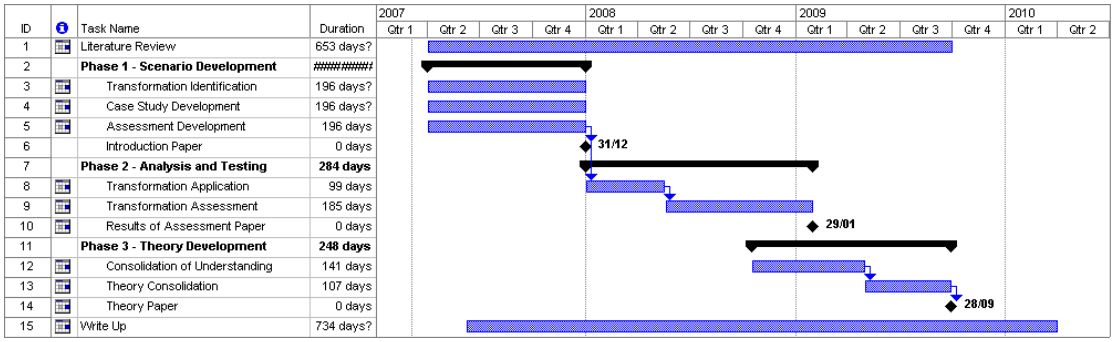


Figure 7 - The Research Plan

Key milestones and deliverables in the project are:

1. Completion of a literature review;
2. A set of evaluation scenarios (including criteria and cases) with a first paper highlighting my approach and assessment criteria;
3. A set of test results with a second paper describing the results of the assessment with lessons learned; and

4. A theory for the faithful and viable transformation of a UML subclass to a representation using a SUDT in the context of an object-relational impedance mismatch; a third paper which presents my theory; and a final PhD thesis.

5.2 Resources

It is anticipated that the following are critical resources for the success of the project:

1. Access to the business knowledge of my colleagues at my client;
2. A UML case tool – for diagramming and scenario development;
3. An ORDBMS (such as DB/2 which implements SUDT and typed tables) – for transformation evaluation;
4. The Java object-oriented programming language for transformation evaluation;
5. Access to the OU library and other internet based research resources;
6. EndNote for bibliography management; and
7. A word processor and spreadsheet for documentation.

6. Summary of Proposal

The success of a software system depends on the accurate representation of real-world concepts and their faithful transformation into software. The UML has become the language of choice for those designing and building object-based software systems.

OR-SQL represents an attempt to catch up with developments in object-oriented technologies since the introduction of the SQL language for describing a relational database. At the heart of this push is the structured user defined type (SUDT).

My hypothesis is that by producing a faithful transformation of a UML subclass, I can assess whether a SUDT provides a viable solution to the problem of an object-relational impedance mismatch. My research proposal tests this hypothesis whilst recognising the interplay of requirements from a number of stakeholders.

My research will identify faithful and viable transformations of a UML subclass to an OR-SQL representation using a SUDT. My goal is to provide a sound theoretical basis for the use of a SUDT to address an object-relational impedance mismatch.

[12,299 Words]

7. References and Bibliography

Ambler, S. (1995). "Mapping objects to relational databases." Software Development **3**(10): 63-69.

Ambler, S. (2006, 6th October 2006). "Mapping Objects to Relational Databases: O/R Mapping In Detail." Retrieved 12th April 2007, from http://www.agiledata.org/essays/mappingObjects.html.

An, Y., A. Borgida, et al. (2006). "Discovering the Semantics of Relational Tables Through Mappings." LNCS 4244 - Journal on Data Semantics VII: 1-32.

Atkinson, C. and T. Kuhne (2002). "Rearchitecting the UML infrastructure." ACM Transactions on Modeling and Computer Simulation **12**(4): 290-321.

Brachman, R. J. (1983). "What IS-A Is and Isn't: An Analysis of Taxonomic Links in Semantic Networks." Computer **16**(10): 30-36.

