Conceptual Schema Optimisation – Database Optimisation before sliding down the Waterfall

H.A. Proper[1] and T.A. Halpin[2]
Department of Computer Science
University of Queensland
Brisbane
Australia 4072
E.Proper@acm.org

Version of June 23, 2004 at 10:31

PUBLISHED AS:

H.A. Proper and T.A. Halpin. Conceptual Schema Optimisation – Database Optimisation before sliding down the Waterfall. Technical Report 341, Department of Computer Science, University of Queensland, Brisbane, Australia, July 1995.

Abstract

In this article we discuss an approach to database optimisation in which a conceptual schema is optimised by applying a sequence of transformations. By performing these optimisations on the conceptual schema, a large part of the database optimisation can be done before actually sliding down the software development waterfall.

When optimising schemas, one would like to preserve some level of equivalence between the schemas before and after a transformation. We distinguish between two classes of equivalence, one based on the mathematical semantics of the conceptual schemas, and one on conceptual preference by humans.

As a medium for the schema transformations we use the universe of all (correct) conceptual schemas. A schema transformation process can then be seen as a journey (a schema- time worm) within this universe. The underlying theory is conveyed intuitively with sample transformations, and formalised within the framework of Object-Role Modelling. A metalanguage is introduced for the specification of transformations, and more importantly their semantics. While the discussion focusses on the data perspective, the approach has a high level of generality and is extensible to process and behaviour perspectives.

1 Introduction

Modern approaches to information system development usually start out by modelling a universe of discourse in terms of a conceptual schema, using the notation of a conceptual modelling method such as Enhanced Entity Relationship (EER) modelling or Object-Role Modelling (ORM). The design of this schema is ideally guided by some kind of conceptual schema design procedure. For many years, ORM has featured well developed design procedures ([Win90], [Hal95]), and some variants of EER now include design techniques (e.g. [EN94], [BCN92]). Most modern Object-Oriented (OO) approaches also include a design procedure ([RBP$^+$91], [CY90], [Kri94]).

[1] Part of this work has been supported by an Australian Research Council grant, entitled: “An expert system for improving complex database design”

[2] Currently on leave at Asymetrix Corporation, Bellevue WA, USA

After a conceptual schema of a given universe of discourse is finalised, it can be mapped to a database schema for the actual implementation. This database schema might include a set of tables for a relational database management system, or a set of classes for an object-oriented database management system. During the mapping from a conceptual schema to a database schema, optimisation issues usually come into play, since one would like the resulting database schema to have a good performance in the given context of the application. In practice this usually means that either during the mapping or afterwards, optimisations of the database schema are made.

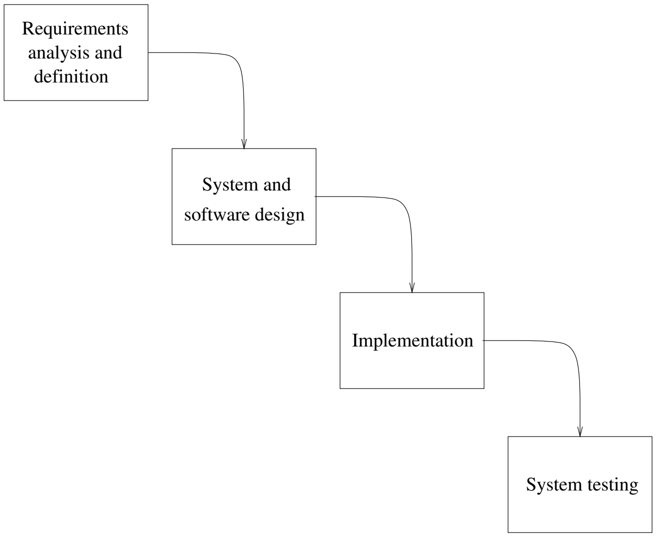

Requirements analysis and definition
System and software design
Implementation
System testing

Figure 1: The simple waterfall view of the software development life-cycle

In this article we are concerned with an approach that allows us to do schema optimisations on the conceptual schema, i.e. before sliding down the waterfall (see figure 1). We realise that the waterfall model only provides a simplistic view on an information system development life cycle, and that in practice the waterfall contains a number of sub-cycles. However, it does provide a good framework to discuss our ideas, although the ideas are independent of the actual waterfall model.

Although not all database schema optimisations can be done at this higher level, many optimisations can indeed be done at this level, offering a number of advantages. For example, conceptual schemas provide a clearer, human-oriented picture of the universe of discourse, so user participation during the optimisation process is more viable, where different schema alternatives are discussed. Moreover, at the conceptual level, schema transformations may be studied in a way independent of the implementation platform (relational, object-oriented, hierarchical, . . .).

To provide a better context for the contributions made in this article, we now describe the information system design process itself as a schema transformation process. Viewing the whole process of transforming a draft conceptual schema to the final database schema as a sequence of schema transformations, is an idea that is closely related to the idea of program derivation by a sequence of transformations ([BBP$^+$79], [PS83], [BMPP89]). In our view, the following kinds of schema transformations ([Hal92b]) may be distinguished during a database modelling process:

1. Conceptual schema draft
2. Conceptual schema refinement
3. Conceptual schema optimisation
4. Conceptual to database schema mapping

5. Database schema optimisation

We now briefly discuss each of these phases. When a conceptual schema is oginally drafted, it typically goes through a series by transformations to improve the current design, in accordance with the design procedure. Most of these transformations do not maintain equivalence. Extra information may be added, and unwanted details may be removed.

Before or after a universe of discourse is completely captured in a conceptual schema, it might be discovered that certain parts of this conceptual schema have alternative representations. One of these alternatives will usually be considered preferable in terms of the way that the user(s) wish to think about the application. So after the initial conceptual schema has been developed, a series of transformations might be performed that lead to a schema that is a preferred view of the universe of discourse. This class of transformations normally has to preserve the (mathematical) equivalence of the schemas as they deal with alternatives describing one and the same universe of discourse. As an example, consider the ORM schemas in figure 2. Here object types are shown as named ellipses with their identification schemes in parentheses; roles are shown as boxes attached to the object types that play them; predicates are depicted as named sequences of roles; constraints on values are listed in braces; and arrow-tipped bars denote uniqueness constraints on roles. Here the uniqueness constraints are the weakest possible (e.g. a lecturer may teach many students, and a student may be taught by many lecturers). The two ORM/NIAM schemas depicted are mathematically equivalent (for a proof of this, refer to [Hal89]). These two alternatives however might not be equally preferable ways for the user(s) to think about the application. The schemas in the example are modelled using the Object-Role Modelling (ORM) technique ([LN88], [HW93], [Win90], [Hal95]). Similar examples for EER or OO models could be given.

Once the user has selected the preferred conceptual schema, this should be used in any later conceptual queries on the implemented information system, assuming the availablity of an appropriate conceptual query language (see e.g. [Mee82], [HPW93], [PW95b]) Before mapping this preferred conceptual schema to the target database schema we may wish to perform some further optimisations to this conceptual schema, under the covers. As an example of such a transformation, again, consider figure 2. Depending on the *data profile* and *access profile*, either schema fragment may be more efficient to implement than the other. More examples of such conceptual schema optimisations can for instance be found in [Hal90], [Hal91], [Hal92a], [BCN92].

To prevent confusion in the remainder of this article, we will use the term *data schema* when we refer to an ORM schema in general, be it a preferred or only a draft conceptual representation of a universe of discourse. The term *conceptual schema* will typically be used for the preferred conceptual view. The term *database schema* is reserved for the target schema in the chosen database management system, which could be a relational, object-oriented, or hierarchical system.

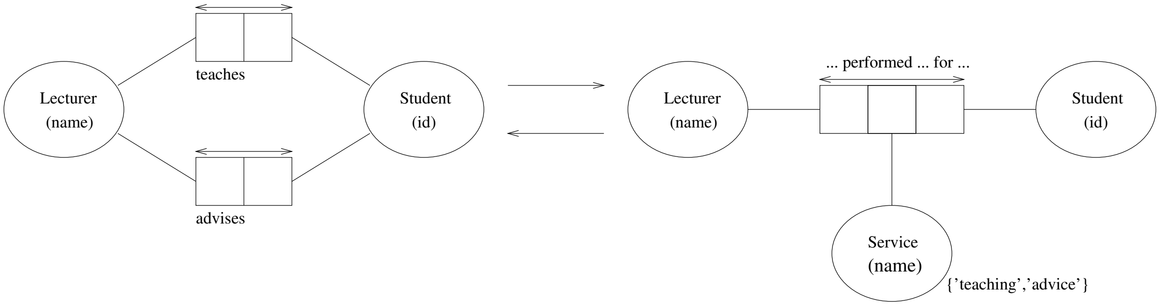


Figure 2: Example equivalence preserving schema transformation

When a data schema is mapped to a database schema, in general, a mapping algorithm is used that tries to find a (nearly) optimal representation. A wide range of algorithms for this purpose exists, for instance: [Ber86], [SEC87], [BW92], [Ris93], [MHR93], [Hal95], [Bom94], and [Rit94]. For the reverse process, reverse engineering, also a wide range of strategies and algorithms exists (e.g. [Kal91], [FG92], [SS93], [CBS94]). Note that reverse engineering may also be regarded as a sequence of schema transformations (basically the reverse of those used for conceptual schema to internal schema mapping).

Once a data schema has been represented as a database schema, this schema may sometimes be optimised further using transformations of the database schemas and adding indexes. Some of these transformations are discussed in e.g. [De 93], [Kob86a], and [BCN92].

The discussed five classes of transformations operate either on a data schema, or an internal schema of a given implementation platform. The interplay of these transformations is illustrated in figure 3. The modelling process up until the start of the database schema mapping can be regarded as a journey through the universe of data (ORM) schemas. This article focusses on schema transformations involved in classes 2 and 3. Furthermore, we limit ourselves to ORM models. This latter limitation, however, is not a strong one as the ORM modelling technique is general enough to cater for ER based data schemas as well ([BBMP95], [HP95]). In [CH94], [Cam94] and [CP96] it is shown how ER can be regarded as an abstraction from ORM schemas.

Research is also underway (together with the authors of [BW92]) to find an apt schema language to describe both data schemas as well as database schemas. This would allow us to describe the entire modelling process of an information system's data(base) schema within one modelling language. A likely candidate for this aim is the tree representation of ORM schemas as introduced in [BW92]. This technique is not a replacement of ORM, but rather an extension which allows the representation of candidate internal representations besides the normal data schema. These candidate internal representations have a one to one correspondence to relational models, NF^2 models, network, hierarchical and O^2 models.

The structure of the remainder of this article is as follows. Although we also consider schema transformations which do not maintain equivalence, it is essential to define exactly what we regard as schema equivalence. Therefore, in section 2 this notion is discussed in more detail. A more elaborate discussion of example transformations and their applications is given in section 3. As stated before, the design process of a data schema can be seen as making a journey through the universe of data schemas. In section 4 we therefore define the universe of ORM data schemas, while the notion of a schema version representing a state of the data schema during the design process is discussed in section 5. In this article we focus on the syntax of ORM models in the context of this universe. The semantics has been discussed in detail elsewhere. A language to define the schema transformation is introduced in section 6. Before concluding, we define in section 7 three ways in which to apply the schema transformations to an existing data schema.

2 Equivalence of Schemas

In the context of schema transformations two notions of schema equivalence are important. The first notion of equivalence is mathematical equivalence. Two schemas are mathematically equivalent if they define isomorphic state spaces. The second notion of equivalence tries to capture the conceptual quality of a conceptual schema. When there are two mathematically equivalent alternative schemas for the same universe of discourse, one alternative may still be a better representation of the domain than the other. For example, consider the two schemas shown in figure 2. For a given universe of discourse, one of these two may be more 'natural'. We first discuss the notion of mathematical equivalence.

2.1 Mathematical equivalence

We start by discussing three existing approaches to defining schema equivalence and their inter-relationships. In the remainder of this article, whenever we refer to *equivalence* without using the prefix *mathematical* or *conceptual*, we implicitly refer to *mathematical equivalence*.

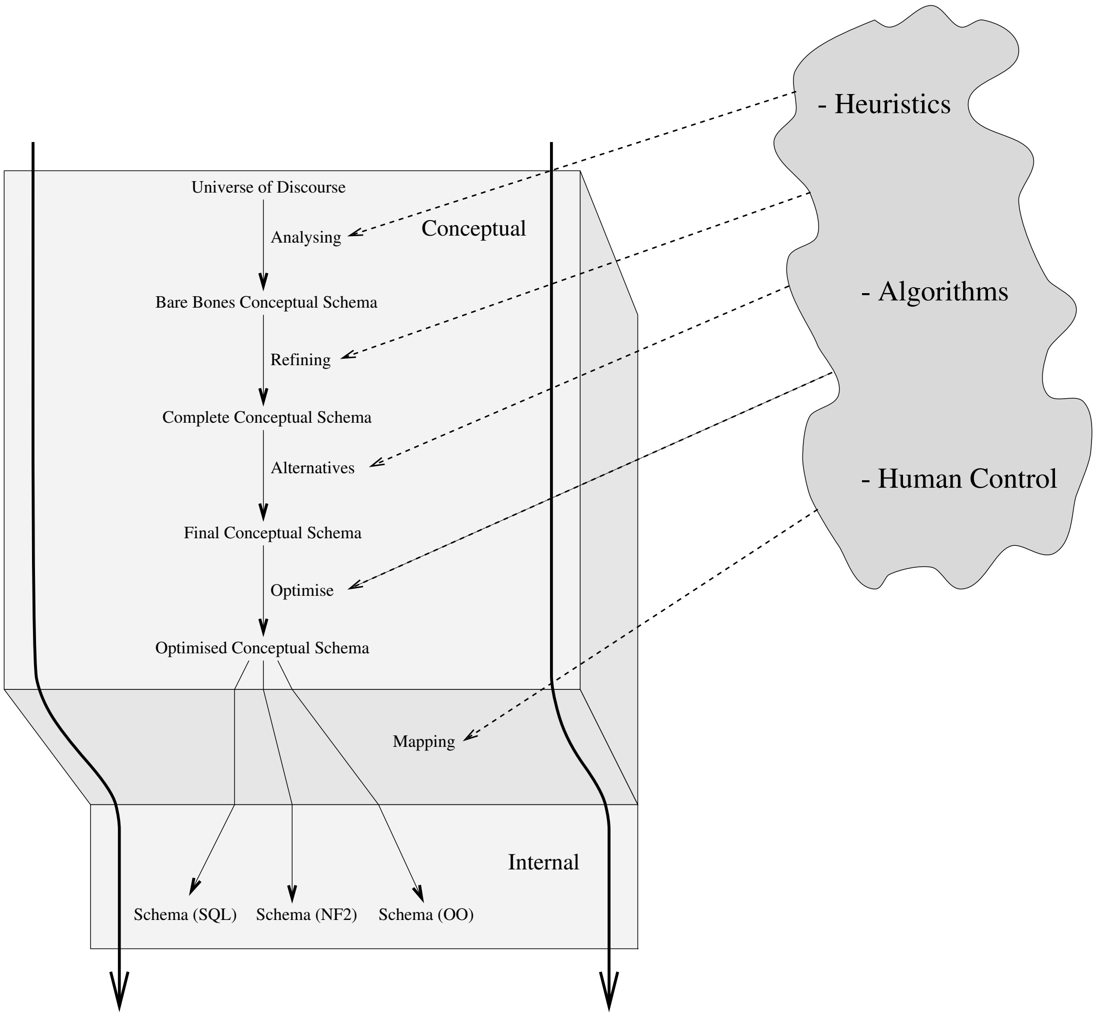


Figure 3: Single schema language

2.1.1 Set based equivalence

A data schema can be seen as an intensional specification of a set of valid populations. This is what we call the semantics of the schema. Even in databases which maintain the history ([RP92]) of the population this holds. In such cases, the schema semantics of the data schema is the set of valid populations at each point in time. If the schema does not evolve in the course of time, then the history of the database takes place within this fixed set of valid populations. How this view needs to be adapted in the case of evolving schemas is discussed in [PW95a] and [PW94].

