Data Schema Design as a Schema Evolution Process

H.A. Proper
Cooperative Information Systems Research Centre
Faculty of Information Technology
Queensland University of Technology
GPO Box 2434, Brisbane, 4001 Australia
E.Proper@acm.org

Version of March 9, 1998 at 17:47

Published as: *Data Schema Design as a Schema Evolution Process*, H.A. Proper, Data & Knowledge Engineering, 22(2), 159–189, 1997

Abstract

In an information system a key role is played by the underlying data schema. This article starts out from the view that the entire modelling process of an information system's data schema can be seen as a schema transformation process. A transformation process that starts out with an initial draft conceptual schema and ends with an internal database schema for some implementation platform. This allows us to describe the transformation process of a database design as an evolution of a schema through a universe of data schemas. Doing so, allows for a better understanding of the actual design process, countering the problem of 'software development under the lamppost'. Even when the information system design is finalised, the data schema can evolve further due to changes in the requirements on the system.

We present a universe of data schemas that allows us to describe the underlying data schemas at all stages of their development. This universe of data schemas is used as a case study on how to describe the complete evolution of a data schema with all its relevant aspects. The theory is general enough to cater for more modelling concepts, or different modelling approaches. To actually model the evolution of a data schema, we present a versioning mechanism that allows us to model the evolutions of the elements of data schemas and their interactions, leading to a better understanding of a schema design process as a whole. Finally, we also discuss the relationship between this simple versioning mechanism and general purpose version management systems.

1 Introduction

When designing an information system, one usually starts out by specifying a so called *conceptual schema*. After such a conceptual schema is finalised, this schema is then implemented on a target platform such as a relational system, a hierarchical system, or an object-oriented system. To this end, the conceptual schema is transformed (mapped) to a schema on the chosen target platform. In this article we take the view that the entire (data) modelling process can be seen as a transformation process of data schemas, where a data schema can be an Entity Relationship schema ([EN94, BCN92]), an Object-Role Modelling schema ([Hal95]), a relational schema, or any other internal representation schema. This view is in line with transformational approaches to software engineering in general ([BBP+79, PS83, BMPP89]), which take a high level software specification and convert this by a sequence of transformations to concrete code. Even more, after the data schema of an information system has been implemented, the requirements on the system may have changed, requiring the information system to evolve as well. This means that the underlying data schema has to undergo yet another set of transformations to incorporate the new requirements. This observation has led to the development of so-called *evolving information systems* (see e.g. [BKKK87, TS92, PW95, OPF94, PW94, HEH+94]).

1.1 Schema Design Through Transformations

To further motivate our view on a schema modelling process as a transformation process, we first take a closer look at the involved transformations. A conceptual schema design process usually starts out with an initial draft conceptual schema which is then subject to a process of refinement and quality improvements, resulting in the final conceptual schema of the universe of discourse. This modelling process is usually guided by some design procedure. For example, in ([Hal95]) a design procedure for Object-Role Modelling techniques is discussed. In [RBP+91, Kri94] design procedures for object-oriented techniques are provided, while [EN94, BCN92] discuss some loose guidelines for ER schema design.

After the conceptual schema has been finalised, one can sometimes perform small (equivalence preserving) transformations on these schemas which result in a schema that allows for a more efficient implementation ([Hal90, Hai91, Hal91, EN94, HP95, PH95]). These transformations typically utilise the rich semantics and clarity of conceptual schemas. Performing such transformations after these schemas have already been mapped to a target platform usually becomes too complicatedi, since these schemas use less concise modelling concepts. This makes it both harder to define the transformations, and harder for the information system designers to track the transformations.

Following the optimisation transformations, the schema is consequently mapped to a target platform. For the different conceptual modelling techniques there are different mapping algorithms following varying styles and strategies. See for instance: [Ber86, SEC87, BW92, Ris93, MHR93, Hal95, Bom94, Rit94]. Once a conceptual schema has been transformed to some sort of representation for a target platform, this schema can sometimes be optimised even further. Some of these transformations are discussed in e.g. [De 93, Kob86, BCN92].

So-far we have distinguished 5 key classes of schema transformations. They are:

1. Conceptual schema refinements
2. Conceptual schema quality improvements
3. (Conceptual) schema optimisations
4. Conceptual to internal (logical) schema mapping
5. Internal schema optimisations

For a more elaborate discussion on the classes of schema transformations, refer to e.g. [Hai91]. Presently, the reverse process of the transformations in 4 and 5 also receives a lot of attention in the database and information systems research community. This reversed process is referred to as *reverse engineering*. For this fairly new area also a wide range of strategies and algorithms exists ([Kal91, FG92, SS93, HCTJ93, HTJC93b, HTJC93a, CBS94]). These algorithms all operate on the base of a set of possible schema transformations and heuristics to best apply them. The above discussed five classes either operate on a conceptual data schema or an internal schema of a given implementation platform. The modelling process up until the start of the internal schema mapping can be regarded as a journey through a universe of data schemas. This is illustrated in figure 1. This universe of data schemas must be rich enough such that the data schemas can include both details relevant for a conceptual presentation as well as the internal representation.

The aim of this article is to develop a mechanism by which we can describe the evolution of a data schema undergoing the above discussed schema transformations. Such a mechanism can then be used in the context of a CASE-Tool or an evolving information system. This evolution describing mechanism consists of two key components. The first describing the state space of the universe of data schemas, and the second the historical aspects. In this article, focus is on the latter aspects. Nevertheless, we will have to introduce a state space that is rich enough for our requirements.

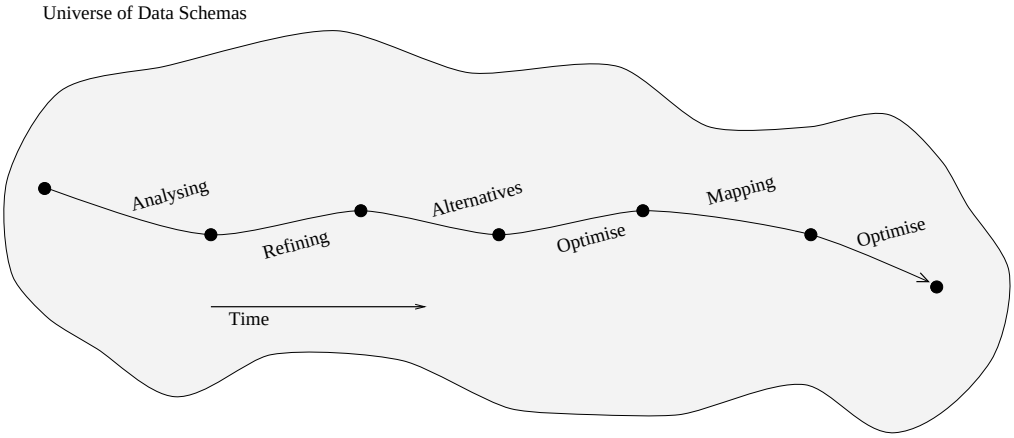


Figure 1: Evolution of a data schema in time

1.2 Overview of the Version Management System

The state space is build from existing, well published, components ([BW92, HW93, BBMP95, CP96]). Therefore, it is only discussed briefly in section 2. What makes the universe of data schemas unique is that it is rich enough to describe (at multiple levels of abstraction) ER and ORM data models *simultaneously* on a conceptual and on an internal level. This is done by combining the CDM (Conceptual Data Modelling) Kernel ([CP96]) with a specification technique for internal representations ([BW92, Bom95]). This latter techniques extends flat data schemas with a special notation that allows us to group roles together in trees. These trees than directly correspond to internal representations. Below we will see an example of this tree representation.