Cabibbo, L. and R. Porcelli (2003). M2ORM2: A Model for the Transparent Management of Relationally Persistent Objects. Database Programming Languages: 9th International Workshop, Potsdam, Germany, Springer Berlin / Heidelberg.

Codd, E. F. (1970). "A relational model of data for large shared data banks." Communications of the ACM **13**(6): 377-387.

Djuric, D., D. Gasevic, et al. (2006). "The Tao of Modeling Spaces." Journal of Object Technology **5**(8): 125-147.

Eichelberger, H. (2003). Nice class diagrams admit good design? Proceedings of the 2003 ACM symposium on Software visualization, San Diego, California, ACM Press.

Evans, A. S. (1998). Reasoning with UML class diagrams. Proceedings. 2nd IEEE Workshop on Industrial Strength Formal Specification Techniques, 1998, Boca Raton, FL

Fitzgerald, B. and D. Howcroft (1998). Competing dichotomies in IS research and possible strategies for resolution. Proceedings of the international conference on Information systems. Helsinki, Finland, Association for Information Systems.

Grant, E. S., R. Chennamaneni, et al. (2006). Towards analyzing UML class diagram models to object-relational database systems transformations. 24th IASTED international conference on Database and applications Innsbruck, Austria ACTA Press, Anaheim, CA, USA.

Grehan, R. (1998). Object marries relational. Byte Magazine. **23:** 101(2).

Hainaut, J.-L. (2006). "The Transformational Approach to Database Engineering." Lecture Notes in Computer Science **4143**: 95-143.

He, X. (2000). Formalizing UML class diagrams-a hierarchical predicate transition net approach. The 24th Annual International Computer Software and Applications Conference, 2000. COMPSAC 2000., Taipei

Hohenstein, U. (1996). Bridging the Gap between C++ and Relational Databases. European Conference on Object-Oriented Programming, Springer-Verlag, Berlin.

ISO (2003). Part 2: Foundation (SQL/Foundation), ISO/IEC 9075.

Jacobson, I., G. Booch, et al. (2005). The Unified Software Development Process. Boston, MA, Addison-Wesley.

Juric, R. (1998). "The UML rules." SIGSOFT Softw. Eng. Notes **23**(1): 92-97.

Keim, D. A., H.-P. Kriegel, et al. (1993). Object-Oriented Querying of Existing Relational Databases. Database and Expert Systems Applications: 4th International Conference, 1993 Proceedings DEXA'93 Prague, Czech Republic, Springer Berlin / Heidelberg

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

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

Marcos, E. and J. M. Cavero (2002). Hierarchies in Object Oriented Conceptual Modeling. Advances in Object-Oriented Information Systems : OOIS 2002 Workshops, Montpellier, France, Springer Berlin / Heidelberg

Marcos, E., B. Vela, et al. (2001). Extending UML for Object-Relational Database Design UML 2001 - The Unified Modeling Language.Modeling Languages,

Concepts, and Tools: 4th International Conference, Toronto, Canada, Springer-Verlag GmbH

Marcos, E., B. Vela, et al. (2003). "A Methodological Approach for Object-Relational Database Design using UML." Software and Systems Modeling **2**(1): 59-72.

Meyer, B. (1996). "The many faces of inheritance: a taxonomy of taxonomy." Computer **29**(5): 105-108.

Mok, W. Y., Y.-K. Ng, et al. (1996). "A normal form for precisely characterizing redundancy in nested relations." ACM Transactions on Database Systems **21**(1): 77-106.

Mok, W. Y. and D. P. Paper (2001). On Transformations from UML Models to Object-Relational Databases. 34th International Conference on System Sciences, Hawaii IEEE.

Moody, D. L., G. Sindre, et al. (2003). Evaluating the quality of information models: empirical testing of a conceptual model quality framework.

Morris, S. and G. Spanoudakis (2001). UML: an evaluation of the visual syntax of the language. Proceedings of the 34th Annual Hawaii International Conference on System Sciences, 2001.

Naiburg, E. J. and R. A. Maksimchuk (2001). UML for Database Design, Addison Wesley.