Usually a population of a data schema $\mathcal{SCH}$ is modelled as a function:

$$p : \mathcal{TP} \to \wp(\Omega)$$

where $\mathcal{TP}$ is the set of (populatable) types defined in $\mathcal{SCH}$ and Ω is some domain of instances. The state space of a data schema $\mathcal{SCH}$ can then be defined as:

$$\mathcal{S}(\mathcal{SCH}) \triangleq \left\{ p : \mathcal{TP} \to \Omega \;\middle|\; \mathsf{IsPop}(\mathcal{SCH}, p) \right\}$$

where IsPop is a predicate that determines whether p is a proper population of $\mathcal{SCH}$. Two data schemas are now equivalent iff there exists a bijection between their state spaces ([HPW92a]), which is equivalent to saying that the state spaces are equally sized (could be infinite):

$$\mathcal{SCH}_1 \equiv \mathcal{SCH}_2 \triangleq \exists_h \left[h \text{ is a bijection } h : \mathcal{S}(\mathcal{SCH}_1) \rightarrow \mathcal{S}(\mathcal{SCH}_2)\right]$$

A direct result of the above two definitions is:

Corollary 2.1 If h is a bijection $h : \mathcal{S}(\mathcal{SCH}_1) \rightarrow \mathcal{S}(\mathcal{SCH}_2)$, then:

$$\mathsf{IsPop}(\mathcal{SCH}_1, p) \iff \mathsf{IsPop}(\mathcal{SCH}_2, h(p))$$

Using this definition, it indeed becomes provable that every ORM schema with a finite (n) set of populations is equivalent to a schema of the form depicted in figure 4 (see also [HPW92a]). Note that the $\#{=}1$ in this figure indicates that the population of value type NatNo only contains one instance, and $\{1..n\}$ limits the instances of NatNo to the interval 1 to n. The proof that each ORM schema is equivalent to this schema is based on the fact that the size of the populations is finite, and therefore there exists a bijection between the state space of the ORM schema and a subset of the natural numbers. The schema depicted in figure 4 simply corresponds to a data schema where each population corresponds to *one* natural number. We realise that this is an unlikely outcome of any schema optimisation process as it moves the optimisation difficulty from storing information to decoding the information. However one may argue that this schema does have a correspondence to reality, as all populations are stored on a hard disk as a sequence of bits (grouped in bytes) which can be seen as one large natural number.

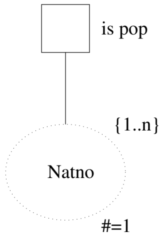


Figure 4: Most general schema

2.1.2 Logic based equivalence

In [De 93] and [Kob86b] a logic based notion of schema equivalence is introduced. In this approach a schema is interpreted as a logic signature with an associated set of basic axioms (the constraints). The notion of equivalence is then defined as:

$$\mathcal{SCH}_1 \equiv \mathcal{SCH}_2 \triangleq \exists_h \left[h \text{ is a bijection } h : \mathbb{I}(\mathcal{SCH}_1) \rightarrow \mathbb{I}(\mathcal{SCH}_2)\right]$$

where $\mathbb{I}(\mathcal{SCH})$ is the set of valid interpretations of data schema $\mathcal{SCH}$. This exactly corresponds to the above notion of equivalence, since $\mathbb{I}(\mathcal{SCH})$ corresponds directly to the notion of state space.

2.1.3 Contextual equivalence

The notion of equivalence as defined in [Hal89] is based on a more pragmatic approach. The above discussed notions of equivalence are not concerned with finding a proof of the equivalence, whereas the approach described in [Hal89] is. Every schema $\mathcal{SCH}$ can be seen as a set of logic formulae $\mathbb{L}(\mathcal{SCH})$ describing the structure and the constraints of the conceptual schema[1]. Given two of such sets ($\mathbb{L}(\mathcal{SCH}_1)$, $\mathbb{L}(\mathcal{SCH}_2)$)

[1] One might argue that not all schemas of all data modelling techniques can be expressed in terms of a set of First Order Predicate Calculus formulae, but for most techniques that are in actual use this can indeed be done

the question of equivalence of schemes $\mathcal{SCH}_1$ and $\mathcal{SCH}_2$ then corresponds to the question whether $\mathbb{L}(\mathcal{SCH}_1)$ is provable from $\mathbb{L}(\mathcal{SCH}_2)$ and vice versa:

$$\mathbb{L}(\mathcal{SCH}_1) \iff \mathbb{L}(\mathcal{SCH}_2)$$

However, in $\mathcal{SCH}_1$ one may have introduced other predicates (other names, other arities) than are present in $\mathcal{SCH}_2$ (and vice versa). Therefore, the need arises to provide some extra formulae to define a translation between these predicates. Therefore, the equivalence of $\mathcal{SCH}_1$ and $\mathcal{SCH}_2$ is defined as:

$$\mathcal{SCH}_1 \equiv \mathcal{SCH}_2 \triangleq \mathbb{L}(\mathcal{SCH}_1) \wedge D_1 \iff \mathbb{L}(\mathcal{SCH}_2) \wedge D_2$$

where D_1 and D_2 are the formulae providing the translation between the two sets of predicates. This leads to the notion of *contextual equivalence*; the set of formulas D_1 and D_2 provide the context of the equivalence. Both D_1 and D_2 are conjunctions of formulae such that the predicates (and functions) provided in $\mathcal{SCH}_2$ are defined in terms of the ones provided in $\mathcal{SCH}_1$ and vice versa. The theories $\mathbb{L}(\mathcal{SCH}_1) \wedge D_1$ and $\mathbb{L}(\mathcal{SCH}_2) \wedge D_2$ should form *conservative extensions* (see e.g. [CK77]) of $\mathbb{L}(\mathcal{SCH}_1)$ and $\mathbb{L}(\mathcal{SCH}_2)$. Proving the logical equivalence of $\mathbb{L}(\mathcal{SCH}_1) \wedge D_1$ and $\mathbb{L}(\mathcal{SCH}_2) \wedge D_2$ is equivalent to proving:

$$\mathbb{I}(\mathbb{L}(\mathcal{SCH}_1) \wedge D_1) = \mathbb{I}(\mathbb{L}(\mathcal{SCH}_2) \wedge D_2)$$

Using *contextual equivalence*, a number of schema equivalence problems become provable using a theorem prover ([Blo93]). Note that in general not all schema equivalence problems are automatically provable, or even decidable.

From the translations between predicates provided by D_1 and D_2 a bijection h as used in the above discussed approaches can be derived. All schemas that can be proven to be equivalent using the approach based on *contextual equivalence* are therefore equivalent in the sense of the first two approaches. However, the reverse does not necessarily hold. For example, consider the schemas depicted in figure 5. The two schemas are isomorphic, but we cannot prove contextual equivalence of the two schemas since this would lead to a naming problem in the conservative extensions. Nevertheless, in these cases equivalence can be proven by extending the name space of roles and types. A name n used in the left schema could be extended to $1.n$, while a name n used in the right schema could be extended to $2.n$.

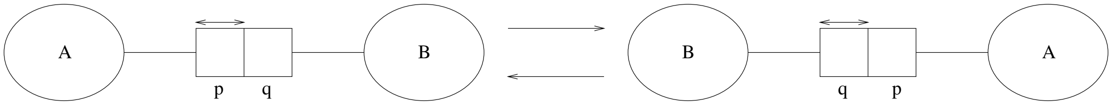


Figure 5: Renaming roles and types

2.1.4 Substitution property

In traditional mathematics, if X and Y are expressions yielding a natural number and we have proven that $X = Y$, then we also know that $X + 1 = Y + 1$ since we are allowed to replace X by Y. This is an example of the substitution property.

Analogously we would like to have a substitution property for schemas. Once we have proven the equivalence of two schemas $\mathcal{SCH}_1$ and $\mathcal{SCH}_2$, we would like to be able to conclude the equivalence of two schemas $\mathcal{SCH}'_1$ and $\mathcal{SCH}'_2$ if all in which they differ from $\mathcal{SCH}_1$ and $\mathcal{SCH}_2$ respectively is a set of schema components X. Without such a property, it is impossible to build a general pool of equivalence preserving transformations, as each time such a transformation is applied the equivalence of the schemas would have to be re-proven. Formally, we would like to have the following property:

Theorem 2.1 (*substitution theorem*) Let $\mathcal{SCH}_1$ and $\mathcal{SCH}_2$ be schemas such that $\mathcal{SCH}_1 \equiv \mathcal{SCH}_2$ and X is a set of schema components.

If $\mathcal{SCH}'_1$ follows from $\mathcal{SCH}_1$ by adding components X, and similarly $\mathcal{SCH}'_2$ from $\mathcal{SCH}_2$ by adding components X, and $\mathcal{SCH}'_1$, $\mathcal{SCH}'_2$ are proper schemas, then:

$$\mathcal{SCH}'_1 \equiv \mathcal{SCH}'_2$$

However, the substitution property does not generally hold for data schemas. For example, in the case of figure 5, suppose we add the schema component $\mathsf{Frequency}(q, 1..3)$; which is a textual representation of a frequency constraint on the role q. Adding this component to the two equivalent schemas results in two non-equivalent schemas! The problem in this case is the fact that the frequency constraint refers to the role q, which has a different semantics in both schemas.

The solution we propose is to require the schemas to provide the contextual equivalence themselves, i.e. as derived types with proper derivation rules. Most modern data modelling techniques allow for the specification of derivation rules. We therefore introduce the notion of *direct equivalence* as:

$$\mathcal{SCH}_1 \equiv \mathcal{SCH}_2 \triangleq \mathbb{L}(\mathcal{SCH}_1) \iff \mathbb{L}(\mathcal{SCH}_2)$$

which equates to:

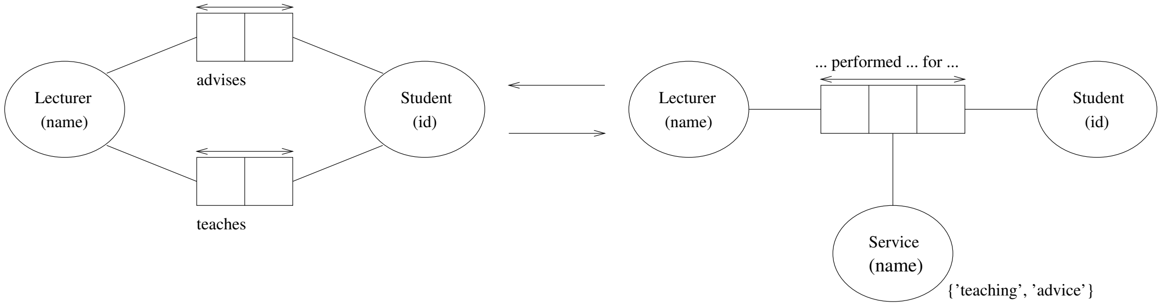
$$\mathcal{SCH}_1 \equiv \mathcal{SCH}_2 \triangleq \mathbb{I}(\mathbb{L}(\mathcal{SCH}_1)) = \mathbb{I}(\mathbb{L}(\mathcal{SCH}_2))$$

Figure 5 is clearly not directly equivalent. As an example of a direct equivalence, consider figure 6, which represents the same universe of discourse as figure 2.

It is not hard to see that the substitution property holds in the case of direct equivalence. Any components added to $\mathcal{SCH}_1$ and $\mathcal{SCH}_2$ now have exactly the same semantics, so if $\mathcal{SCH}'_1$ and $\mathcal{SCH}'_2$ follow from the original schemas by adding components X we have:

$$\mathbb{L}(\mathcal{SCH}_1) - \mathbb{L}(\mathcal{SCH}'_1) = \mathbb{L}(\mathcal{SCH}_2) - \mathbb{L}(\mathcal{SCH}'_2)$$

Since $\mathbb{L}(\mathcal{SCH}_1) \iff \mathbb{L}(\mathcal{SCH}_2)$ we therefore have: $\mathbb{L}(\mathcal{SCH}'_1) \iff \mathbb{L}(\mathcal{SCH}'_2)$.



Lecturer l performed Service se for Student st IFF
(Service se has ServiceName 'teaching' AND Lecturer l teaches Student st) OR
(Service se has ServiceName 'advice' AND Lecturer l advises Student st)

Lecturer l teaches Student s IFF
Lecturer l performed Service 'teaching' for Student s

Lecturer l advises Student s IFF
Lecturer l performed Service 'advice' for Student s

Figure 6: Example of direct equivalent schemas

2.1.5 Strengthening a schema

A schema $\mathcal{SCH}_2$ is stronger than a $\mathcal{SCH}_1$ if each interpretation of $\mathcal{SCH}_2$ is an interpretation of $\mathcal{SCH}_1$, so when:

$$\mathbb{I}(\mathbb{L}(\mathcal{SCH}_1)) \supseteq \mathbb{I}(\mathbb{L}(\mathcal{SCH}_2))$$

This leads to the following definition of a stronger schema:

$$\mathcal{SCH}_1 \preccurlyeq \mathcal{SCH}_2 \triangleq \mathbb{L}(\mathcal{SCH}_1) \Leftarrow \mathbb{L}(\mathcal{SCH}_2)$$

So when $\mathcal{SCH}_1 \preccurlyeq \mathcal{SCH}_2$ we say that $\mathcal{SCH}_2$ is stronger than $\mathcal{SCH}_1$. In the next section we see an example of a schema strengthening transformation.