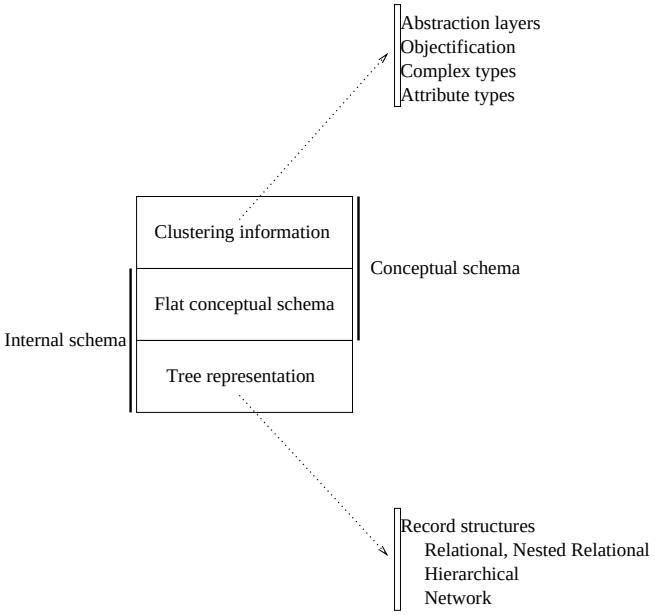


Figure 2: A complete data schema

This combination of the CDM Kernel and the tree representations, leads to the view shown in figure 2. The actual conceptual schema is provided by the flat data schema and the clustering information, which provides the distinction between attribute types, entity types, complex types, and the layers of abstraction. The internal schema is provided by the flat data schema in conjunction with a tree representation of the roles

in the conceptual schema, where each tree corresponds to one record structure of the internal schema. For convenience, the resulting data modelling technique is still referred to as the CDM Kernel in the remainder. This article can therefore actually be seen as the third and final part in a trilogy on the CDM Kernel. The first article ([CP96]) defining the kernel in full detail, the second one defining an algorithm for bottom up abstraction of flat conceptual models ([CHP96]) in the kernel, and finally this article which adds a version management system.

The second component of the versioning mechanism, introduced in figure 3, provides us with a way to model the evolution of the schema design. In our approach we are able to actually model any interactions schema components may have with each other during a schema design process. An example of this is shown in figure 3, where schema elements $e1$ and $e2$ fuse by reaction R to become $e3$. For instance two relationship types that merge. This allows modellers to trace the evolution of schema components through a modelling or evolution process. This should provide the modeller (and even the end user) with more insight into the evolution of the schema design.

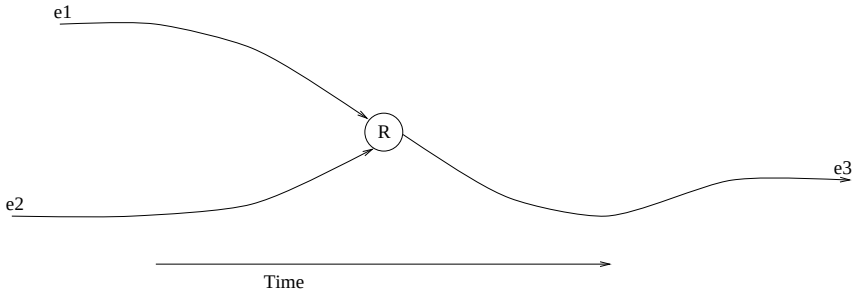


Figure 3: A reaction between schema elements

This latter issue should not be underestimated. Already in [CKSI87], it was stated that most existing software process models, including the (then) new approaches like prototyping and program transformation, "*focus on the series of artifacts that exist at the end of the phases of the process, rather than on the actual processes that are conducted to creat the artifacts*". Curtis et al, compared this phenomenon to the old story of the man who lost his wallect across the street, but searches it on this side under the lamppost, simply because the light is better. We expect that by providing better insight in the interactions between model elements during a schema design process, the light will also reach the other side of the street.

In [CKSI87, CKI88], it was found that more insight into the actual modelling process, e.g. decission points, was crucial for a better understanding of the project as a whole. We believe that our version management system provides a start for this in the context of schema design. By being able to explicitly model interactions of schema elements in the course of a schema design, and adding appropriate explanations to each of these interactions and design steps, a better understanding of the design process should result.

The integration between the schema universe and the version management components is provided in section 4. This yields a complete specification for an evolution mechanism that allows us to describe the evolution of a data schema through all its stages of design, including that of evolution of the information system as a whole due to changed requirements. The resulting evolution mechanism is referred to as the CDM Evolver.

Existing approaches to the modelling of data schema evolution (e.g. [BKKK87, TS92, PW94]) are only able to describe the evolution of either the conceptual level, or the internal level, but not both as a unity. For example, [BKKK87, TS92] describe the evolution at the actual database level and not the conceptual level. EVORM ([PW94]), on the other hand, only allows us to describe the evolution of a conceptual schema. In this article, we re-apply some of the principles used in the development of EVORM, which was derived from the Object-Role Modelling (ORM) variation PSM ([HW93]), while extending it to meet our new requirement to include abstraction layers and internal mapping aspects. Furthermore, the historical dimension of EVORM is further refined in the sense that we are, as stated before, able to explicitly model interactions between data schema elements. For example, the merger of two relationship types into one.

2 Data Schema Universe

The data schema universe is the set of valid data schemas. This set of data schemas is defined by the data schema language. As stated before, this language must be rich enough to allow us to model both the conceptual aspects of a data schema as well as internal representation aspects in one single model. In this section we therefore extend the CDM Kernel with an existing technique ([BW92]) to cater for this. The extension allows us to group roles into trees. Each tree corresponds directly to a record structure, be it conform the (flat) relational model, the nested relational model, the network model, or the hierarchical model. This technique was developed as a means to study alternative internal representations of the Predicator Model ([BHW91]). The Predicator Model is one of the first formalisations of Object-Role Modelling, and it was one of the ancestors of the CDM Kernel.

As the CDM Kernel is the result of a series of extensions and refinements of formalisations of ORM, each of which have been published before ([BHW91, HW93, PW94, BBMP95, CP96]), we allow ourselves the luxury of only briefly discussing the current CDM Kernel. We start with a concise description of flat conceptual schemas in subsection 2.1, followed in subsection 2.2 by the abstraction layers that can be built on top of that. In subsection 2.3, the tree representation of flat conceptual schemas is discussed. These trees are used to determine the internal representation of the conceptual schema. We also discuss briefly how these trees can be interpreted for different implementation platforms.