Neward, T. (2006). "The Vietnam of Computer Science." Retrieved February 2007, from http://blogs.tedneward.com/2006/06/26/The+Vietnam+Of+Computer+Science.aspx.

Oates, B. J. (2006). Researching Information Systems and Computing. London, SAGE Publications.

OMG. (2004). "Unified Modeling Language (UML) Specification: Infrastructure version 2.0." Retrieved January, 2006, from http://www.omg.org/technology/documents/modeling_spec_catalog.htm.

Phillips, E. M. and D. S. Pugh (2005). How to get a PhD - A handbook for students and their supervisors, Open University Press.

Purchase, H. C., L. Colpoys, et al. (2001). UML class diagram syntax: an empirical study of comprehension. Australian symposium on Information visualisation - Volume 9, Sydney, Australia, Australian Computer Society, Inc.

Schmid, H. A. and J. R. Swenson (1975). On the semantics of the relational data model. Proceedings of the 1975 ACM SIGMOD international conference on Management of data, San Jose, California, ACM Press.

Shroff, M. and R. B. France (1997). Towards a formalization of UML class structures in Z. Proceedings., The Twenty-First Annual International Computer Software and Applications Conference, 1997. COMPSAC '97. , Washington, DC

Siau, K. (2004). "Relationship Construct in Modeling Information Systems: Identifying Relationships Based on Relation Element Theory." Journal of Database Management **15**(3).

Smith, J. M. and D. C. P. Smith (1977). "Database Abstractions: Aggregation and Generalization." ACM Transactions on Database Systems **2**(2): 105-133.

Snyder, M. and T. O'Connor. (2005). "Object-Relational Mapping In Java with SimpleORM." from http://www.ddj.com/184406344.

Soutou, C. (2001). "Modeling relationships in object-relational databases." Data and Knowledge Engineering **36**(1): 79-107.

Soutou, C. (2005). Referential Integrity in Commercial Object-Relational Databases, University of Toulouse**:** 17.

Stonebraker, M., L. A. Rowe, et al. (1990). "Third-generation database system manifesto." ACM SIGMOD Record **19**(3): 31-44.

Stonebroker, M. (1998). Object-relational DBMSs: Tracking the Next Great Wave, Morgan Kaufmann.

Sutherland, J., M. Pope, et al. (1993). The Hybrid Object-Relational Architecture (HORA): an integration of object-oriented and relational technology. Proceedings of the 1993 ACM/SIGAPP symposium on Applied computing: states of the art and practice. Indianapolis, Indiana, United States, ACM Press.

Zhang, N., N. Ritter, et al. (2001). Enriched relationship processing in object-relational database management systems. The Proceedings of the Third International Symposium on Cooperative Database Systems for Advanced Applications, 2001. CODAS 2001., Beijing

Zhang W., R. N. (2001). The Real Benefits of Object-Relational DB-Technology for Object-Oriented Software Development. British National Conference on Databases, Springer-Verlag, Germany.

Zhang, W. P. and N. Ritter (2001). The Real Benefits of Object-Relational DB-Technology for Object-Oriented Software Development. British National Conference on Databases, Springer-Verlag, Germany.

A. Informal Survey of Practitioners

An informal, email survey of nine practitioners within two City of London financial institutions indicated some of the reasons why ORDB technology has not been adopted.

The nine practitioners who responded were application developers, project leaders and DBAs.

There responses are summarised here:

1. Lack of knowledge or expertise on the part of practitioners in object-relational features;
2. The use of object-relational features is seen as an overhead on existing projects;
3. The design of the system did not use an object-oriented approach therefore why use object-relational features;
4. Too costly to redesign the system to make use of object-relational features;
5. The database is considered a bottleneck and so it is used only for data storage;
6. The languages used (Pro*C is cited as one example) do not support the features (this may be a lack of knowledge on the part of the respondent);
7. Object-relational features are only useful for a certain type of application (in this case derivatives trading is suggested). Equities are far more stable and less likely to change and therefore would not benefit from this new technology; and
8. A lot of bank systems are based on packages and the vendors have not made use of the object-relational features.