2.2 Conceptual equivalence

The second class of schema equivalence is based on the relationship between conceptual schemas and a universe of discourse. Most ORM/NIAM based modelling techniques are based on the presumption that when modelling a universe of discourse one actually constructs a grammar of the communication taking place within this universe of discourse. These modelling techniques therefore usually start out from a set of sample sentences describing the universe of discourse. These latter sentences are generally elicited from *domain experts*, in accordance with the following postulate:

> *Domain experts can fully describe their universe of discourse using (semi) natural language; which can sometimes be done in the form of a complete set of significant examples.*

This postulate may seem simple, but it is a crucial base of an increasing number of modern approaches to conceptual modelling ([Nij89], [CY90], [Win90], [Hal95], [Kri94]). In principle, some schema features can never have a set of significant examples (e.g. subtype definitions), but small significant example sets are easy to generate for most constraints. The set of examples is usually gathered by using the so-called telephone paradigm ([Hal95]):

> *Explain your observations to a non-expert via a telephone.*

The set of sentences that follows from this exercise defines a language; the *domain expert language* ([HPW97]). The underlying grammar is referred to as $\mathcal{G}_{\mathcal{EXP}}$. The aim of a conceptual modelling process can now be described as ([HPW97]):

> *Find, within a certain class of sufficiently efficient computable grammars, a grammar which best approximates $\mathcal{G}_{\mathcal{EXP}}$.*

The conceptual schema which follows from a conceptual design procedure in itself defines a grammar $\mathcal{G}_{\mathcal{SCH}}$. All correct ORM/NIAM/ER schemas essentially define such a computable grammar $\mathcal{G}_{\mathcal{EXP}}$. Therefore, the aim of the conceptual modelling process can be re-formulated as:

> *Find, a conceptual schema $\mathcal{SCH}$ such that the associated grammar $\mathcal{G}_{\mathcal{SCH}}$ best approximates $\mathcal{G}_{\mathcal{EXP}}$.*

Please note that for $\mathcal{SCH}$ to be a correct schema certain well-formedness criteria must be met, including restrictions on the verbalisations of the fact types from the universe of discourse ([Hal95]). For example, ORM requires the verbalised fact types to be elementary, i.e. the resulting facts should not be splittable into fact types of lower arity.

If we would have a formal notion of a distance between grammars:

$$\mathsf{Distance}(\mathcal{G}_{\mathcal{SCH}}, \mathcal{G}_{\mathcal{EXP}}) = \delta$$

then we would be able to decide between two equivalent schema alternatives for a given universe of discourse. Let $\mathcal{SCH}_1$ and $\mathcal{SCH}_2$ be two equivalent schema alternatives for a universe of discourse with expert language $\mathcal{G}_{\mathcal{EXP}}$, then $\mathcal{SCH}_1$ is a more *natural* description of the universe of discourse if:

$$\mathsf{Distance}(\mathcal{G}_{\mathcal{SCH}_1}, \mathcal{G}_{\mathcal{EXP}}) < \mathsf{Distance}(\mathcal{G}_{\mathcal{SCH}_2}, \mathcal{G}_{\mathcal{EXP}})$$

The schemas are conceptually equivalent when they have the same distance to the expert language:

$$\mathsf{Distance}(\mathcal{G}_{\mathcal{SCH}_1}, \mathcal{G}_{\mathcal{EXP}}) = \mathsf{Distance}(\mathcal{G}_{\mathcal{SCH}_2}, \mathcal{G}_{\mathcal{EXP}})$$

Verifying such equivalences formally will most likely remain hard, if not impossible. The key problem being the fact that determining the equivalence between grammars is known to be a hard problem. Nevertheless, this definition of *naturalness* of a conceptual schema with respect to a given universe of discourse does allow us to make better design decisions in a conceptual schema design procedure when considering alternative schemas.

3 Example Transformations

In this section we discuss some example applications of schema transformations followed by their underlying schema transformations. For a more complete treatise of such transformations, the reader is referred to [Hal89], and [Hal95]. In [Hal89] proofs of equivalences can be found as well.

3.1 A simple schema transformation

As a first simple example, consider the medical report shown below. Here a tick in the appropriate column indicates that a patient smokes or drinks. Since both these “vices” can impair health, doctors are often interested in this information.

Patient#	Patient name	Smoker?	Drinker?
1001	Adams, A		√
1002	Bloggs, F	√	√
1003	Collins, T		

Figure 7 shows one conceptual schema for this universe of discourse, together with the sample population. The black dot is a mandatory role constraint (each patient has a name). Here two optional unaries are used for the smoker-drinker facts. Instead of using unaries, we may model the smoker-drinker facts using two functional binaries: Patient has SmokerStatus; Patient has DrinkerStatus.

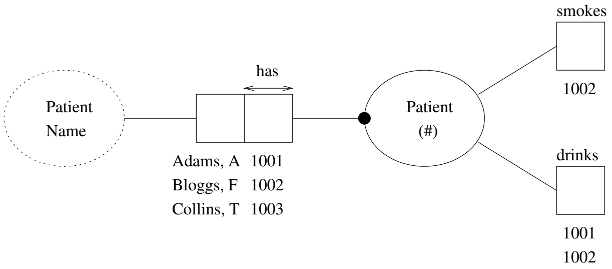


Figure 7: One way of modelling the hospital universe of discourse

A third way to model this is generalise the smoking and drinking fact types into a single binary, introducing the object type Vice (S = Smoking, D = Drinking) to maintain the distinction (see figure 8). Intuitively, most people would consider the schemas of figure 7 and 8 to be equivalent. Formally, this intuition can be backed by introducing Vice as a derived type to the schema of figure 8, and by specifying exactly how the fact types of each can be translated into the fact types of the other. For example, facts expressed in the first model may be expressed in terms of the second model using the translations:

Patient p smokes IFF Patient p indulges in Vice ‘S’
Patient p drinks IFF Patient p indulges in Vice ‘D’

and facts in the second model may be expressed in the first using the translation:

Patient p indulges in Vice v IFF (Patient p smokes AND Vice v has ViceCode 'S')
OR (Patient p drinks AND Vice v has ViceCode 'D')

Even though the second schema might sometimes be considered more ”natural”, the first schema will usually be more efficient to implement with respect to the number of relational tables.

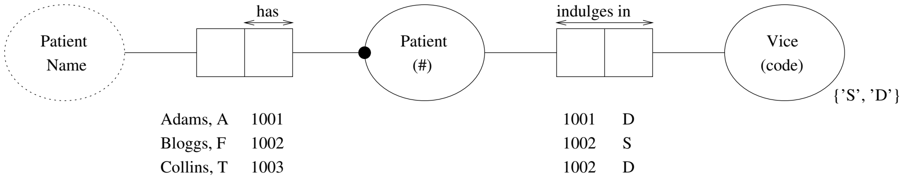


Figure 8: Another way of modeling the hospital universe of discourse

3.2 Predicate specialisation and generalisation

In this subsection we consider a class of schema transformations known as predicate specialisation, as well as its inverse, predicate generalisation. If two or more fact types may be thought of as special cases of a more general fact type then we may replace them by the more general fact type, as long as the original distinction can be preserved in some way. For example, if we transform the schema of figure 7 into that of figure 8, we generalise smoking and drinking into indulging in a vice, where vice has two specific cases. If we transform in the opposite direction, we specialise indulging in a vice into two fact types, one for each case.

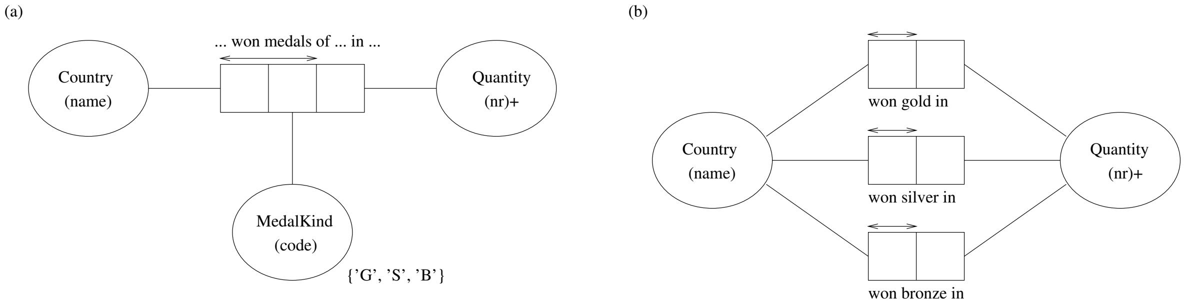


Figure 9: Olympic Games universe of discourse

A fact type may be specialised if a value constraint or a frequency constraint indicates that it has a finite number of cases. Examples with value constraints are more common, so we examine these first. The drinker-smoker example provides one illustration, where Vice has the value constraint {'S','D'}. As another example, consider the Olympic Games schema depicted in figure 9 (a). Because there are exactly three kinds of medal, the ternary may be specialised into three binaries, one for each medal kind, as shown in figure 9 (b).

You may visualise the transformation from schema (a) into schema (b) thus: when the object type MedalKind is absorbed into the ternary fact type, it breaks it up (or specialises it) into the three binaries. The reverse transformation from (b) to (a) generalises the three binaries into the ternary by extracting the object type MedalKind.

Notice that in the vices example, a binary is specialised into unaries. With the games example, a ternary is specialised into binaries. In general, when an n-valued object type is absorbed into a fact type, the n specialised fact types that result each have one less role than the original (since the object type has

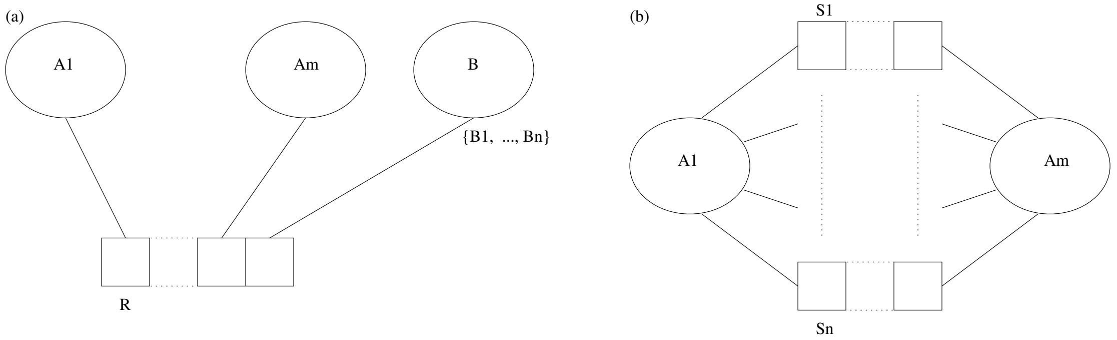


Figure 10: Object type absorption transformation

been absorbed). This general result is set out in figure 10. In this case, the answer to the question which alternative will be most efficient depends on such things as the access and data profile of the application in question. Since we consider these transformations in terms of data schemas in a conceptual modelling technique, we can actually involve end users (domain experts) to give an indication of the access and data profiles. This will generally be harder when only a database schema is available.

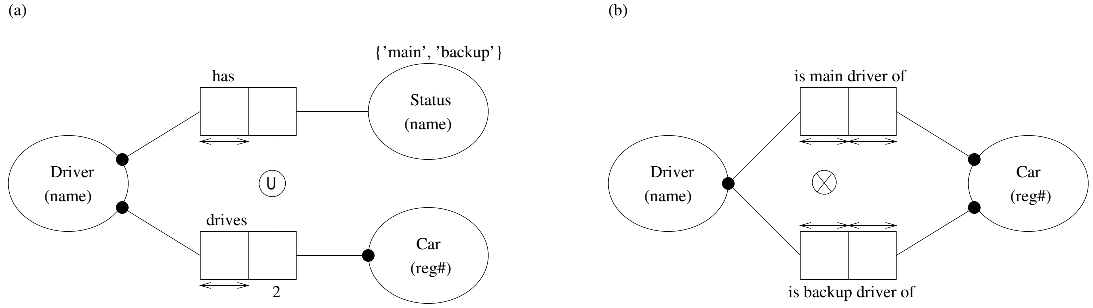


Figure 11: The drives fact type is specialised by absorbing Status

As an introduction to a second schema transformation, consider figure 11. The two schemas provide alternative models for a fragment of a car rally application. The circled "u" is an external uniqueness constraint (each Status-Car combination applies to at most one driver) and the "2" is a frequency constraint. Each car in the rally has two drivers (a main driver and a backup driver), and each person drives exactly one car. Schema (a) is transformed into schema (b) by absorbing the object type Status into the drives fact type, specialising this into the main driver and backup driver fact types. The circled "X" is an exclusion constraint (no driver is both a main driver and a backup driver). The reverse transformation generalises the specific driver fact types into the general one by extracting the object type Status. Since this object type appears in a different fact type, this equivalence does not fit the pattern of the transformation provided in figure 10.

Note how the constraints are transformed. The external uniqueness constraint in (a) says that each car has at most one main driver and at most one backup driver. This is captured in (b) by the uniqueness constraints on the roles of Car. The uniqueness constraint on the drives fact type in (a) corresponds in (b) to the uniqueness constraints on the roles of Driver. The uniqueness constraint on the status fact type in (a) is captured by the exclusion constraint in (b). The mandatory and frequency constraints on Car's role in (a) require the two mandatory role constraints on Car in (b). Finally, the mandatory role constraints on Driver in (a) are catered for in (b) by the disjunctive mandatory role constraint (shown explicitly here). This example illustrates our second specialisation/generalisation transformation as shown in figure 12. In our example, A, B and C correspond to Driver, Status and Car; and the equality constraint is implied by the two mandatory role constraints on Driver.