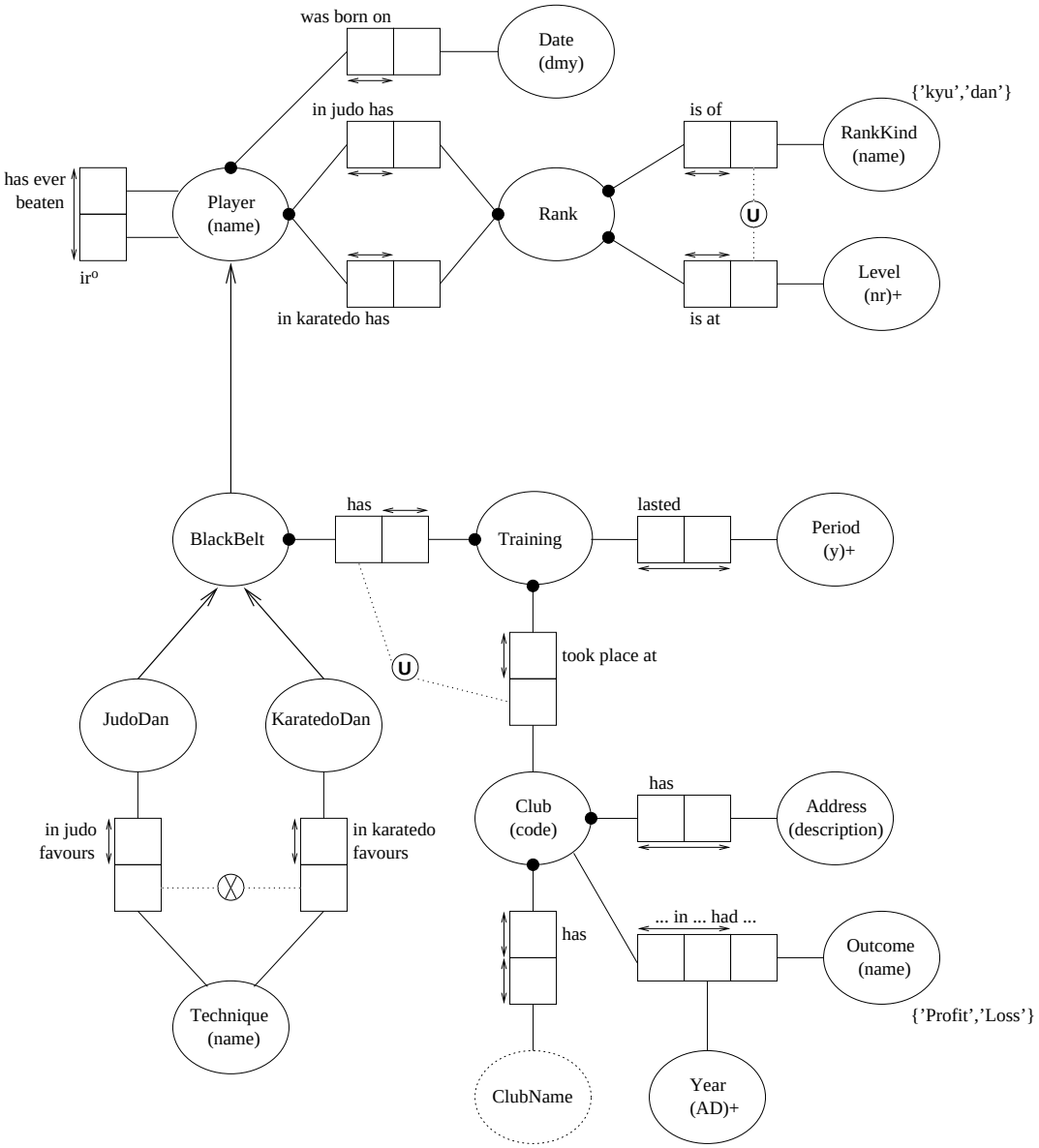
2.1 Flat Conceptual Schemas

In figure 4 an example, taken from [Hal95], of a flat conceptual schema is shown. The domain is concerned with martial arts, and the schema should be self explanatory. The notation used is the style of Object-Role Modelling (ORM). For a flat conceptual schema we cannot directly use the ER notation, as no information is available to distinguish between entity types and attribute types. For more details on this refer to [CP96].

Formally, a flat conceptual schema version at point in time t consists of the following components:

1. A set $\mathcal{OB}_t$ of object types.
2. A subset $\mathcal{VL}_t \subseteq \mathcal{OB}_t$ of value types. Value types are types that have instances which are directly denotable on some communication medium. For example, numerals, strings, sound, video and HTML. The function $\mathsf{Dom}_t : \mathcal{VL}_t \to \mathcal{DO}_t$ is used to associate a domain of values to each value type.
3. A set $\mathcal{RL}_t$ of relationship types. The roles of the relationship types are provided as $\mathcal{RO}_t$, while the function $\mathsf{Roles}_t : \mathcal{RL}_t \to \wp^+(\mathcal{RO}_t)$ partitions the roles of the relationship types. The players of the roles are given by the function $\mathsf{Player}_t : \mathcal{RO}_t \to \mathcal{OB}_t$. The set of all types is defined as $\mathcal{TP}_t \triangleq \mathcal{OB}_t \cup \mathcal{RL}_t$.
4. The subtyping hierarchy is captured by the predicate $\mathsf{SubOf}_t \subseteq \mathcal{OB}_t \times \mathcal{OB}_t$.
5. The identification schemes of object types are given by the function $\mathsf{Ident}_t : \mathcal{OB}_t \to (\wp(\mathcal{RO}_t \times \mathcal{RO}_t))^+$. The CDM Kernel allows identification schemes to be inherited from supertypes onto subtypes, but subtypes may also provide their own identification scheme.
6. A set $\mathcal{CN}_t$ of constraints.
7. Some types in the conceptual schema may be derivable (most notably subtypes). Therefore, $\mathsf{DerRule}_t : \mathcal{TP}_t \rightarrowtail \texttt{DerivationRules}$ may associate a derivation rule to types. Furthermore, for each derivable type an update rule may be specified defining how to deal with updates on the derived types: $\mathsf{UpdRule}_t : \mathcal{TP}_t \rightarrowtail \texttt{UpdateRules}$.

In this article we do not elaborate on correctness rules for single schema versions. For a detailed discussion on these rules see [CP96]. For the martial arts domain we have the following excerpt:



EACH BlackBelt IS A Player WHO EITHER in judo has Rank THAT is of RankKind 'dan'
OR in karatedo has Rank THAT is of RankKind 'dan'
EACH JudoDan IS A BlackBelt WHO in judo has Rank that is of RankKind 'dan'
EACH KaratedoDan IS A BlackBelt WHO in karatedo has Rank that is of RankKind 'dan'

Figure 4: Martial arts example

$$
\begin{array}{rcl}
\mathcal{OB}_t & = & \{\text{Date}, \text{dmy}, \text{Player}, \text{PlayerName}, \text{Rank}, \ldots, \text{Outcome}\} \\
\mathcal{VL}_t & = & \{\text{dmy}, \text{PlayerName}, \text{RankKindName}, \ldots, \text{OutcomeName}\} \\
\text{Dom}_t(\text{dmy}) & = & \{\text{"d-m-y"} \mid 1 \leq \text{d} \leq 31 \wedge 1 \leq \text{m} \leq 12 \wedge 0 \leq \text{y}\} \\
\text{Dom}_t(\text{PlayerName}) & = & \text{String} \\
\text{Dom}_t(\text{RankKindName}) & = & \{\text{'kyu'}, \text{'dan'}\} \\
\text{Dom}_t(\text{LevelNr}) & = & \mathbb{N} \\
\mathcal{RL}_t & = & \{\text{has ever beaten}, \text{was born on}, \text{in judo has}, \ldots, \text{in karatedo favours}\} \\
\text{Roles}_t(\text{has ever beaten}) & = & \{\text{has ever beaten-1}, \text{has ever beaten-2}\} \\
\text{Roles}_t(\text{was born on}) & = & \{\text{was born on-1}, \text{was born on-2}\} \\
\text{Roles}_t(\text{is at}) & = & \{\text{is at-1}, \text{is at-2}\} \\
\text{Player}_t(\text{was born on-1}) & = & \text{Player} \\
\text{Player}_t(\text{was born on-2}) & = & \text{Date} \\
\text{Player}_t(\text{has ever beaten-1}) & = & \text{Player} \\
\text{Player}_t(\text{has ever beaten-2}) & = & \text{Player}
\end{array}
$$

$$\text{BlackBelt}\ \text{SubOf}_t\ \text{Player},\quad \text{JudoDan}\ \text{SubOf}_t\ \text{BlackBelt},\quad \text{KaratedoDan}\ \text{SubOf}_t\ \text{BlackBelt}$$

The derivation rules for this domain are the three subtype defining rules as specified in the graphical representation.