B.Initial Case Study

The Financial Trading Institution (FTI) is a fictitious Bank based in the City of London. Its principal activities are the trading of financial instruments (referred to simply as instruments). Trading involves the purchase and sale of financial instruments.

The trading activities of FTI are focussed on two main financial instruments: equities and bonds. Specifically the Bank buys and sells equities and bonds on its own behalf (known as proprietary trading) and on the behalf of clients. Ultimately the aim is to make a profit through these dealing activities.

Each financial instrument is assigned a unique ID called the ISIN (International Securities Identifying Number). The structure of an ISIN is defined in ISO 6166.

Equities are also known as shares and the terms are synonymous. The purchase of one share entitles the owner of that share to literally share in the ownership of the company a fraction of the decision-making power, and potentially a fraction of the profits (see http://en.wikipedia.org/wiki/Shares for further details).

Equities may be listed on financial markets within any of the many financial exchanges world-wide. A market is a sub-division of an exchange. These sub-divisions are created by the exchange for many reasons not limited to the value and associated investment risks of the companies listed therein.

A company will apply for a listing of their equity on a market in a specific country because it wishes to raise capital in that country. The process of listing makes company shares available to buy or sell on a particular financial market. There are a number of financial regulations which govern the listing of company equity on an exchange but if an application is successful a company's equity becomes tradable on a specific market within that exchange.

Bonds are also known as debt instruments. Bonds are a means of financing a company's debt over a period of time (called the term). One way to think about bonds is as a loan from the purchaser of the bond to the issuing company or government.

Rather than owning a share of the company, the purchaser of a bond is buying part of the debt of that company. In so doing the purchaser becomes entitled to interest payments from the company on the bond; and the return of their initial investment on maturity if they still own the bond.

Typically government bonds are issued by governments to finance capital investment. The purchaser of the bond is effectively lending money to the particular government in question. In return the purchaser is guaranteed the return of their initial investment at maturity plus interest payments whilst they hold the bond.

Interest payments on a bond may be fixed or variable (also called floating) and are the key source of income from holding the bond. At the end of the term the bond holder also receives a repayment of their investment.

The trading process at FTI starts either with an order from a client (electronically or by telephone) to buy or sell some financial instrument or from an order raised by one of the FTIs own sales people (a proprietary order).

An order is a *registration of intent* to buy or sell some financial instrument.

An FTI sales person may order shares to develop their own position. A position may be long (i.e. a positive number of shares) or short (i.e. a negative number of shares).

A short position can be profitable in a falling market where the price to buy the shares back later will be less than the price at which they were sold. Holding a position in a stock provides FTI with exposure risk. They may take this risk if the research

department suggests that the stock is currently undervalued (by holding a long position) or overvalued (by holding a short position).

Sales people and traders are just two of the many different kinds of staff within the bank. They are distinguished from other kinds of staff in that they earn revenue for the bank through their work activities.

Sales people and traders are offered incentives through a profit sharing scheme. The scheme pays them anything from 2% to 10% of profits as an annual bonus subject to achieving targets.

A trader picks up an order to buy or sell an instrument and deals on the market (i.e. executes trades) with other traders in other institutions.

At FTI, some traders trade in the shares of companies with a small capitalisation[8] or large capitalisation whist others trade in both.

A trade differs from an order in that it represents an *actual exchange* of financial instruments[9].

Ultimately the idea is to buy as cheaply as possible and to sell at the highest price possible in order to make a profit, although traders also charge a commission for trades which fulfil a client order.

A trader may undertake a number of trades to complete an order. One reason they do this is to minimize the effect of their trading activity on the market price of the instrument they are trying to buy or sell. If they trade to buy a significant amount of an instrument for example, other traders will identify the significance of this trade and

[8] Capitalisation refers to the value (price * quantity) of shares in circulation.