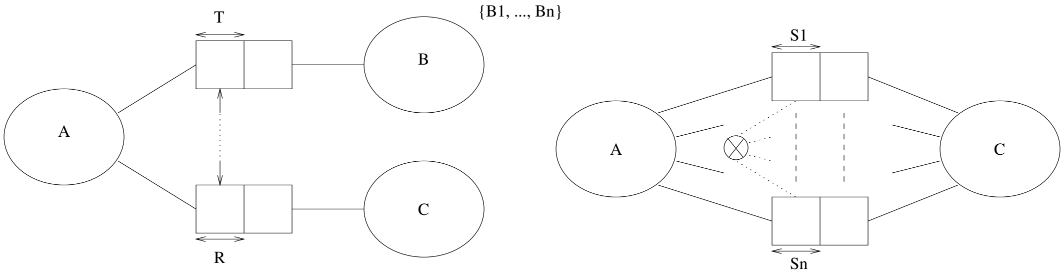


Figure 12: Another absorption transformation

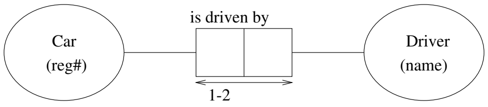


Figure 13: Can the predicate be specialised?

Sometimes we may wish to transform a schema into another that is not quite equivalent. The reason for doing this may be that by slightly changing the semantics of the schema, a much more efficient storage becomes possible. For example, suppose that in our car rally application we limit each car to at most two drivers, but do not classify the drivers in any meaningful way (e.g. as main or backup drivers). Let us also remove the constraint that drivers may drive only one car. This situation is schematised in figure 13.

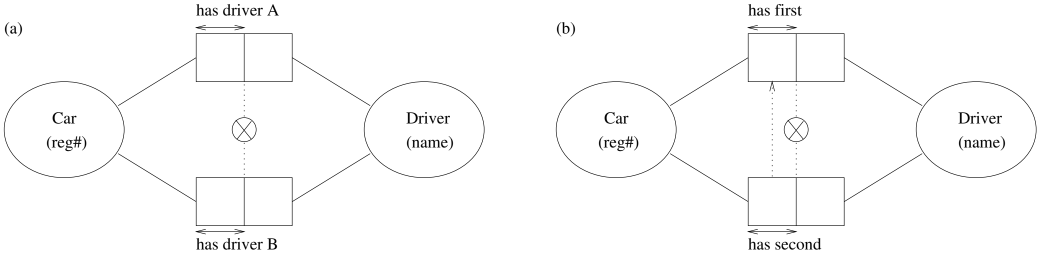


Figure 14: Two ways of strengthening

Although no Status object type is present, the frequency constraint in Figure 13 tells us that each car has at most two drivers. This enables us to introduce an artificial distinction to specialise the fact type into two cases, as shown in figure 14. Since this distinction is not present in the original schema, the alternatives shown in figure 14 are not equivalent to the original; they are in fact stronger. Schema (b) is actually stronger than schema (a). If the Car role in figure 13 is mandatory, then the Car roles in (a) are disjunctively mandatory, and the top Car role in (b) is mandatory (which then implies the subset constraint (shown as a dotted arrow)). In an application where other facts are stored about cars but not about drivers, one of these alternatives may well be chosen to avoid a separate table being generated for car-driver facts when the schema is mapped (the specialised fact types are functional rather than m:n). In practice, schema (b) of figure 14 would normally be chosen. Transforming from the original schema to one of those in figure 14 strengthens the schema by adding information. Transforming in the opposite direction weakens the schema by losing information. Any such transformations which add or lose information should be the result of conscious decisions which are acceptable to the client (for which the application is being modeled). The rationale behind such a transformation is that although the number of allowed populations might decrease, the resulting schema might be much more efficient to implement. If the excluded populations will actually never have to be stored, than it certainly makes sense to make such a transformation.

This example illustrates our third specialisation/generalisation transformation, as shown in figure 15. In practice, the transformation is usually performed from right to left (strengthening the schema), with a subset constraint added from the first role of S2 to that of S1 if $n = 2$. In cases like this, end user participation in the optimisation process is almost essential as only the end users can really decide whether the strengthening of a schema, and thus limitation of the underlying grammar, outweighs the expected efficiency gains.

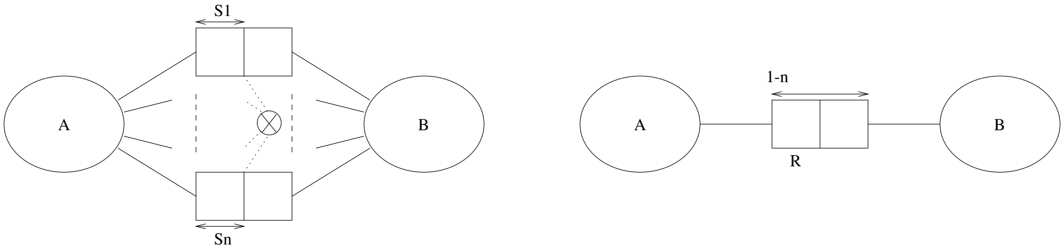


Figure 15: The left-hand schema is stronger than the right-hand schema

This completes our discussion of a small potpourri of schema transformations. As stated before, a wider range of schema transformations can be found in [Hal89] and [Hal95].

4 Conceptual Schema Universe

Even though formalisations of ORM have been published before ([Hal89], [HW93], [HPW93], [BBMP95], [HP95]), we provide such a formalisation once more to be self contained. However, in this formalisation we limit ourselves to syntactical issues only. Issues regarding semantics can be found in the referenced publications. Furthermore, a related formalisation is provided in [CP96]. The formalisation given there is based on a smaller number of basic concepts while maintaining the full expressibility of ORM.

We also introduce the concept of a universe of ORM models which is to be used as a medium for schema transformations. The next section then discusses what a schema version is within this ORM universe. The notion of having a universe of data schemas, and the data schema of a universe of discourse describing a journey (evolution) through this universe as a sequence of versions has been introduced before in the field of evolving information systems ([PW95a], [PW94], [Pro94]).

4.1 Information structure universe

We assume the reader has a basic working knowledge of the concepts underlying ORM or ER. A conceptual schema is presumed to consist of a set of types $\mathcal{TP}$. This set can be divided in three subclasses. The first class is the set of object types (entity types) $\mathcal{OB}$. Within this class a subclass of value types $\mathcal{VL} \subseteq \mathcal{OB}$ can be distinguished. Instances from value types usually originate from some underlying domain such as strings, natural numbers, audio, video, etc. Later a function is introduced that assigns a domain (set of values) to each value type. A separate class of types are the relationship types $\mathcal{RL}$. We now have the following type classes:

$$\begin{aligned} \mathcal{TP} &\triangleq \mathcal{OB} \cup \mathcal{RL} \\ \mathcal{VL} &\subseteq \mathcal{OB} \end{aligned}$$

An example ORM conceptual schema can be found in figure 16. For this schema we have:

$$\begin{aligned} \mathcal{VL}_i &\triangleq \{\mathsf{DateCode}, \mathsf{\$Value}, \mathsf{HouseNr}, \mathsf{Zipcode}, \mathsf{EstimationNr}\} \\ \mathcal{OB}_i &\triangleq \{\mathsf{Date}, \mathsf{MoneyAmt}, \mathsf{House}, \mathsf{Estimation}, \mathsf{DateCode}, \mathsf{\$Value}, \mathsf{HouseNr}, \mathsf{Zipcode}, \mathsf{EstimationNr}\} \\ \mathcal{RL}_i &\triangleq \{\mathsf{was\ on}, \mathsf{assessed\ value\ as}, \mathsf{is\ for}, \mathsf{has}, \mathsf{is\ in\ region\ with}\} \end{aligned}$$

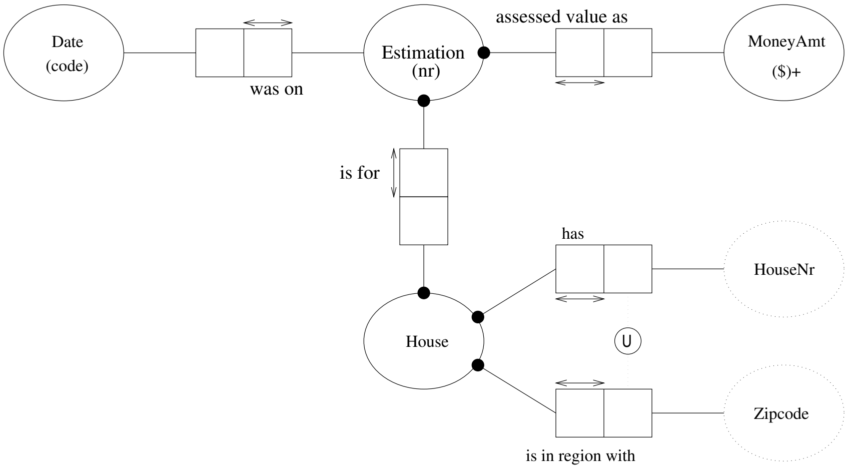


Figure 16: Example information structure

where the index i indicates that we are dealing with schema version i.

Types are interrelated in a number of different ways. In the ORM universe, we have the following relationships between types:

1. Relationship types consist of a number of *roles* (also referred to as *predicators*). The roles of an ORM schema are captured in the set $\mathcal{RO}$. The roles in $\mathcal{RO}$ are distributed among the relationship types by the partition: $\mathsf{Roles} : \mathcal{RL} \rightarrow \wp^+(\mathcal{RO})$. (Note that $\wp^+(\mathcal{RO})$ yields all non-empty subsets of $\mathcal{RO}$). The object types playing each of the roles are yielded by the function $\mathsf{Player} : \mathcal{RO} \rightarrow \mathcal{TP}$.

   For the Roles function, we have the following 'inverse' function returning the relationship to which a given role belongs: $\mathsf{Rel} : \mathcal{RO} \rightarrow \mathcal{RL}$, which is defined by: $\mathsf{Rel}(r) = f \iff r \in \mathsf{Roles}(f)$.

2. The inheritance of properties is captured by the $\mathsf{SubOf} \subseteq \mathcal{OB} \times \mathcal{OB}$ relation; with the intuition:

   if $x\,\mathsf{SubOf}\,y$ then 'the population of x is a subset of the population of y'

   One of the reductions in the number of basic concepts for ORM models as reported in [CP96] is the integration of traditional subtyping and polymorphism ([HP95]), also known as categorisation in EER ([EWH85]). This is the SubOf relationship, which is strictly concerned with inheritance of populations between type. For a more detailed discussion on the relationship to traditional subtyping and polymorphism, refer to [CP96].

3. An information structure version within the universe has to adhere to certain well-formedness rules. An information structure version is fully determined by the set of types contained in it. Therefore we presume to have the following well-formedness predicate: $\mathsf{IsInfStr} \subseteq \wp(\mathcal{TP})$. The exact definition of this predicate follows in the next section.

An information structure universe $\mathcal{U}_{\mathcal{IS}}$ over a set of domains $\mathcal{D}$ is determined by the components of the following tuple:

$$\langle \mathcal{RL}, \mathcal{OB}, \mathcal{RO}, \mathsf{SubOf}, \mathsf{Roles}, \mathsf{Player}, \mathsf{IsInfStr} \rangle$$

The first 3 components provide the types and roles present in the information structure, and the last 4 components describe their mutual relationships; providing the 'fabric' of the information structure.

Relationship types and object types are exclusive:

[ISU1] (*type exclusion*) $\mathcal{RL} \cap \mathcal{OB} = \varnothing$

The careful reader may now raise the question: *whatever happened to objectification/nesting of relationship types?* This question will be answered at the end of this section.

For the relationships between types some well-formedness rules apply. Firstly, the inheritance hierarchy is both transitive and irreflexive:

[ISU2] (*transitive*) $x \,\mathsf{SubOf}\, y \,\mathsf{SubOf}\, z \Rightarrow x \,\mathsf{SubOf}\, z$

[ISU3] (*irreflexive*) $\neg x \,\mathsf{SubOf}\, x$

The separation between value types and non value types must be maintained in the inheritance hierarchy:

[ISU4] (*separation*) If $x \,\mathsf{SubOf}\, y$, then: $x \in \mathcal{VL} \iff y \in \mathcal{VL}$.

From the transitive SubOf relation we can derive the intransitive one (SubOf_1) as follows:

$$x \,\mathsf{SubOf}_1\, y \iff x \,\mathsf{SubOf}\, y \land \neg\exists_z \left[x \,\mathsf{SubOf}\, z \,\mathsf{SubOf}\, y\right]$$

The finite depth of the identification hierarchy in ORM is expressed by the following schema of induction:

[ISU5] (*identification induction*)

If F is a property for object types, such that:

$$\text{for any } y, \text{ we have: } \forall_{x:y \,\mathsf{SubOf}_1\, x} \left[F(x)\right] \Rightarrow F(y)$$

then $\forall_{x\in\mathcal{OB}} \left[F(x)\right]$

The latter axiom was not explicitly present in the previous discussions, but was always presumed to be implicitly present. In this paper, it is only stated for reasons of completeness. Note that the identification induction schema can be proven from the properties of SubOf if the axiomatic setup were extended with the natural numbers and their axioms (in particular natural number induction). In our formalisation, however, we do not presume the presence of the natural numbers.

4.2 Conceptual schema universe

A conceptual schema universe $\mathcal{U}_{\mathcal{CS}}$, over a set of concrete domains $\mathcal{D}$, for ORM is now identified by the following components:

$$\langle \mathcal{U}_{\mathcal{IS}}, \mathcal{CN}, \mathcal{DR}, \mathsf{Dom}, \mathsf{IsSch} \rangle$$

These components are:

1. Each conceptual schema contains an information structure as its backbone. So the information structure universe is part of the schema universe.
2. The first set of rules are the constraints. They must be taken from the set of constraints $\mathcal{CN}$. As a textual language to define constraints one may choose e.g. FORML or LISA-D. ORM also has a graphical representation for the most generally used constraints such as uniqueness, mandatory (totality) and exclusiveness constraints.
3. As some of the types in the information structure are derivable from other types, conceptual schemas may contain derivation rules. The set of derivation rules is provided as $\mathcal{DR}$.

   In this article we do not elaborate on a language for the definition of these rules. However, we do presume the existence of two functions on derivation rules:

$$\mathsf{Defines} : \mathcal{DR} \to \mathcal{TP}$$

yielding the relationship type that is being defined by a derivation rule, and

$$\mathsf{Depends} : \mathcal{DR} \rightarrow \wp(\mathcal{TP})$$

returning the types to which the derivation rule (directly) refers.