2.2 Abstraction Layers

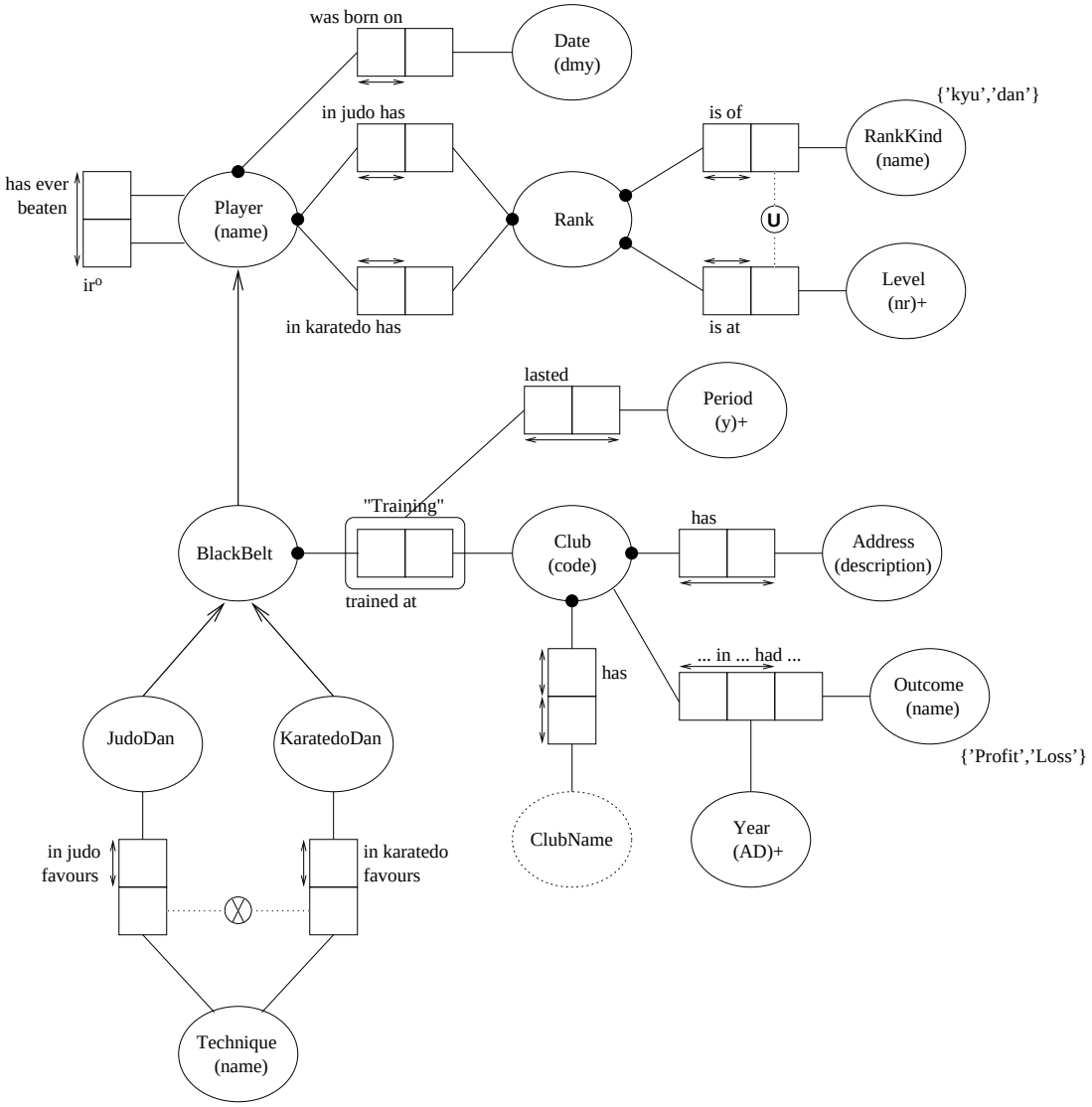
In the past, a much heard critique on Object-Role Modelling based techniques was that they were too detailed. Where ORM models show too much detail, (E)ER models lack detail. The reason being that (E)ER, by the virtue of its attribute types, already provides a rudimentary means to introduce a single abstraction level. This allows modellers to focus on what they experience as key object types (which then become the entity types). ORM forces modellers to initially regard all object types as equals. This means that an ORM diagram looks initially much more complex than an ER diagram would do. For large applications, however, the problem of complex and uncomprehensible schemas also haunts the (E)ER modeller ([CJA90]).

These observations have sparked the development of bottom-up abstraction algorithms that allow for (manual, semi-manual, or automatic) generation of higher abstraction layers by the identification of so called *major object types*. Examples of such algorithms can be found in e.g. [Ver83, Sho85, TWBK89, CJA90, CHP96].

The CDM Kernel provides modelling constructs that allow for top-down abstraction as well as bottom-up abstraction. In e.g. [De 91, DJ93], similar extensions of ORM are discussed. However, the ideas presented there remain informal, and do not establish a connection to (E)ER modelling. Furthermore, the fully automatic bottom-up abstraction algorithm described in [CHP96] generates these abstraction layers for a given flat conceptual model in the CDM Kernel. This makes the CDM Kernel highly suitable for both bottom-up and top-down modelling.