[9] We use the term "exchange of financial instruments" because although we do not show it on Figure 8, cash is just another form of financial instrument. An equity trade in its simplest form is therefore an exchange of a given quantity of shares for a given quantity of cash.

increase the price at which they are willing to sell the instrument. It is hoped that by trading in a larger number of smaller amounts will not adversely impact the market price.

Proprietary dealing activities are supported by a team of researchers who look for trading opportunities through examination of micro and macro economic factors. The bank may also provide this research information to selected clients for a fee.

All FTI trading activity is supported by a suite of IT systems which provide a range of services including order entry, trade execution, market interfaces and settlement. Technologists provide support for these systems and ensure continuity of the 24/7 global business.

Figure 8 presents a UML class model for the business at FTI (the universe of discourse). It is only one possible model of the business and it is not the aim of this paper to undertake a critique of the model itself. This model will be developed further and used as the basis of examples throughout the research project.

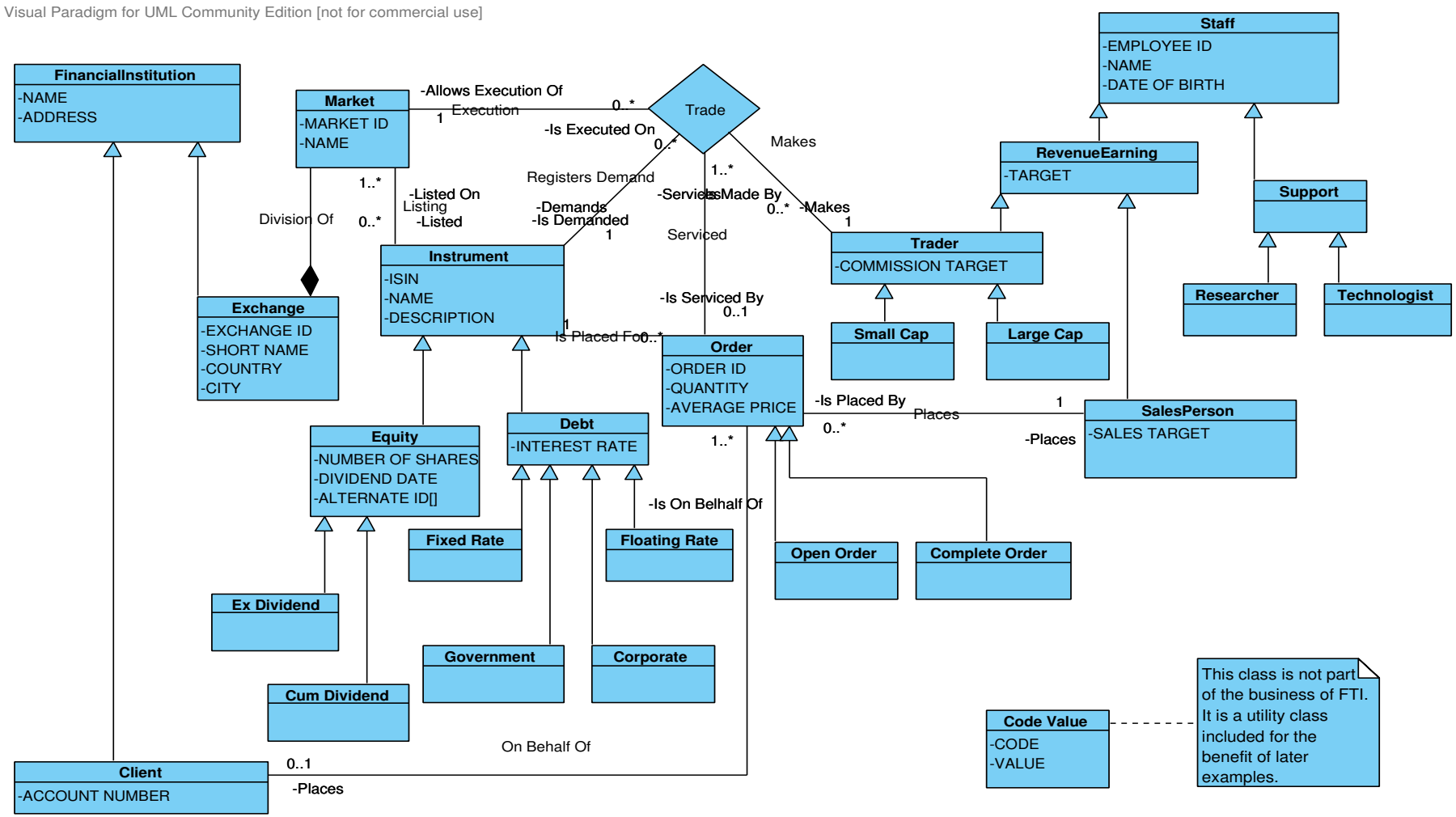


Figure 8 - The FTI UML Class Model

C. An Object-Relational Impedance Mismatch

In this appendix I present a brief worked-example of an object-relational impedance mismatch. The example is based on the FTI case study (Appendix B) and a schema representation which does not make use of object-relational features of SQL.

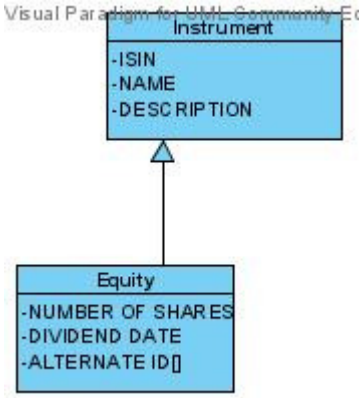


Figure 9 - The Instrument and Equity Classifications

This example is based on two classifications (Figure 9). All instruments are uniquely identified by their ISIN code. For example, the equity instrument "Vodafone" is identified by the ISIN **GB0007192106**. An equity is a kind of financial instrument which pays a dividend.

```
create table INSTRUMENT
ISIN    char(12) not NULL,
NAME    char(20) not NULL,
DESCRIPTION char(255) NULL,
PRIMARY KEY (ISIN))
```

```
create table EQUITY(
ISIN                    char(12) not NULL,
NUMBER_OF_SHARES        integer NOT NULL,
DIVIDEND_DATE           char(8) NULL,
constraint C1           FOREIGN_KEY(ISIN)
references              INSTRUMENT)