4. All atomic value types have associated a domain of pre-defined denotable values. For instance, the value type Nat no will typically have associated some subset of the natural number. Formally, the possible relationships between the atomic value types and the domains is provided as: $\mathsf{Dom} = (\mathcal{AT} \cap \mathcal{VL}) \rightarrow \mathcal{D}$, where $\mathcal{D}$ denotes the set of domains.

5. Finally, not all schemas in the universe of schemas spanned by the above components correspond to a proper schema. Therefore the predicate IsSch is introduced to distinguish the proper schema versions. Its formal definition will follow below.

4.3 Objectified relationship types and other complex types

The first axiom we formulated on type classes was the fact that the set of relationship types is disjoint from the set of object types. This requirement seems to be in contradiction with the existence of so-called objectified relationship types. In ORM, as well as ER, one can objectify relationship types and as of then treat them as if they were object types. An example of this is shown in the left hand side of figure 17. An Enrollment is both used as a relationship type between Student and Subject and as an object type playing a role in the resulted in relationship type. It is a well known fact that semantically objectifications are equivalent to co-referenced object types (i.e. object types that require a combination of two or more reference types to identify them). In the right hand side of figure 17, the equivalent schema using a co-referenced object type Enrollment is shown.

In this article we will treat all objectifications as if they were co-referenced object types. The reason for doing this is that it reduces the complexity of the formalisation; it leads to a reduction of the number of type classes. Nevertheless, when modelling a universe of discourse one would still like to be be able to use objectification simply because it sometimes leads to a more natural way of modelling domains (see subsection 2.2). For example, for some given universe of discourse, the top schema of figure 17 may be much more natural than the bottom schema. Although the two depicted schemas are mathematically equivalent, they are not likely to be equally preferable.

Objectification may be seen as a form of abstraction. In terms of the example, the Enrollment objectification is an abstraction of the underlying relationship types is in and for. Objectification, and abstraction in general, can therefore be seen as a third dimension for a flat conceptual model. The flat model consists of relationship types and object types as defined for the information structure universe, while the third dimension is concerned with abstractions, like objectification. This third dimension is concerned only with abstraction, and therefore has no influence on mathematical equivalence. Therefore, for schema transformations aimed at optimisation of data schemas one may ignore abstraction mechanisms like objectifications.

In [Cam94], [CP96] and [CHP96] a more elaborate discussion is provided on abstractions as a third dimension for flat conceptual modelling. Finally, the above discussion for objectification also holds for other complex types like set types, bag types and sequence types. On the left hand side of figure 18 an example of a set type is shown. A Convoy consists of a set of ships, where each ship is identified by a unique code. Convoys are not identified by some surrogate code but by the actual set of ships in the convoy. In the right hand side of this figure the flat view, without abstraction, is shown. The circled eu signifies an *existential uniqueness* constraint ([HW94], [HW97]). The existential uniqueness constraint in this case enforces the rule that no two different convoys have the same set of ships associated. In this example, Convoy is a set type and can be seen as an abstraction of the underlying consisting of relationship type.

5 Versioning within a Universe

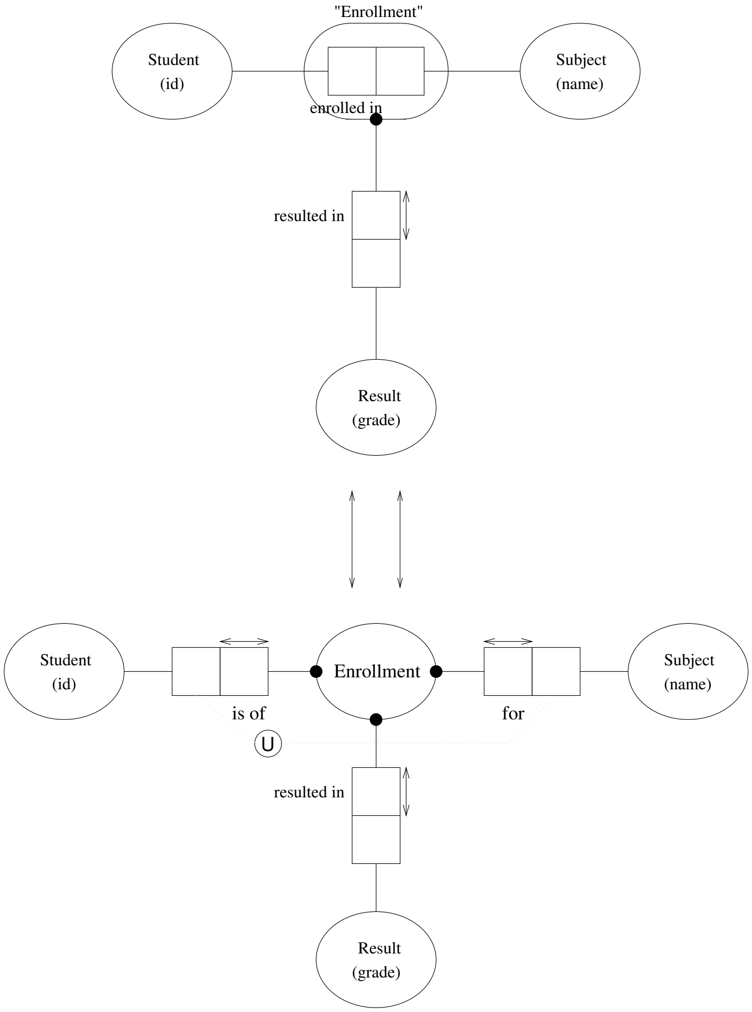


Figure 17: Example objectification

Once the user community has agreed that a certain conceptual schema is the preferred representation of the universe of discourse, that schema should remain fixed during the optimisation process (unless a new optimisation is displayed to and then preferred by the user). The schema may be implemented differently, but the approved conceptual schema should remain as is from a user's point of view. This means that the types of the original schema should not be removed during schema optimisations. This is why we will make a distinction between so called *conceptual types* and *internal types*. The types of the original conceptual schema are obviously all marked as conceptual types. Any types introduced for schema optimisation purposes are marked as internal, and should not be 'accessible' directly by users.

When a conceptual type is transformed to a (set of) internal type(s), a derivation rule is specified that derives the population of the transformed type. This means that the user can still access the information in the database in terms of the conceptual schema. For example using a conceptual query language like LISA-D ([HPW93]). Conversely, when updating the database, the user would like to do this in terms of the conceptual schema as well. This means that for the transformed conceptual types, update rules must be specified which translate updates of the conceptual types to updates of the proper internal types. In a later section we provide extra rules to enforce certain well-formedness rules on the derivation rules in combination with the update rules (e.g. to avoid view update problems).

In the remainder of this section we discuss what constitutes a schema version within a schema universe. To identify a schema version $\mathcal{SCH}_i$, the following components are required:

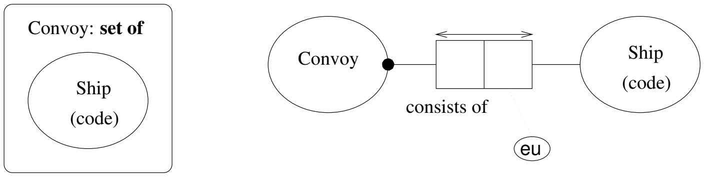


Figure 18: The convoy set type example

1. A set of types $\mathcal{TP}_i \subseteq \mathcal{TP}$ which identifies the information structure version.
2. The internal types are types: $\mathcal{IT}_i \subseteq \mathcal{TP}_i$.
3. A set of constraints $\mathcal{CN}_i \subseteq \mathcal{CN}$ that should hold for this version.
4. A set of derivation rules $\mathcal{DR}_i \subseteq \mathcal{DR}$ for the derivable types in this version.
5. A set of update rules $\mathcal{UR}_i \subseteq \mathcal{DR}$.
6. A domain mapping $\mathsf{Dom}_i \in \mathsf{Dom}$ for the value types of this version.

The set of types $\mathcal{TP}_i$ spans the information structure version $\mathcal{IS}_i$ for this conceptual schema version. Sometimes we have to refer to a specific type class of schema components within one information structure version. These classes are derived as follows:

$$\begin{aligned}
\mathcal{OB}_i &\triangleq \mathcal{TP}_i \cap \mathcal{OB} \\
\mathcal{VL}_i &\triangleq \mathcal{TP}_i \cap \mathcal{VL} \\
\mathcal{RL}_i &\triangleq \mathcal{TP}_i \cap \mathcal{RL} \\
\mathcal{RO}_i &\triangleq \bigcup_{r \in \mathcal{RL}_i} \mathsf{Roles}(r)
\end{aligned}$$

5.1 Well-formedness of information structure versions

We now first focus on well-formedness of information structure versions. These rules mainly deal with completeness of a single version.

If a role is part of the current version, then so must the player of the role:

[ISV1] $p \in \mathcal{RO}_i \Rightarrow \mathsf{Player}(p) \in \mathcal{TP}_i$

Types in this version that are lower in the type hierarchy (SubOf) must have at least one forefather present:

[ISV2] $x \in \mathcal{OB}_i \wedge x \,\mathsf{SubOf}\, z \Rightarrow \exists_{y \in \mathcal{TP}_i} \left[x \,\mathsf{SubOf}\, y \right]$

A schema version should be a connected graph:

[ISV3] The graph $\langle \mathcal{TP}_i, E_i \rangle$ where

$$E_i = \left\{ \langle \mathsf{Player}(p), \mathsf{Rel}(p) \rangle \;\middle|\; p \in \mathcal{RO}_i \right\} \cup \left\{ \langle x, y \rangle \;\middle|\; x \,\mathsf{IdfBy}\, y \right\}$$

should be connected.

Together the ISV axioms define the IsInfStr predicate on sets of types:

Definition 5.1
Let $X \subseteq \mathcal{TP}$, then

$\mathsf{IsInfStr}(X) \triangleq$ the information structure version spanned by X adheres to the ISV axioms

□

5.2 Conceptual schema versions

For schema versions well-formedness rules can be formulated as well. In this article, we do not discuss all rules that could be formulated, but limit ourselves to the bare essentials. For example, in previous formalisations one can find rules requiring the existance of proper identification schemes (reference schemes) that provide for all types a proper denotation of their instances in terms of values. An identification scheme is a property that is relevant for a data schema that is a conceptual schema, and does therefore not play a role during data schema optimisations.

The following rules are relevant for schema versions during an optimisation process. Domain assignment for value types within a version must be complete:

[CSV1] (*complete domain assignment*) $\mathsf{Dom}_i : (\mathcal{VL}_i \cap \mathcal{BS}) \rightarrow \mathcal{D}$

Only one derivation (and update) rule can be defined for a type:

[CSV2] (*unique rules*) If $r, s \in \mathcal{UR}_i$ or $r, s \in \mathcal{DR}_i$, then:

$$\mathsf{Defines}(r) = \mathsf{Defines}(s) \Rightarrow r = s$$

Finally, an update rule must be specified for all the internal types which are populatable:

[CSV3] (*update rule completeness*) $\left\{\mathsf{Defines}(r) \;\middle|\; r \in \mathcal{UR}_i\right\} = \mathcal{IT}_i$

As stated before, more rules could be added but have been omitted. For a more elaborate discussion of well-formedness of ORM model versions, refer to [HW93], [HPW92b], [BBMP95], or [HP95]. In actual fact, some of the extra rules that could be added are based on which particular ORM version (or ER version for that matter) one uses for conceptual modelling.

We are now finally in a position to define exactly what a proper ORM schema is:

Definition 5.2
Let $\mathcal{SCH}_i \triangleq \langle \mathcal{TP}_i, \mathcal{IT}_i, \mathcal{CN}_i, \mathcal{DR}_i, \mathcal{UR}_i, \mathsf{Dom}_i \rangle$ *such that:*

$$\mathcal{TP}_i \subseteq \mathcal{TP},\ \mathcal{IT}_i \subseteq \mathcal{TP}_i,\ \mathcal{CN}_i \subseteq \mathcal{CN},\ \mathcal{DR}_i \subseteq \mathcal{DR},\ \mathcal{UR}_i \subseteq \mathcal{UR},\ \textit{and}\ \mathsf{Dom}_i : \mathcal{VL} \rightarrowtail \mathcal{D}$$

then:

$$\mathsf{IsSch}(\mathcal{SCH}_i) \triangleq \mathcal{SCH}_i \text{ adheres to the CSV axioms} \land \mathsf{IsInfStr}(\mathcal{TP}_i)$$

□

The set of schemas in a conceptual schema universe is now defined as:

$$\mathcal{SCH} = \wp(\mathcal{TP}) \times \wp(\mathcal{TP}) \times \wp(\mathcal{CN}) \times \wp(\mathcal{DR}) \times \wp(\mathcal{DR}) \times \mathsf{Dom}$$

Note that this carrier set contains both correct and incorrect schemas.

6 Formalisation of Transformations

In this section we discuss a language to define the actual schema transformations. A general format is introduced in which classes of schema transformations can be defined using a so called *data schema transformation scheme*. When applying such a transformation scheme to a concrete data schema, the transformation scheme is interpreted in the context of that schema, leading to a concrete transformation.

We do not provide a fully elaborated and formalised language to specify transformation schemes. Doing so would take up too much space and is not expected to lead to a better understanding of the problems addressed in this article. We do, however, discuss an example of a transformation scheme and an application to an existing data schema.