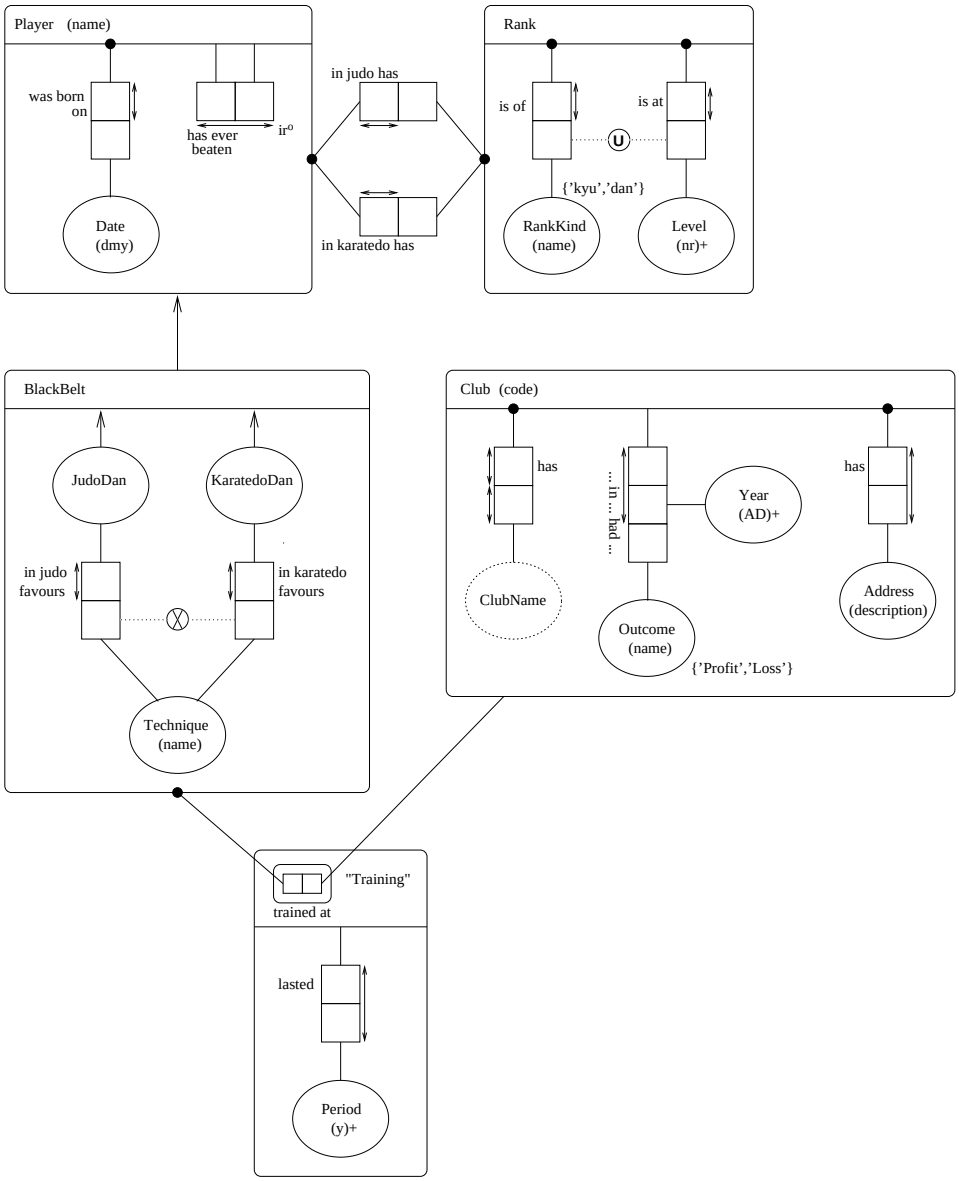
Besides the above discussed notion of abstraction, traditional ORM and ER schemas already (implicitly) feature a different notion of abstraction. In both traditional ORM and ER, but in particular in the many extensions, it was found natural to present complex types as an undividable entity. This has led to such notions as: objectification, collection types, sequence types, aggregate types, bag types, schema types, etc. As an example of this, consider the Training object type in figure 4. From a conceptual point of view, it may be more natural to treat Training as an objectification of a relationship between a BlackBelt and Club. So each relation between a BlackBelt and a Club is treated as if it is an individual instance of an object type. This objectified view is shown in figure 5. Which view is more natural depends very much on the underlying universe of discourse. In ORM this is usually detected by studying the way example facts from the universe of discourse are verbalised. The objectified view of figure vr02objc hides the underlying relationship types has and took place at, and by doing so it provides a form of abstraction.

To return to the martial arts example, after identifying the abstractions to hide the details of complex types, we can perform abstractions based on the importance of object types. For the martial arts domain, this may lead to the representation as shown in figure 6. This representation focuses on the major object types



EACH BlackBelt IS A Player WHO EITHER in judo has Rank THAT is of RankKind 'dan'
OR in karatedo has Rank THAT is of RankKind 'dan'
EACH JudoDan IS A BlackBelt WHO in judo has Rank that is of RankKind 'dan'
EACH KaratedoDan IS A BlackBelt WHO in karatedo has Rank that is of RankKind 'dan'

Figure 5: Objectification flavour of abstraction



EACH BlackBelt IS A Player WHO EITHER in judo has Rank THAT is of RankKind 'dan'
OR in karatedo has Rank THAT is of RankKind 'dan'
EACH KaratedoDan IS A BlackBelt WHO in karatedo has Rank that is of RankKind 'dan'
EACH JudoDan IS A BlackBelt WHO in judo has Rank that is of RankKind 'dan'

Figure 6: Abstractions

in the domain, and is likely to be more comprehensible by modellers and their discussion partners. It is now interesting to see that a level-one abstraction from a flat object-role model directly corresponds to an (E)ER schema. In figure 7 we have depicted the (E)ER version of the schema in figure 6. This example also illustrates how (E)ER ([BCN92]) schemas are represented in the context of the CDM Kernel. They are treated as flat conceptual schemas with a pre-determined level-one abstraction layer.

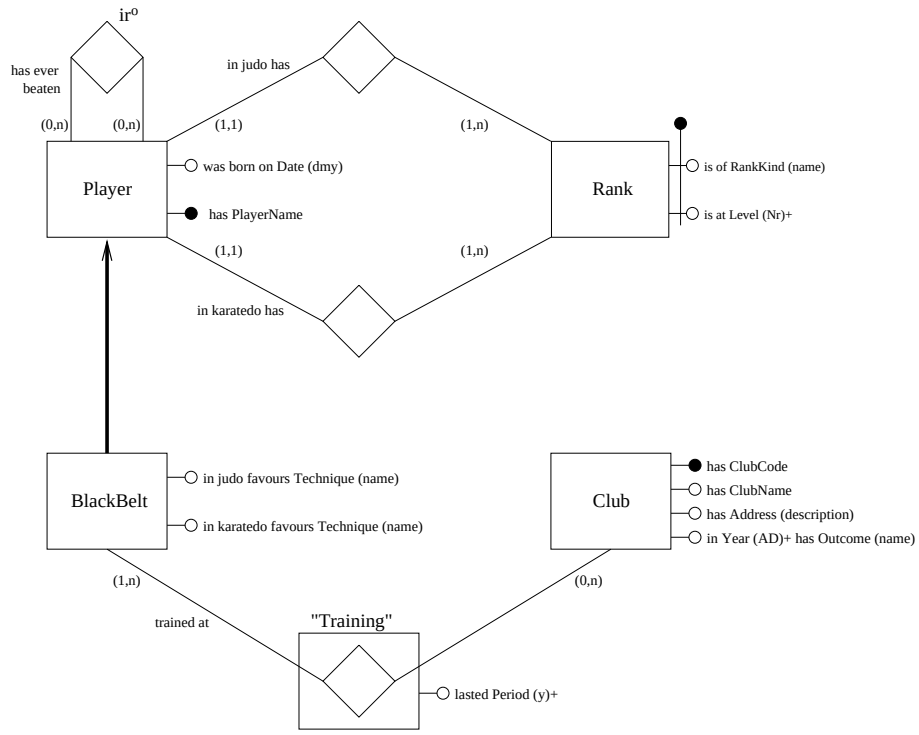


Figure 7: An ER view on the martial arts domain

Formally, the abstraction layers for the CDM Kernel are provided by the clustering function:

$$\mathsf{Cluster}_t : \mathbb{N} \times \mathcal{OB}_t \rightarrow \wp(\mathcal{TP}_t)$$