```

Figure 10 - The Tables INSTRUMENT and EQUITY

Using a table-per-class transformation to SQL ((Keller 1997), p13), the classifications Instrument and Equity are transformed into the tables INSTRUMENT and EQUITY. The attributes of a class become columns in the corresponding table. I have added a foreign key constraint on column ISIN of table EQUITY. This represents the semantics of an equity as a kind of instrument.

From the UML model, the class Equity will be transformed into the Java class: Equity (Figure 11).

```
/* Assume all necessary JDBC classes are imported */
/* No error handling included in this example. */
public class Equity extends Instrument
```

```
{
  private int m_NumberOfShares;
  private string m_DividendDate;

  public void Equity(…) { /* Constructor */ }
  public Save(Equity anEquity) {
         /* Connect to DB */
         connection = DriverManager.getConnection( url, user, password);
         /* Allocate a statement handle */
         statement = connection.createStatement();

         /* Map Instrument fields to SQL insert statement */
         string sqlInstrument = "insert into INSTRUMENT("
                              +"ISIN, NAME, DESCRIPTION)"
                              +"values("' + anEquity.ISIN + '",'"
                                        + anEquity.NAME + '",'"
                                        + anEquity.DESCRIPTION + "')";
         /* Access DB to save Instrument data into table INSTRUMENT */;
         statement.executeUpdate(sqlInstrument);


         /* Map Equity fields to SQL insert statement */
         string sqlEquity = "insert into EQUITY("
                              +"ISIN, NUMBER_OF_SHARES, DIVIDEND_DATE)"
                              +"values("' + anEquity.ISIN + '","
                                  + anEquity.NUMBER_OF_SHARES + ",'"
                                  + anEquity.DIVIDEND_DATE + "')";
         /* Access DB to save Instrument data into table EQUITY */
         statement.executeUpdate(sqlEquity);

         /* Fee handle */
         statement.close();
         /* Disconnect from DB */
         connection.close();
  }
}
```

Figure 11 - The Java Class Equity

If there were no impedance mismatch between the Java program and the SQL schema we would expect a single database access to save all the data values necessary to fully represent an Equity object.