6.1 A language for transformation schemes

A schema transformation scheme is typically centered around a set of parameters. When applying such a scheme on an actual data schema, these parameters have to be instantiated with actual components from the data schema that is to be transformed.

```
Transformation schema OTEmission $(x!n, (r!n)!m, s!n, t, y, v, w, l, d, i!m)$;
  Object types:
    $x!n, y, l$;

  Value types:
    $l : d$;

  Relationship types:
    $(f = [(x : r)!n])!m$,
    $g = [(x : s)!n, y : u]$,
    $h = [y : v, l : w]$;

  Constraints:
    c1: UNIQUE $\{v\}$;
    c2: UNIQUE $\{w\}$;
    c3: MANDATORY $\{v\}$;
    c4: EACH $l$ IS IN $i!m$;

  From:
    $f!m$;

  To:
    $y, l, d, g, h$, c1, c2, c3, c4;

  Derivation rules:
    $(f = \text{PROJ}[(r = s)!n]\ \text{SEL}[u = v, w = i]\ g\ \text{JOIN}\ h)!m$;

  Update rules:
    $h = \text{UNION OF}(\text{PROJ}[(s = r)!n, u = \text{Val}(y, i)] f)!m$;
    $g = \{\langle v = \text{Val}(y, i), w = i\rangle!m\}$

End Transformation schema.
```

Table 1: An example transformation scheme

An example transformation schema is given in table 6.1. It is the textual denotation of the schema transformation scheme underlying the example transformation depicted in figure 10 (going from (b) to (a)). The parameters to this transformation schema are $x!n, (r!n)!m, s!n, t, y, v, w, l, d, i!m$. This is a set of parameters with a variable size. The expression $x!n$ denotes the list of parameters $x_1, \ldots, x_n$, where n is not a-priori known. Analogously, $(r!n)!m$ denotes the sequence of parameters:

$$r_{1,1}, \ldots, r_{n,1}, \ldots, \ldots, r_{1,m}, \ldots, r_{n,m}$$

So this transformation scheme takes $n + nm + n + 4 = n(m + 2) + 4$ parameters, where n and m are to be determined at the time of application. In figure 19 we have depicted a graphical representation (omitting derivation and update rules) of this transformation scheme.

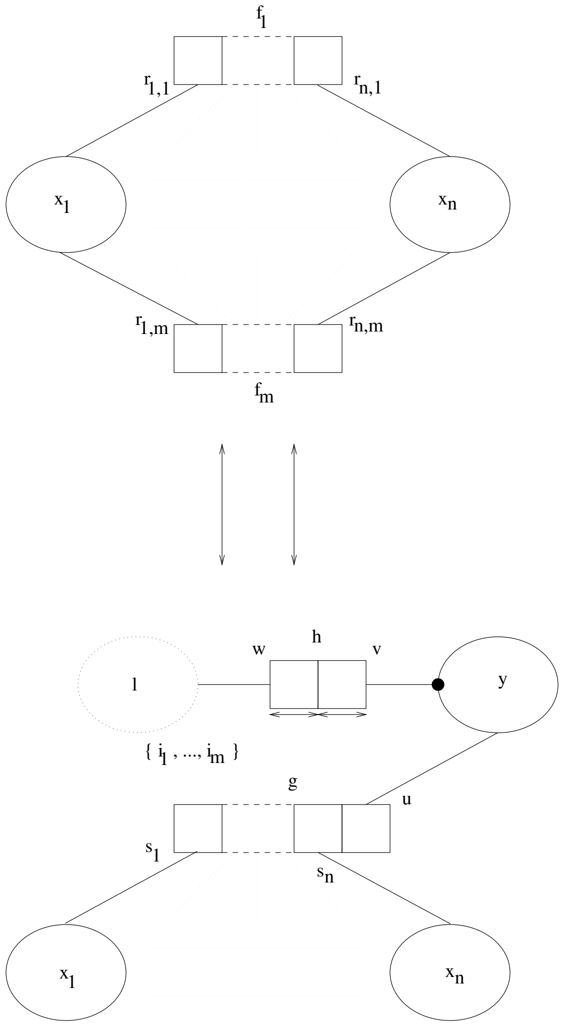


Figure 19: Graphical representation of transformation scheme

Please note that relationship h between object type y and value type l provides the identification for object type y. This relationship is normally implicitly represented by placing (l) inside the ellipse for object type y. For example, in figure 9, the MedalKind object type is identified via the value type code. The relationship between this object type and the value type is not drawn explicitly.

A concrete example of an application of the above transformation scheme would be:

$OTEmission($ [Country, Quantity],
[[won-gold-in-1, won-gold-in-2], [won-silver-in-1, won-silver-in-2], [won-bronze-in-1, won-bronze-in-2]],
[won-medals-of-in-1, won-medals-of-in-3],
won-medals-of-in-2, MedalKind, MedalKind.code-1, MedalKind.code-2, char,
['G', 'S', 'B'])

which is the textual representation of the transformation from (b) to (a) in figure 9. Here we have used the convention that won-gold-in-1, won-gold-in-2 refer to the first and second roles of the fact type labelled won gold in respectively. Furthermore, MedalKind.code-1 and MedalKind.code-2 refer to the roles of the relationship type that is implicitly present between the object type MedalKind and the value type code. This relationship type in itself will be referred to by MedalKind.code.

This leads to the following instantiation of the variables for the transformation scheme:

x_1 = Country	x_2 = Quantity	$r_{1,1}$ = won-gold-in-1
$r_{2,1}$ = won-gold-in-2	$r_{1,2}$ = won-silver-in-1	$r_{2,2}$ = won-silver-in-2
$r_{1,3}$ = won-bronze-in-1	$r_{2,3}$ = won-bronze-in-2	s_1 = won-medals-of-in-1
s_2 = won-medals-of-in-3	t = won-medals-of-in-2	y = MedalKind
u = MedalKind.code-1	v = MedalKind.code-2	l = code
d = char	i_1 = 'G'	i_2 = 'S'
i_3 = 'B'		

This allows us to further fill in the transformation scheme. The Object types statement simply requires that $x_1, \ldots, x_n, y, l$ are object types in the ORM universe. In our case we have to verify that Country, Quantity, MedalKind and code are object types; which they indeed are. By the Value types statement it is required that l be a value type with underlying domain d.

A Relationship types statement requires the presence of a proper relationship type in the universe of ORM schemes. In our case we have the requirements:

1. for each $1 \leq i \leq m$ there is a relationship type f_i in the ORM universe such that we have Roles$(f_i) = \{r_{1,i}, \ldots, r_{n,i}\}$ and furthermore $\forall_{1 \leq j \leq n}$ [Player$(r_{j,i}) = x_j$].
2. there is a relationship type g in the ORM universe such that Roles$(g) = \{s_1, \ldots, s_n\}$ and furthermore $\forall_{1 \leq j \leq n}$ [Player$(s_j) = x_j$].
3. a relationship type h exists in the ORM universe such that Roles$(h) = \{v, w\}$, Player$(v) = y$ and Player$(w) = l$.

It is easy to verify that this holds for the running example with:

f_1 = won-gold-in, f_2 = won-silver-in, f_3 = won-bronze-in, g = won-medals-of-in, h = MedalKind.code

With the Constraints statements we capture the requirement that there exists a constraint $c4$ in the universe with definition EACH l IS IN $i!m$. In our example case this becomes

EACH code IS IN 'G', 'S', 'B'

The constraints $c1, c2$, and $c3$, are needed to ensure that value type l can be used to properly identify the instances of entity type y. This is part of the graphical convention regarding the use of (l), in the example this is the relation between (code) and MedalKind.

What follows is a listing of components that are to be replaced (From) by the transformation. In the running example these components are: won-gold-in, won-silver-in, and won-bronze-in. Similarly the To statement lists the components added by the transformation. In the example they are: MedalKind, code, char, won-medals-of-in, MedalKind.code and constraints $c1$ to $c4$.

The Update rules and Derivation rules provide the translation of populations between the schema before and after the transformation. We have specified these rules in a textual representation of a relational algebra like language. As mentioned before, in this article we are not concerned with a concrete language for these purposes. One statement in the update rules requiring some explanation though is the Val(y, i) statement. This function should generate a (unique) instance, for object type y, which is identified by the value i. Remember that instances of y are identified by instances of value type l. Value i is an instance of l, and Val(y, i) simply refers to the associated instance of y. It is not legal to just take Val$(y, i) = i$ since this would lead to a situation where the population of y and l overlap, which would be in contradiction with the fact that y and l are distinct object types that are not part of the same type hierarchy. Actually, as y is not a value type, Val(y, i) should lead to a value that is not directly representable on a communication medium; it should be an *abstract* instance. So Val should provide some global encoding schema of value

```
Transformation schema OTEmission ;
  Object types:
    Country, Quantity, MedalKind, code;
  Value types:
    code: char;
  Relationship types:
    won-gold-in = [Country:won-gold-in-1, Quantity:won-gold-in-2];
    won-silver-in = [Country:won-silver-in-1, Quantity:won-silver-in-2];
    won-bronze-in = [Country:won-bronze-in-1, Quantity:won-bronze-in-2];
    won-medals-of-in = [Country:won-medals-of-in-1, Quantity:won-medals-of-in-3, MedalKind:won-medals-of-in-2];
    MedalKind.code =
      [MedalKind:MedalKind.code-1,code:MedalKind.code-2];
  Constraints:
    c1: UNIQUE { MedalKind.code-1 };
    c2: UNIQUE { MedalKind.code-2 };
    c3: MANDATORY { MedalKind.code-2 };
    c4: EACH MedalKind IS IN ’G’, ’S’, ’B’;
  From: won-gold-in, won-bronze-in
  To: MedalKind, code, char, MedalKind.code, won-medals-of-in, c1, c2, c3, c4;
  Derivation rules:
    won-gold-in =
      PROJ[won-gold-in-1 = won-medals-of-in-1, won-gold-in-2 = won-medals-of-in-3]
        SEL[won-medals-of-in-2 = MedalKind.code-1, MedalKind.code-2 = ’G’]  won-medals-of-in JOIN MedalKind.code
    won-silver-in =
      PROJ[won-silver-in-1 = won-medals-of-in-1, won-silver-in-2 = won-medals-of-in-3]
        SEL[won-medals-of-in-2 = MedalKind.code-1, MedalKind.code-2 = ’S’)]  won-medals-of-in JOIN MedalKind.code
    won-bronze-in =
      PROJ[won-bronze-in-1 = won-medals-of-in-1, won-bronze-in-2 = won-medals-of-in-3]
        SEL[won-medals-of-in-2 = MedalKind.code-1. MedalKind.code-2 = ’B’)]  won-medals-of-in JOIN MedalKind.code
  Update rules:
    won-medals-of-in =
        PROJ[won-medals-of-in-1 = won-gold-in-1, won-medals-of-in-3 = won-gold-in-2,
                won-medals-of-in-2 = Val(MedalKind,’G’)] won-gold-in
      UNION
        PROJ[won-medals-of-in-1 = won-silver-in-1, won-medals-of-in-3 = won-silver-in-2,
                won-medals-of-in-2 = Val(MedalKind,’S’)] won-silver-in
      UNION
        PROJ[won-medals-of-in-1 = won-bronze-in-1, won-medals-of-in-3 = won-bronze-in-2,
                won-medals-of-in-2 = Val(MedalKind,’B’)] won-bronze-in
    MedalKind.code =
        { ⟨MedalKind.code-1 = Val(MedalKind,’G’), MedalKind.code-2 = ’G’⟩,
          ⟨MedalKind.code-1 = Val(MedalKind,’G’), MedalKind.code-2 = ’G’⟩,
          ⟨MedalKind.code-1 = Val(MedalKind,’G’), MedalKind.code-2 = ’G’⟩ }
End Transformation schema.
```

Table 2: The running example transformation

type instances into abstract non-value type instances. Finally, the completely substituted transformation is shown in table 6.1.

An observant reader might now note that the transformation scheme does not cater for the transformation of the constraints defined over the types involved in the transformation. In [Hal89], and [Hal95] transformation of constraints was covered by corollaries to the basic schema transformations. While useful, this approach leads to many corollaries to deal with different classes of constraints. Moreover, it provides no solution for constraints in general, and currently ignores most textual (non graphical) constraints that must be formulated in some formal textual language. The approach taken in this article, is to continue enforcing the (uniqueness) constraints on the transformed relationships on the (now) derived relationships. So in the Olympic games schema in schema (a), the won-gold-in relationship is derivable, while we enforce the uniqueness of its first role on this derived relationship. Using a constraint re-writing mechanism, constraints on derived object types can be re-written to constraints on the non derivable types. Such a re-writing mechanism, however, very much depends on the language used to express constraints and derivation rules. In the next section we briefly return to this issue.

6.2 Semantics of transformation schemes

Although we do not provide a formal semantics of the language used to specify the transformation schemes, we do presume the existence of three functions providing these semantics. When an actual language is defined these functions will become concrete. The (partial) functions are:

$$\begin{aligned} \mathsf{From}&\colon \mathtt{TransSchema} \times \mathtt{ParList} \rightarrowtail \mathcal{SCH} \\ \mathsf{To}&\colon \mathtt{TransSchema} \times \mathtt{ParList} \rightarrowtail \mathcal{SCH} \\ \mathsf{Schema}&\colon \mathtt{TransSchema} \times \mathtt{ParList} \rightarrowtail \mathcal{SCH} \end{aligned}$$

where $\mathtt{ParList} \subseteq ((\mathcal{RO} \cup \mathcal{TP})^*)^*$ such that in each $X \in \mathtt{ParList}$ any role or type occurs only once. These functions are partial since some combinations of transformation schemes and lists of parameters may define incorrect transformations.

The From function returns the schema components that will be changed by the transformation. What exactly is going to happen with these schema components depends on the aim with which the transformation is applied. In the next section these aims will be discussed in more detail, together with their different semantics. The result of the From statement is given as a (sub) schema. As this usually is a sub-schema without a proper context, this is not likely to be a complete and correct ORM schema. In our example the schema resulting from the From function only contains the three relationship types from the (b) variation of the olympic games schemas, without the Country and Quantity object types as such.

Similarly, the To function returns the added schema components in the form of a sub-schema. This returned schema is usually incomplete as well since it also misses the proper context. The To function not only returns the components listed by the To statement in the transformation scheme, but also the derivation and update rules. In the schema resulting from the transformation, these rules are required to derive the instances of the transformed types and translate the updates of the transformed types to updates of the new types.

Finally, the Schema function yields all schema components listed in the transformation scheme, and returns this as the schema. However, the update rules are ignored. The resulting schema provides the *context* of the schema transformation, and is used as a base for providing equivalence preservation proofs.

Whenever we use these functions we will apply the style of denotational semantics ([Sto77]). So we will for example write $\mathsf{Schema}[\![t]\!]\,(X) = \mathcal{SCH}_i$ to express the fact that $\mathcal{SCH}_i$ is the subschema that resuts when applying function Schema to transformation scheme t with parameter list X.

6.3 Inverse transformations

If t is an equivalence preserving schema transformation, then $\mathsf{Inv}(t)$ denotes the inverse transformation. In this case the lists provided by the To and From statements have to be swapped, as well as the derivation rules and update rules. So if t is a transformation, then $\mathsf{Schema}[\![\mathsf{Inv}(t)]\!]\,(X)$ has as derivation rules the update rules of t (with the proper parameters from X instantiated), and vice versa.