The intuitive meaning is that if $\mathsf{Cluster}(i, x) = Y$, then the types in Y are clustered to x on level i. In [CP96] some general completeness rules for these clusterings are provided. For the martial arts domain we would for example have:

$$\begin{aligned} \mathsf{Cluster}_t(0, \mathsf{Training}) &= \{\ \mathsf{Training, has, BlackBelt, took\ place\ at, Club}\ \} \\ \mathsf{Cluster}_t(1, \mathsf{Player}) &= \{\ \mathsf{Player, was\ born\ on\ date, has\ ever\ beaten, ..., dmy}\ \} \\ \mathsf{Cluster}_t(1, \mathsf{Rank}) &= \{\ \mathsf{Rank, is\ of, RankKind, ..., LevelNr}\ \} \end{aligned}$$

To distinguish between the different flavours of abstraction we also introduce the function:

$$\mathsf{CFlavour}_t : \mathcal{OB}_t \times \mathcal{TP}_t \rightarrowtail \mathtt{Flavours}$$

with as intuition that if $\mathsf{CFlavour}_t(x, y) = f$, then the clustering of object type x to y has flavour f. For example:

$$\begin{aligned} \mathsf{CFlavour}_t(\mathsf{has}, \mathsf{Training}) &= \mathit{objectification} \\ \mathsf{CFlavour}_t(\mathsf{took\ place\ at}, \mathsf{Training}) &= \mathit{objectification} \end{aligned}$$

The clusters resulting from $\mathsf{CFlavour}$ can also be interpreted as object classes in an object-oriented sense. Each of the clusters in figure 6 can be seen as an object class in an object-oriented approach. This view

also allowed us to introduce some object oriented aspects into traditional data modelling ([CP96]). In [KS92, DJ93, CH94] the effects of extending ORM or ER with object-oriented features are also discussed. The important aspects with which ORM is extended are: the overriding of inherited relationship types, the association of methods to object types, and the encapsulation of methods and clustered object types. These aspects are discussed in full detail in [CP96]. In this article we only state the way in which we have 'implemented' these features in the theory. To capture overriding of inheritance, we introduce the function $\mathsf{RoleLim}_t : \mathcal{OB}_t \times \mathcal{RO}_t \rightarrowtail \mathcal{OB}_t$. If $\mathsf{RoleLim}_t(x, p) = y$, then in the context of the relationship types clustered to x, the population of $\mathsf{Player}_t(p)$ should be limited to the population of object type y. The second object-oriented aspect is concerned with the association of methods to object types. To each schema a set $\mathcal{OP}_t \subseteq \mathcal{OB}_t \rightarrowtail \mathtt{Methods}$ of operations can be associated, accompanied by a signature function $\sigma_t : \mathcal{OP}_t \rightarrow \mathcal{TP}_t^+$. If $o \in \mathcal{OP}_t$, we have an operation with signature $\sigma_t(o)$, while the operation o itself is a function assigning a specific method for different object types (within one single subtyping hierarchy). Obviously, the methods $o(x)$ may differ for different types x in the same type hierarchy, but inheritance is the default. The way in which we use the terms operation and method is borrowed from [RBP+91]. The operations themselves are also introduced on different levels of abstraction. These abstraction levels are provided by the function: $\mathsf{Ops}_t : \mathbb{N} \rightarrow \wp(\mathcal{OP}_t)$.

Finally, as a third object-oriented aspect, the CDM Kernel offers encapsulation. The CDM Kernel offers two flavours of encapsulation. The most liberal one is encapsulation on the type level, which is provided by the function $\mathsf{Encap}_t : \mathcal{OB}_t \rightarrow \wp(\mathcal{OP}_t \cup \mathcal{TP}_t)$. This flavour of abstraction allows us to encapsulate operations and relationship types within definitions of clusterings. Only relationship types can be encapsulated, since the players of relationship may be shared among relationship types. Similarly, in most object-oriented modelling techniques attributes can be encapsulated, but the underlying domains of these attributes can *not* be encapsulated. If relationship type $r \in \mathsf{Encap}_t(x)$, then r is only visible from instances of x.

2.3 Internal Representation

In [BW92] a representation technique is introduced that allows us to specify the internal representation of a conceptual schema as a forest of trees, where each tree corresponds to one record structure and each node in the tree consists of a set of roles. An example of this is given in figure 8. The top part of this figure displays one tree corresponding to one single relational table. This table is shown in the bottom part. Please note that one-on-one relationships, like the one between Rank and the combination of RankKindName and Level, are ignored in the relational table representation. This can be done due to the fact that Rank is identified by means of a RankKand and Level, which on their turn receive their identification from RankKindName and LevelNr. As each Rank is thus identified by a RankKindName and a LevelNr, we can replace Rank completely by this combination in the relational table representation. As argued in [BW92, Bom95], the tree representation can be used for non-relational target platforms as well.

Formally, a forest representation of a flat data schema is given by a set of nodes $\mathcal{NO}_t$, a partition $\mathsf{Roles}_t : \mathcal{NO}_t \rightarrow \wp(\mathcal{RO}_t)$ yielding the roles grouped to the given node (rendering the Roles_t function symbol overloaded), and a set of labelled edges $\mathsf{Edge}_t \subseteq \mathcal{NO}_t \times \mathcal{NO}_t \times \mathcal{RL}_t$. In [BW92] a number of correctness rules for the resulting forest are given. These rules can actually be simplified slightly as we now treat objectification as a form of abstraction, leading to a simplification of the flat conceptual schema level, whereas in [BW92] exceptions were needed to cope with objectifications.

Please note that it is quite easy to extend the data modelling language with additional constructs to include, for example, indexing options for the internal representation. For instance, a predicate $\mathsf{Index}_t \subseteq \wp(\mathcal{RO}_t)$ where $\mathsf{Index}_t(P)$ signifies that there is an index defined on the combination of roles P. Obviously, the roles in P must be part of the same tree (relational table) to make it useful to have an index defined on them. Another relevant extension would be to cater for distribution of the conceptual schema over different sites. This could for instance (simplistically) be modelled by a function $\mathsf{Site}_t : \mathcal{TP}_t \rightarrow \wp(\mathtt{Sites})$.