The impact of an impedance mismatch is apparent from the extra code required to save the details in the table INSTRUMENT as well as EQUITY.

As a result we could quantify the impact of this impedance mismatch problem using the number of additional Java statements required to implement a solution. In this case

there are two extra statements (in bold) required over and above the use of a single table.

Other issues include:

1. Although the SQL representation uses the ISIN as a primary key, the Java object is identified by a system generated object identifier. This identifier is not present in the database and has no relevance outside the context of a Java program;
2. References between classes in a Java program will be based on the internal object identifier not the ISIN. In the SQL representation the associations are based on ISIN;
3. The names of the member variables and SQL columns may differ depending on the naming conventions in place. A programmer must know that the member variable m_ISIN maps to the SQL column ISIN;
4. This is just a simple example using two classes. The problem is compounded when one considers links to other objects in other classes. An object may include a reference another object(s). We have deliberately excluded the attribute ALTERNATE_ID of Equity from this example in order to keep it simple. How does one define the scope of an SQL query to provide all the necessary details to reconstruct an object? Too narrow a scope will increase the number of database accesses required as referenced objects are loaded on demand (if this is indeed possible).
5. In order to maintain the integrity of a Java object, all its details must be loaded from the database. It is not possible to create just part of an object. One compromise is to allow Java member variables to be null. The semantics of this solution are not clear: are they null because there is no value or because no value has yet been loaded from the database?

D.Synopsis of My Previous Work

In this appendix I present a summary of a set of papers I have produced over the past 18 months. These papers provide a consolidation of my research into the field of ORIC.

The Role of Semantics

Examines what we mean by the term "semantics"; why semantics are important; and to whom are they important in the delivery of an IT system.

The Semantics of Subtypes

Presents subtypes within the context of conceptual modelling. The paper examines the role of subtypes in a process of abstraction and the use of inheritance as one possible implementation. The effects of a subtype on the semantics of a class association are discussed. Finally the semantics of a subtype as implemented in the OR-SQL standard are analysed by examining the strengths and weaknesses of the approach using a SUDT.

The Semantics of UML Subclasses

This paper examines the semantics of a UML subclass. The structure and content of a UML class model are examined in order to gain an understanding of how a class model may be used. The paper finishes with a comparison of the UML subclass relationship with eleven different kinds of inheritance identified by (Meyer 1996). The comparison highlighted the need for precision when using a UML subclass.

Reflections on Subclasses - Conceptual Modelling

This paper draws on the work on taxonomy by Marcos (Marcos and Cavero 2002). In their paper they present a taxonomy of taxonomies. In particular they discuss issues surrounding the difference between taxonomy and inheritance. Importantly this paper establishes the importance of both a definition of terms and a naming convention in academic writing. It also incorporates a case study based on a University.

The Semantics of Subtypes – Revisited

This paper is based on the work of (Brachman 1983) and (Marcos and Cavero 2002). They describes a number of different kinds of IS-A (subclass) relation. Although this paper was not completed, my goal was to describe the semantics of different kinds of IS-A relationship in such a way as to be then able provide possible OR-SQL implementations. This work for this paper may be continued within the PhD research project because it forms part of the

definition of a universe of possible subclass semantics for the proposed case study transformations.

Review of Research Methods

The ultimate goal of this paper is to identify a suitable research method for my PhD.

The paper starts by examining the role of a research method as a way of going about research. The paper adopts a pluralist approach (Fitzgerald and Howcroft 1998) to the selection of a method by considering the role of the researcher, the researched, the context and a plan of activities.

An analysis of five published papers in the field of Computing results in a taxonomy of research methods. This is then used as the basis for the selection of an appropriate research method for one *possible* research question.