In general it only makes sense to invert an equivalence preserving transformation scheme. We will now take a closer look at the equivalence preservation properties which transformation schemes may possess.

6.4 Properties of transformation schemes

With respect to equivalence preservation, there are two transformation cases in which we are interested. A transformation can be equivalence preserving or it can strengthen an existing schema. When a schema is strengthened, the number of correct populations decreases, but this is sometimes seen as an acceptable trade-off to gain efficiency.

We generalise the above properties to transformation schemes. A transformation scheme is equivalence preserving if and only if all correct applications lead to equivalence preserving transformations. Formally:

$$\forall_{\langle T,X\rangle \in \mathsf{dom}(\mathsf{Schema})} \left[\mathsf{Schema}[\![T]\!]\,(X) \equiv \mathsf{Schema}[\![\mathsf{Inv}(T)]\!]\,(X)\right]$$

The example transformation scheme provided in table 6.1 is equivalence preserving.

A transformation scheme is strengthening if and only if all correct applications lead to strengthening transformations. Formally:

$$\forall_{\langle T,X\rangle \in \mathsf{dom}(\mathsf{Schema})} \left[\mathsf{Schema}[\![T]\!]\,(X) \preccurlyeq \mathsf{Schema}[\![\mathsf{Inv}(T)]\!]\,(X)\right]$$

These last two properties can be proven by generalising the proof of a concrete transformation. How a concrete transformation can be proven to be equivalence preserving or strengthening is discussed in section 2. For obvious reasons, the generalisation needs to be done to the parameters of the transformation scheme.

It is interesting to note that the derivation rules and update rules provided with a transformation scheme are now used as the *conservative extension* needed to prove the direct equivalence of the schema before and after the transformation.

6.5 Distributive updates

Finally, we put one more restriction on the derivation and update rules. This restriction has as a benefit that the view update problem is avoided.

When tranforming a conceptual schema $\mathcal{SCH}_i$ to a data schema $\mathcal{SCH}_j$, the user will still want to perform the updates as if they are done on the original conceptual schema. That is why we have added the update rules. These rules translate the updates of the conceptual types that have been replaced by other types to updates of these replacement types. Conversely, when querying the database, the user is not interested in the schema used for the actual storage of the data, but rather the original conceptual schema. In doing so, however, we may find ourselves exposed to the view update problem.

To allow the user to specify updates on the conceptual level such that they can be processed directly, we require that the update rules are *update distributive*. A set of update rules can be regarded as a function $\mu : \mathsf{POP} \rightarrow \mathsf{POP}$ that takes a population of the original schema and transforms that into a population of the actually stored data schema. The set POP contains all possible populations, so $\mathsf{POP} = \mathcal{OB} \rightarrow \wp(\Omega)$, where Ω is the set of possible instances of object types. The derivation rules perform the opposite function μ^{-1}, and for an equivalence preserving schema transformation this μ^{-1} is the inverse function of μ.

Before continuing we need the following generalisation of binary operations on sets. Let $p_1, p_2 \in \mathsf{POP}$, then we can generalise each binary operation Θ on sets (of instances) to populations as a whole by:

$$(p_1 \,\Theta\, p_2)(x) = p_1(x) \,\Theta\, p_2(x)$$

A population transforming function: $\mu : \mathsf{POP} \rightarrow \mathsf{POP}$. is called *update distributive* iff for $\Theta \in \{\cup, -\}$ and a correct schema $\mathcal{SCH}$ we have:

$$\mathsf{IsPop}(p, \mathcal{SCH}) \wedge \mathsf{IsPop}(p \,\Theta\, x, \mathcal{SCH}) \Rightarrow \mu(p \,\Theta\, x) = \mu(p) \,\Theta\, \mu(x)$$

This allows us to define a further restriction on the update rules specified in a transformation scheme:

> Let μ be the population transformation function following from the update rules from a given transformation scheme t, then that μ must be update distributive.

If t is equivalence preserving, then obviously the update rules of $\mathsf{Inv}(t)$ must define an update distributive function as well. From the definition of $\mathsf{Inv}(t)$ follows that this function must then necessarily be the inverse population transformation function following from the derivation rules in t.

With such a μ we can now safely translate any update of the population of the original schema to an update of the transformed schema. For a conceptual schema optimisation process this means that a user can continue specifying updates on the original conceptual schema.

Proving this property for a concrete transformation scheme is usually not hard. For the running example in this section it follows from the observation that each tuple added to (or deleted from) the population of a relationship f_i is turned into an addition (or deletion) of a tuple from the relationship h with an extra column containing the unique constant v_i. Conversely, when considering the inverse transformation, each update to the relation h is translated to an update of one of the relations f_i. Which one depends on the value of the u column of the tuples involved in the update.

Typical update rules that are now excluded due to this requirement are:

1. rules containing aggregations like: summation, maximums, etc.
2. rules containing encodings which need to consider the entire existing population to perform the encoding.

7 Transformation Steps

As stated before, in this article we are mainly interested in equivalence preserving or strengthening transformations. In the previous section we introduced a mechanism that enables us to define transformation schemes which can be applied to a concrete data schema. Other schema transformations, like the ones used in the conceptual schema design procedure, usually do not lend themselves to a representation as a general transformation scheme. The transformations in the conceptual schema design procedure have a much more ad-hoc character.

When focusing on the equivalence (or strengthening) transformation schemes, there are roughly three reasons to apply transformation schemes:

1. to select an alternative conceptual schema which is regarded as a better representation of the universe of discourse,
2. for the enrichment of the schema with derivable parts creating diverse alternative views on the same conceptual schema as a part of the original schema,
3. to optimise a finished conceptual schema before mapping it to a logical design,

The latter application is of course the main focus of this article. Nevertheless, we will also discuss the other applications of the schema transformations.

As an example, consider the transformation from figure 7 to figure 8. One (the user community in conjunction with the modeller) might consider the second schema to be a better conceptual representation of the universe of discourse. On the other hand, one might decide to let both alternatives co-exist together, this means that either figure 7 or figure 8 can be used as a base for the implementation, and that users can access the stored information in terms of the union of both schemas. Even more, once the conceptual schema is fixed, it may turn out that the alternative chosen for the conceptual representation is not the most optimal with respect to its implementation, in which case the other alternative needs to be used as a base for the actual implementation.

Corresponding to these three ways to apply a transformation scheme, we have three differing semantics of a transformation scheme. Before introducing these semantics, however, we first need to introduce a more elementary operation on schemas.

7.1 Cleaning up a schema

When transforming a given schema to a new schema, certain parts of the schema may become obsolete. This means that these parts can (must) be removed from the new schema. These removals may arise because:

1. object types may have become isolated,
2. derivation rules or update rules could be collapsed into simpler rules,
3. constraints may have become derivable,

We first focus on derivable types (together with their update and derivation rules) that are removable from a schema version $\mathcal{SCH}_i$. A derivable type can be removed if:

1. It has both an update rule and derivation rule that can be removed.

   Deleting a derivable type when the update or derivation rules cannot be removed would lead to a breach of the derivability of changed conceptual types.

2. It is marked as an internal type.

   We do not want to remove conceptual types.

3. When another type is dependent on this type.

   When no other type depends on the type for consideration of removal, then the removal of this type would lead to a collapse of a part of the information structure.

Note that 2 automatically follows from the fact that update rules can only be defined for internal types. These observations lead to the following formal definition of the derivable types that can be removed (R_i):

$$\begin{array}{lcll} D_i & \triangleq & \big\{\mathsf{Defines}(r) \bigm| r \in \mathcal{DR}_i \land \mathsf{Defines}(r) \notin \mathsf{Depends}(r)\big\} & \text{removable derivable types} \\ U_i & \triangleq & \big\{\mathsf{Defines}(r) \bigm| r \in \mathcal{UR}_i \land \mathsf{Defines}(r) \notin \mathsf{Depends}(r)\big\} & \text{removable internal types} \\ P_i & \triangleq & \big\{x \bigm| \exists_{y \in \mathcal{TP}_i}\left[x \,\mathsf{SubOf}\, y\right]\big\} & \text{types which have dependent types} \\ R_i & \triangleq & (D_i \cap U_i) - P_i & \text{removable derivable types} \end{array}$$

The second class of types that can be removed is the set of types that are isolated, i.e. not connected to any other type. This set is identified as:

$$UC_i \quad \triangleq \quad \big\{x \in \mathcal{TP}_i \bigm| \neg\exists_{p \in \mathcal{RO}_i}\left[\mathsf{Player}(p) = x\right]\big\} \cup \big\{x \in \mathcal{TP}_i \bigm| \neg\exists_{y \in \mathcal{TP}_i}\left[y \,\mathsf{SubOf}\, x\right]\big\}$$

Now we know which types can be removed ($R_i \cup UC_i$) in a schema version $\mathcal{SCH}_i$, we can define the sets of removable derivation rules and update rules:

$$\begin{aligned} RD_i &\triangleq \left\{ r \in \mathcal{DR}_i \;\middle|\; \mathsf{Defines}(r) \in R_i \cup UC_i \right\} \\ RU_i &\triangleq \left\{ r \in \mathcal{UR}_i \;\middle|\; \mathsf{Defines}(r) \in R_i \cup UC_i \right\} \end{aligned}$$

The actual CleanUp operation is defined recursively. For a given schema version $\mathcal{SCH}_i$, the one-step cleanup operation $\mathsf{CleanUp}^1(\mathcal{SCH}_i)$ leads to a new schema $\mathcal{SCH}_j$ where:

1. $\mathcal{TP}_j = \mathcal{TP}_i - UC_i - R_i$
2. $\mathcal{IT}_j = \mathcal{IT}_i \cap \mathcal{TP}_j$
3. $\mathcal{DR}_j = (\mathcal{DR}_i - RD_i)|^{RD_i}$ where $X|^{RD_i}$ replaces all references to rules in *RD* by the body of these rules (i.e. substituting the definition of the removed derivation rules in the remaining rules).
4. $\mathcal{UR}_j = (\mathcal{UR}_i - RU_i)|^{RU_i \cup RD_i}$

   Note that in this case also the derivation rules that are removed need to be substituted as the update rules may refer to derivable types that have just been removed.
5. $\mathcal{CN}_j = \mathsf{Reduce}(\mathcal{SCH}_i, \mathcal{CN}_i|^{RD_i})$ where Reduce tries to remove derivable constraints and rewrite the constraints to constraints on non-derivable types. Below we elaborate more on this function.
6. $\mathsf{Dom}_j = \left\{ \langle v, d \rangle \in \mathsf{Dom}_i \;\middle|\; v \in \mathcal{VL}_j \right\}$

This defines the one-step clean-up operation. After one clean-up step additional types may have become isolated. Therefore, the clean-up operation needs to be applied recursively. For $n > 1$ we therefore have:

$$\mathsf{CleanUp}^n(\mathcal{SCH}_i) = \mathsf{CleanUp}^1(\mathsf{CleanUp}^{n-1}(\mathcal{SCH}_i))$$

From this the general cleaning up function can be defined by:

$$\mathsf{CleanUp}(\mathcal{SCH}_i) = \mathsf{CleanUp}^n(\mathcal{SCH}_i) \text{ where } n \text{ is such that: } \mathsf{CleanUp}^n(\mathcal{SCH}_i) = \mathsf{CleanUp}^{n+1}(\mathcal{SCH}_i)$$

This leaves the definition of the Reduce function. The Reduce function is needed for efficiency reasons, since it is much more efficient to enforce constraints on base types (and eventually on tables) then on derived types. Furthermore, algorithms that map data schemas to internal schemas (e.g. a relational schema) typically make mapping decisions based on the constraints that hold on the base types. However, the exact definition of such a function depends very much on the language chosen for the specificiation of constraints and derivation rules.

For example in the transformation given in figure 9, one would like to have the system derive the uniqueness constraints on the three binary fact types in the resulting schema from the single two-role uniqueness constraint in the original schema. Given a formal language for constraint specification and derivation rule specification (for example a relation algebra), one could specify a set of re-write rules for such constraint and derivation rule specifications.

7.2 Schema alternatives

The first semantic interpretation of a transformation scheme we consider leads to schema alternatives. These semantics are provided by the function:

$$\mathsf{Alternative} : \mathtt{TransSchema} \times \mathtt{ParList} \times \mathcal{SCH} \rightarrowtail \mathcal{SCH}$$

which is defined by

$$\mathsf{Alternative}[\![T]\!]\,(X, \mathcal{SCH}_i) \triangleq \mathsf{CleanUp}(\mathcal{SCH}_j)$$

where (using $\mathcal{SCH}_f = \mathsf{From}[\![T]\!]\,(X)$ and $\mathcal{SCH}_t = \mathsf{To}[\![T]\!]\,(X)$) the components of $\mathcal{SCH}_j$ are defined as:

1. $\mathcal{TP}_j = \mathcal{TP}_i \cup \mathcal{TP}_t$.
2. $\mathcal{IT}_j = \mathcal{IT}_i \cup \mathcal{TP}_f$.

   We cannot simply remove the changed types ($\mathcal{TP}_f$) since some of them may be used in constraints for the derivation or construction of other types. Therefore they (initially) need to remain present in the schema, unless a closer study reveals that a type is indeed not needed ($\mathsf{CleanUp}$).
3. $\mathcal{DR}_j = \mathcal{DR}_i \cup \mathcal{DR}_t$.
4. $\mathcal{UR}_j = \mathcal{UR}_i \cup \mathcal{UR}_t$.
5. $\mathcal{CN}_j = \mathcal{CN}_i \cup \mathcal{CN}_t$
6. $\mathsf{Dom}_j = \mathsf{Dom}_i \cup \mathsf{Dom}_t$.

The schema transformation must, however, obey certain rules. If these rules are violated, the schema transformation is considered undefined. The rules are:

1. The correctness of schemata is preserved by the transformation:
$$\mathsf{IsSch}(\mathcal{SCH}_i) \Rightarrow \mathsf{IsSch}(\mathsf{CleanUp}(\mathcal{SCH}_j))$$
2. The types changed by the transformation are present in the original schema as non-derivable types: $\mathcal{TP}_f \subseteq \mathcal{TP}_i - \left\{ \mathsf{Defines}(r) \;\middle|\; r \in \mathcal{DR}_i \right\}$.
3. The new types should not already be present in the current data schema: $\mathcal{TP}_t \cap \mathcal{TP}_i = \varnothing$.