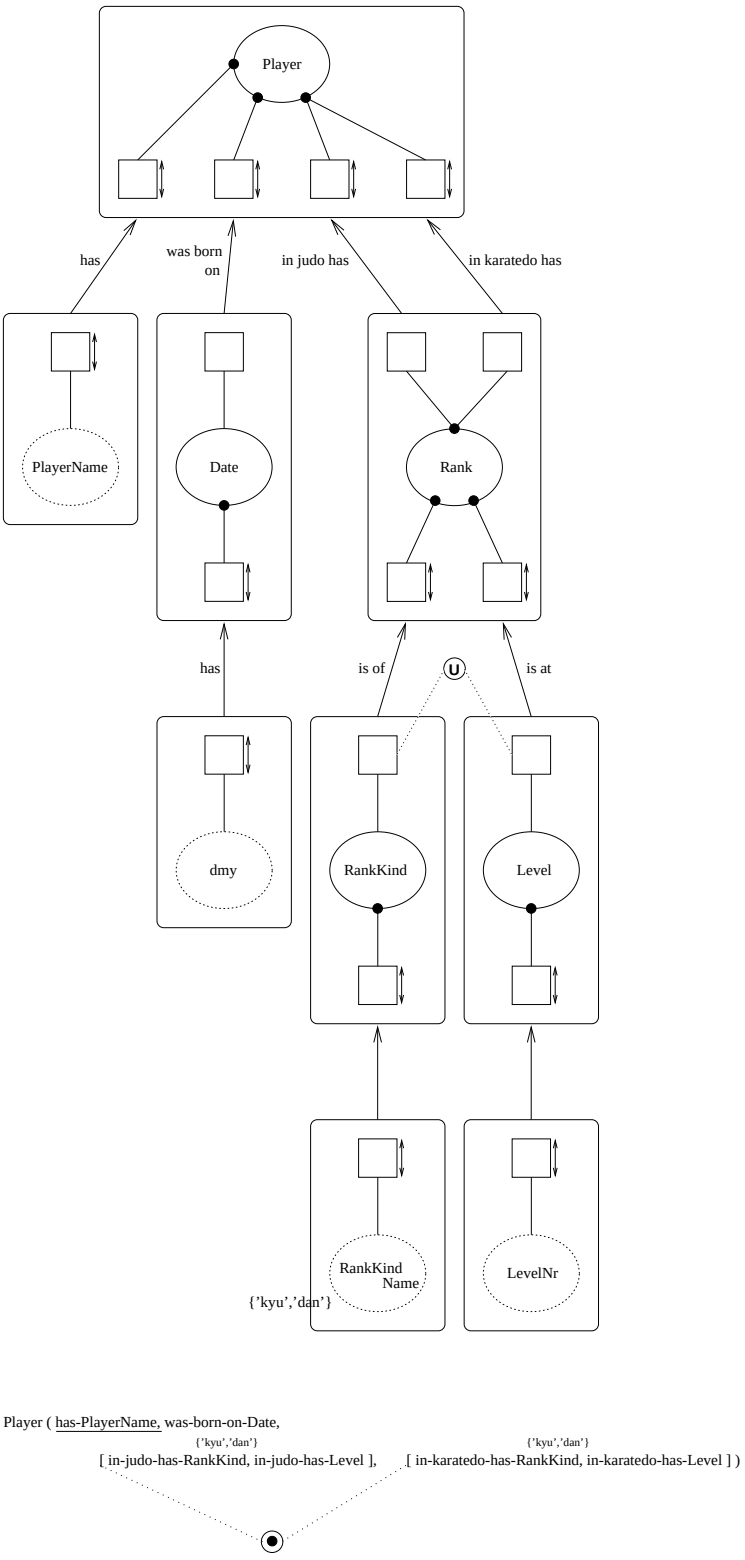


Figure 8: Internal representation by clustering of roles

2.4 Summary of Components

This completes the overview of the data schema universe. We can now put the different components together. The structural aspects of a data model are fully determined by the components of the following tuple:

$$\mathcal{IS}_t = \langle \mathcal{RL}_t, \mathcal{OB}_t, \mathcal{VL}_t, \mathcal{RO}_t, \mathcal{NO}_t, \mathsf{SubOf}_t, \mathsf{Roles}_t, \mathsf{Player}_t, \mathsf{Cluster}_t, \mathsf{CFlavour}_t, \mathsf{RoleLim}_t, \mathsf{Ident}_t, \mathsf{Edge}_t \rangle$$

The first 5 components provide the types, roles and nodes present in the information structure, and the last 9 components describe their mutual relationships providing the 'fabric' of the information structure (together with the abstraction layers and internal representation).

A complete conceptual schema over a set of concrete domains $\mathcal{DO}$, is then identified by the following components:

$$\mathcal{CS}_t \triangleq \langle \mathcal{IS}_t, \mathcal{CN}_t, \mathcal{OP}_t, \sigma_t, \mathsf{Ops}_t, \mathsf{DerRule}_t, \mathsf{UpdRule}_t, \mathsf{Encap}_t, \mathsf{Dom}_t \rangle$$

At the moment work is underway in establishing a similar universe of models for the process aspects of an information system. The CDM Evolver as it will be presented in the next section is flexible enough to be applied directly to such a process modelling technique as well.

In [CP96] a series of well-formedness rules for the CDM Kernel is provided. In this article we simply presume that the predicate $\mathsf{IsSch}(\mathcal{CS}_t)$ determines whether a given conceptual schema $\mathcal{CS}_t$ is correct.

3 Model Evolution

The previous section briefly discussed the substance of transformation/evolution, i.e. the corpus evolutionis. This section focuses on a way to model the actual evolution of the corpus evolutionis. We try do this in a generalised way such that it could also be applied to other domains than data schemas (for example process models). The next section provides the actual coupling between the components of a data schema in the CDM Kernel and the versioning mechanism. In subsection 3.6, we return to the earlier mentioned relationship between the way we model evolution of data schemas, and generic version management systems.

The model evolution framework we use consists of the following components:

$$\langle \mathcal{TI}_s, \mathcal{ELV}, \mathcal{ELC}, \mathsf{VClass} \rangle$$

A linear time axis is provided by $\mathcal{TI}_s$. To cater for the fact that model elements can have different versions, we introduce a set $\mathcal{ELV}$ of $\mathcal{EL}$ement $\mathcal{V}$ersions. The elements of this set are abstract representations of the underlying version of model elements. A simple classification system, e.g. distinguishing between types and constraints, is provided for these elements by a set of $\mathcal{EL}$ement $\mathcal{C}$lasses: $\mathcal{ELC}$. The function $\mathsf{VClass} : \mathcal{ELV} \rightarrow \mathcal{ELC}$ provides the class for each element version. Below we discuss the aim and pragmatics of these components in more detail. In the next section we provide a concrete definition of $\mathcal{ELV}$ and $\mathcal{ELC}$ for the CDM Kernel.

3.1 Granularity of Versioning

A data schema, or any model for that matter, can be seen as a set of elements. Some of these elements may be atomic while some of them may be composed, e.g. relations between other elements. In a version management system, one can decide to maintain different versions of models as a whole, or maintain different versions of the elements. We opt for the latter approach. The advantage is that the version history is not just a series of snapshots of the complete model, but that one can actually track the version history (evolution) of single elements. For instance, the evolution of a relationship type during the modelling

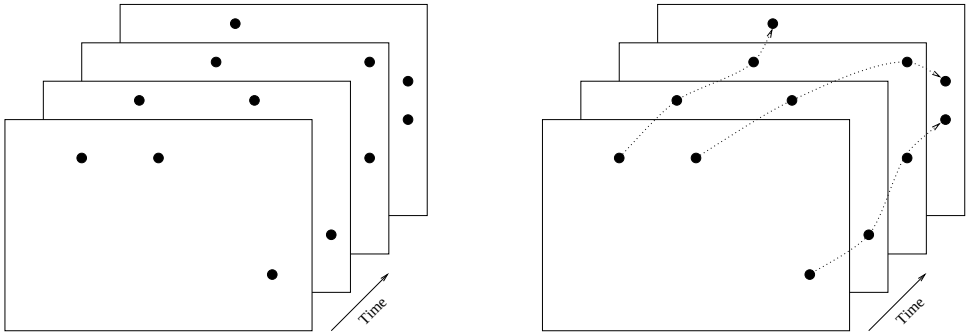


Figure 9: Granularity of versioning

process. As an illustration, consider figure 9. The left hand side of this figure depicts a sequence of snapshots, whereas the right hand side shows the same sequence of snapshots, but this time we can see the evolution of the individual elements.