7.3 Schema enrichment

The second semantic interpretation is the enrichment of an existing data schema. No types are removed, but the set of known types is enriched by new types defined in terms of the existing ones. The semantics are provided by the function

$$\mathsf{Enrich} : \mathtt{TransSchema} \times \mathtt{ParList} \times \mathcal{SCH} \rightarrowtail \mathcal{SCH}$$

which is defined by

$$\mathsf{Enrich}[\![T]\!]\,(X, \mathcal{SCH}_i) \triangleq \mathsf{CleanUp}(\mathcal{SCH}_j)$$

where $\mathcal{SCH}_j$ is defined the same as for $\mathsf{Alternative}$ except for the internal types:

2 The internal types of the schema remain unchanged: $\mathcal{IT}_j = \mathcal{IT}_i$.

The same extra conditions as defined on $\mathsf{Alternative}$ apply for Enrich as well.

7.4 Schema optimisation

The third interpretation defines a transformation leading to a data schema optimised for internal representation: The semantics are provided by the function

$$\mathsf{Optimise} : \mathtt{TransSchema} \times \mathtt{ParList} \times \mathcal{SCH} \rightarrowtail \mathcal{SCH}$$

which is defined by

$$\mathsf{Optimise}[\![T]\!]\,(X, \mathcal{SCH}_i) \triangleq \mathsf{CleanUp}(\mathcal{SCH}_j)$$

where $\mathcal{SCH}_j$ is defined the same as for $\mathsf{Alternative}$ except, again, for the internal types:

2 The internal types of the new schema are: $\mathcal{IT}_j = \mathcal{IT}_i \cup (\mathcal{TP}_t - \mathcal{TP}_i)$.

All types added are for internal purposes only.

The same extra conditions as defined on $\mathsf{Alternative}$ apply for $\mathsf{Optimise}$ as well.

8 Conclusions and Further Research

In this article we have presented an approach to automated conceptual schema transformations. We have sketched a broader context for these transformations, and have focussed the attention on transformations improving the conceptual quality of schemas as well as transformations leading to more efficient implementations.

Given a concrete language for constraint and derivation rule specifications, a re-write system must now be developed for this language that allows constraints to be re-written (as much as possible) in terms of base types. It is planned to implement the ideas presented in this article in the context of a project aiming for the development of a generic conceptual (data) modelling CASE Tool ([CP96]). Furthermore, the pool of strengthening and equivalence preserving schema transformation schemes needs to be extended.

Heuristics and algorithms to drive a schema optimisation process have been developed [Hal95], but these need to be extended to cover more cases (e.g. use of complex types). One cannot expect a user to manually select both the schema components and a transformation scheme to be used in a schema transformation step. A tool for performing schema optimisation in a semi-automated way is currently being designed.

References

[BBMP95] G.H.W.M. Bronts, S.J. Brouwer, C.L.J. Martens, and H.A. Proper. A Unifying Object Role Modelling Approach. *Information Systems*, 20(3):213–235, 1995.

[BBP$^+$79] F.L. Bauer, M. Broy, H. Partsch, P. Pepper, and H. Wössner. Systematics of transformation rules. In F.L. Bauer and M. Broy, editors, *Program construction*, volume 69 of *Lecture Notes in Computer Science*, pages 273–289, Berlin, Germany, 1979. Springer-Verlag.

[BCN92] C. Batini, S. Ceri, and S.B. Navathe. *Conceptual Database Design - An Entity-Relationship Approach*. Benjamin Cummings, Redwood City, California, 1992.

[Ber86] S. Berman. A semantic data model as the basis for an automated database design tool. *Information Systems*, 11(2):149–165, 1986.

[Blo93] A. Bloesch. *Signed Tableaux – a Basis for Automated Theorem Proving in Nonclassical Logics*. PhD thesis, University of Queensland, Brisbane, Australia, 1993.

[BMPP89] F.L. Bauer, B. Möller, H. Partsch, and P. Pepper. Formal Program Construction by Transformations – Computer-Aided, Intuition-Guided Programming. *IEEE Transactions on Software Engineering*, 15(2):165–180, February 1989.

[Bom94] P. van Bommel. Implementation Selection for Object-Role Models. In T.A. Halpin and R. Meersman, editors, *Proceedings of the First International Conference on Object-Role Modelling (ORM-1)*, pages 103–112, Magnetic Island, Australia, July 1994.

[BW92] P. van Bommel and Th.P. van der Weide. Reducing the search space for conceptual schema transformation. *Data & Knowledge Engineering*, 8:269–292, 1992.

[Cam94] L.J. Campbell. Adding a New Dimension to Flat Conceptual Modelling. In T.A. Halpin and R. Meersman, editors, *Proceedings of the First International Conference on Object-Role Modelling (ORM-1)*, pages 294–309, Magnetic Island, Australia, July 1994.

[CBS94] R. Chiang, T. Barron, and V. Storey. Reverse engineering of relational databases: Extraction of an eer model from a relational database. *Data & Knowledge Engineering*, 12(2):107–142, 1994.

[CH94] L.J. Campbell and T.A. Halpin. Abstraction Techniques for Conceptual Schemas. In R. Sacks-Davis, editor, *Proceedings of the 5th Australasian Database Conference*, volume 16, pages 374–388, Christchurch, New Zealand, January 1994. Global Publications Services.

[CHP96] L.J. Campbell, T.A. Halpin, and H.A. Proper. Conceptual Schemas with Abstractions – Making flat conceptual schemas more comprehensible. *Data & Knowledge Engineering*, 20(1):39–85, 1996.

[CK77] C.C. Chang and H.J. Keisler. *Model Theory*. North-Holland, Amsterdam, The Netherlands, 2nd edition, 1977.

[CP96] P.N. Creasy and H.A. Proper. A Generic Model for 3-Dimensional Conceptual Modelling. *Data & Knowledge Engineering*, 20(2):119–162, 1996.

[CY90] P. Coad and E. Yourdon. *Object-Oriented Analysis*. Yourdon Press, New York, New York, 1990.

[De 93] O.M.F. De Troyer. *On Data Schema Transformations*. PhD thesis, University of Tilburg (K.U.B.), Tilburg, The Netherlands, 1993.

[EN94] R. Elmasri and S.B. Navathe. *Fundamentals of Database Systems*. Benjamin Cummings, Redwood City, California, 1994. Second Edition.

[EWH85] R. Elmasri, J. Weeldreyer, and A. Hevner. The category concept: An extension to the entity-relationship model. *Data & Knowledge Engineering*, 1:75–116, 1985.

[FG92] M.M. Fonkam and W.A. Gray. An Approach to Eliciting the Semantic of Relational Databases. In P. Loucopoulos, editor, *Proceedings of the Fourth International Conference CAiSE'92 on Advanced Information Systems Engineering*, volume 593 of *Lecture Notes in Computer Science*, pages 463–480, Manchester, United Kingdom, 1992. Springer-Verlag.

[Hal89] T.A. Halpin. *A logical analysis of information systems: static aspects of the data-oriented perspective*. PhD thesis, University of Queensland, Brisbane, Australia, 1989.

[Hal90] T.A. Halpin. Conceptual schema optimization. *Australian Computer Science Communications*, 12(1):136–145, 1990.

[Hal91] T.A. Halpin. A Fact-Oriented Approach to Schema Transformation. In B. Thalheim, J. Demetrovics, and H.-D. Gerhardt, editors, *MFDBS 91*, volume 495 of *Lecture Notes in Computer Science*, pages 342–356, Rostock, Germany, 1991. Springer-Verlag.

[Hal92a] T.A. Halpin. Fact-oriented schema optimization. In A.K. Majumdar and N. Prakash, editors, *Proceedings of the International Conference on Information Systems and Management of Data (CISMOD 92)*, pages 288–302, Bangalore, India, July 1992.

[Hal92b] T.A. Halpin. WISE: a Workbench for Information System Engineering. In V.-P. Tahvanainen and K. Lyytinen, editors, *Next Generation CASE Tools*, volume 3 of *Studies in Computer and Communication Systems*, pages 38–49. IOS Press, 1992.

[Hal95] T.A. Halpin. *Conceptual Schema and Relational Database Design*. Prentice-Hall, Sydney, Australia, 2nd edition, 1995.

[HP95] T.A. Halpin and H.A. Proper. Subtyping and Polymorphism in Object-Role Modelling. *Data & Knowledge Engineering*, 15:251–281, 1995.

[HPW92a] A.H.M. ter Hofstede, H.A. Proper, and Th.P. van der Weide. A Note on Schema Equivalence. Technical Report 92-30, Department of Information Systems, University of Nijmegen, Nijmegen, The Netherlands, EU, 1992.

[HPW92b] A.H.M. ter Hofstede, H.A. Proper, and Th.P. van der Weide. Data Modelling in Complex Application Domains. In P. Loucopoulos, editor, *Proceedings of the Fourth International Conference CAiSE'92 on Advanced Information Systems Engineering*, volume 593 of *Lecture Notes in Computer Science*, pages 364–377, Manchester, United Kingdom, EU, May 1992. Springer Verlag, Berlin, Germany, EU. ISBN 3540554815

[HPW93] A.H.M. ter Hofstede, H.A. Proper, and Th.P. van der Weide. Formal definition of a conceptual language for the description and manipulation of information models. *Information Systems*, 18(7):489–523, October 1993.

[HPW97] A.H.M. ter Hofstede, H.A. Proper, and Th.P. van der Weide. Exploiting Fact Verbalisation in Conceptual Information Modelling. *Information Systems*, 22(6/7):349–385, September 1997.

[HW93] A.H.M. ter Hofstede and Th.P. van der Weide. Expressiveness in conceptual data modelling. *Data & Knowledge Engineering*, 10(1):65–100, February 1993.

[HW94] A.H.M. ter Hofstede and Th.P. van der Weide. Fact Orientation in Complex Object Role Modelling Techniques. In T.A. Halpin and R. Meersman, editors, *Proceedings of the First International Conference on Object-Role Modelling (ORM-1)*, pages 45–59, Townsville, Australia, July 1994.

[HW97] A.H.M. ter Hofstede and Th.P. van der Weide. Deriving Identity from Extensionality. *International Journal of Software Engineering and Knowledge Engineering*, 8(2):189–221, June 1997.

[Kal91] K. Kalman. Implementation and critique of an algorithm which maps a relational database to a conceptual model. In R. Andersen, J.A. Bubenko, and A. Sølvberg, editors, *Proceedings of the Third International Conference CAiSE'91 on Advanced Information Systems Engineering*, volume 498 of *Lecture Notes in Computer Science*, pages 393–415, Trondheim, Norway, May 1991. Springer-Verlag.

[Kob86a] I. Kobayashi. Classification and transformations of binary relationship relation schemata. *Information Systems*, 11(2):109–122, 1986.

[Kob86b] I. Kobayashi. Losslessness and semantic correctness of database schema transformation: another look of schema equivalence. *Information Systems*, 11(1):41–59, 1986.

[Kri94] G. Kristen. *Object Orientation – The KISS Method, From Information Architecture to Information System*. Addison-Wesley, Reading, Massachusetts, USA, 1994. ISBN 0201422999

[LN88] C.M.R. Leung and G.M. Nijssen. Relational database design using the NIAM conceptual schema. *Information Systems*, 13(2):219–227, 1988.

[Mee82] R. Meersman. The RIDL Conceptual Language. Research report, International Centre for Information Analysis Services, Control Data Belgium, Inc., Brussels, Belgium, 1982.

[MHR93] J.I. McCormack, T.A. Halpin, and P.R. Ritson. Automated mapping of conceptual schemas to relational schemas. In C. Rolland, F. Bodart, and C. Cauvet, editors, *Proceedings of the Fifth International Conference CAiSE'93 on Advanced Information Systems Engineering*, volume 685 of *Lecture Notes in Computer Science*, pages 432–448, Paris, France, 1993. Springer-Verlag.

[Nij89] G.M. Nijssen. An Axiom and Architecture for Information Systems. In E. D. Falkenberg and P. Lindgreen, editors, *Information System Concepts: An In-depth Analysis*, pages 157–175. North-Holland/IFIP, Amsterdam, The Netherlands, 1989.

[Pro94] H.A. Proper. *A Theory for Conceptual Modelling of Evolving Application Domains*. PhD thesis, University of Nijmegen, Nijmegen, The Netherlands, EU, 1994. ISBN 909006849X

[PS83] H. Partsch and R. Steinbrüggen. Program Transformation Systems. *Computing Surveys*, 15(3), 1983.

[PW94] H.A. Proper and Th.P. van der Weide. EVORM - A Conceptual Modelling Technique for Evolving Application Domains. *Data & Knowledge Engineering*, 12:313–359, 1994.

[PW95a] H.A. Proper and Th.P. van der Weide. A General Theory for the Evolution of Application Models. *IEEE Transactions on Knowledge and Data Engineering*, 7(6):984–996, December 1995.

[PW95b] H.A. Proper and Th.P. van der Weide. Information Disclosure in Evolving Information Systems: Taking a shot at a moving target. *Data & Knowledge Engineering*, 15:135–168, 1995.

[RBP$^+$91] J. Rumbaugh, M. Blaha, W. Premerlani, F. Eddy, and W. Lorenson. *Object-Oriented Modeling and Design*. Prentice-Hall, Englewood Cliffs, New Jersey, 1991.

[Ris93] N. Rishe. A methodology and tool for top-down relational database design. *Data & Knowledge Engineering*, 10:259–291, 1993.

[Rit94] P.R. Ritson. *Use of Conceptual Schemas for a Relational Implementation*. PhD thesis, University of Queensland, Brisbane, Australia, 1994.

[RP92] J.F. Roddick and J.D. Patrick. Temporal semantics in information systems - A survey. *Information Systems*, 17(3):249–267, 1992.

[SEC87] P. Shoval and M. Even-Chaime. ADDS: A system for automatic database schema design based on the binary-relationship model. *Data & Knowledge Engineering*, 2(2):123–144, 1987.

[SS93] P. Shoval and N. Shreiber. Database reverse engineering: From the Relational to the Binary Relationship model. *Data & Knowledge Engineering*, 10:293–315, 1993.

[Sto77] J.E. Stoy. *Denotational Semantics: The Scott-Strachey Approach to Programming Language Semantics*. MIT Press, Cambridge, Massachusetts, 1977.

[Win90] J.J.V.R. Wintraecken. *The NIAM Information Analysis Method: Theory and Practice*. Kluwer, Deventer, The Netherlands, EU, 1990.