Below we will see how we are able to model the interaction of different elements in the course of time. For instance, consider the schema transformation shown in figure 10. This transformation is taken from [PH95], and is an example of an equivalence preserving schema transformation. In discussing reactions between element histories, we adopt the terminology used in quantum physics. For the example transformation (left to right) we can say that the type of the reaction is a *fact type reaction*, where teaches and advises *fuse* to become ... performed ... for .... This reaction *absorbs* the uniqueness constraints on the two original fact types. The reaction *emits* the uniqueness constraint on the resulting fact type, together with the Service object type and its associated value type (name) and value constraint {'teaching', 'advice'}. We can model this behaviour explicitly in our versioning mechanism, providing deeper insight in the actual design processes, moving away from the tradtional snapshot view (i.e. searching under the lamppost).

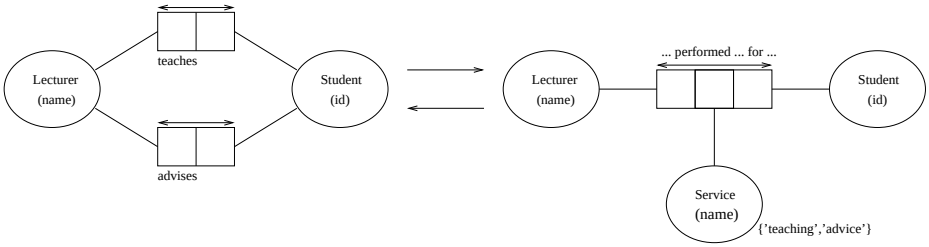


Figure 10: Example schema transformation

3.2 Time Axis

To model the evolution of a model, some linear time axis is required. In subsection 3.6 we discuss how to deal with alternative evolution-versions. The time axis we use is provided as: $\mathcal{TI}_s \triangleq \langle \mathcal{TI}, \{<\} \rangle$, where $\mathcal{TI}$ is a set of points in time, and $<$ provides a complete total order on $\mathcal{TI}$. The $<$ relation allows us to define a one step increment function $\triangleright : \mathcal{TI} \rightarrow \mathcal{TI}$ on the time axis:

$$\triangleright t_1 = t_2 \quad \Leftrightarrow \quad t_1 < t_2 \wedge \neg \exists_s [t_1 < s < t_2]$$

From the order on the time axis we can also derive:

$$\begin{aligned} t_1 = t_2 \quad &\Leftrightarrow \quad \neg(t_1 < t_2 \vee t_2 < t_1) \\ t_1 \leq t_2 \quad &\Leftrightarrow \quad \neg(t_2 < t_1) \end{aligned}$$

3.3 Element Evolutions

As stated before, we model the versioning history of each element separately. To elegantly model the version history of these elements, and their interactions, we introduce the notion of an extra-temporal (time independent) *element identifier*. This concept is indeed similar to the object identifier notion of object-oriented approaches. Let $\mathcal{EID}$ be the set of element identifiers. At each point in time, an element may have a version. To remain as general as possible, we do not yet want to elaborate on what such a version actually is. Therefore we use the set $\mathcal{ELV}$ of version elements to identify these versions, and for now treat these versions as abstract objects. This means that we can now associate to each element identifier a function $\mathcal{TI} \rightarrowtail \mathcal{ELV}$ describing the evolution of the element in terms of its versions in the course of time. The history of all element identifiers, the complete model history, can therefore be seen as a function: $H : \mathcal{EID} \rightarrow (\mathcal{TI} \rightarrowtail \mathcal{ELV})$. The set of all possible $\mathcal{M}$odel $\mathcal{HI}$stories is then: $\mathcal{MHI} \triangleq \mathcal{EID} \rightarrow (\mathcal{TI} \rightarrowtail \mathcal{ELV})$.

A model history as such does not capture the full spectre of the evolution of a model. What is still missing are interactions that have taken place between model elements in the course of time. This is the extra spice in the meal that makes a model history into a true model evolution. The interactions of elements in the course of their evolution can be modelled as a tuple $\langle E, c, t \rangle$, where $E \subseteq \mathcal{EID}$ is a set of element identifiers which are together involved in some form of *reaction* of type $c \in \mathcal{ELC}$ at point in time t. The general set of such $\mathcal{REA}$ctions is: $\mathcal{REA} \triangleq \wp(\mathcal{EID}) \times \mathcal{ELC} \times \mathcal{TI}$. On reactions, the following access functions can be defined:

$$\mathsf{RTime}(\langle E, c, t \rangle) \triangleq t \quad \mathsf{Reagents}(\langle E, c, t \rangle) \triangleq E \quad \mathsf{RClass}(\langle E, c, t \rangle) \triangleq c$$

A complete evolution of models build from version identifiers $\mathcal{ELV}$ with element identifiers $\mathcal{EID}$ can now be described by a tuple:

$$\langle R, H \rangle \in \wp(\mathcal{REA}) \times \mathcal{MHI}$$

where R provides the reactions and H is provides the historical component. The set of all possible $\mathcal{M}$odel $\mathcal{EV}$olutions is then:

$$\mathcal{MEV} \triangleq \wp(\mathcal{REA}) \times \mathcal{MHI}$$

3.4 Correctness of Model Evolutions

The CDM Kernel provides a number of rules to which a conceptual schema should adhere. Some of these rules are actually formulated as optional rules (*electronic switches*) to cater for well-formedness variations based for the different data modelling schools. In this subsection we introduce some general well-formedness rules on model *evolutions* $\langle R, H \rangle \in \mathcal{MEV}$. It should be noted that also for the model evolutions one may chose to add extra rules based on ones philosophical stance with respect to conceptual data modelling. Here we can do nothing more but provide a general framework.

If all of the MEW (**M**odel **E**volution **W**ell-formedness) axioms we introduce in this section hold, then a model evolution $\langle R, H \rangle \in \mathcal{MEV}$ is deemed correct: $\mathsf{IsModEvol}(R, H)$. In the definitions provided in this section we shall use the following abbreviation $h{\downarrow}t \triangleq t \in \mathsf{dom}(h)$, where $\mathsf{dom}(h)$ is the domain of function h. So if $h{\downarrow}t$, then function h is defined for t.

An element evolution cannot be empty:

[MEW1] $e \in \mathcal{EID} \Rightarrow \exists_t [H(e){\downarrow}t]$

The nature of the relationship between reactions and element histories is captured by the following two axioms:

[MEW2] If $r \in R$, then:

$$\forall_{e \in \mathsf{Reagents}(r)} [H(e){\downarrow}\mathsf{RTime}(r) \ \vee \ H(e){\downarrow\triangleright}\mathsf{RTime}(r)]$$

[MEW3] If $e \in \mathcal{EID}$ such that $H(e)\downarrow t \wedge H(e)\downarrow\!\triangleright t$, then:

$$H(e)(t) \neq H(\triangleright t) \Rightarrow \exists_{r \in R}\,[e \in \mathsf{Reagents}(r) \wedge \mathsf{RTime}(r) = t]$$

The first axiom states that all elements involved in some reaction must be alive at, or immediately after, the reaction takes place. In plain words this means that dead elements cannot partake in any reaction. The second axiom requires all changes in an element's evolution to be the result of some reaction.

The next rule states that evolution of elements is bound to classes. For example, a type may not evolve into a method, and a constraint may not evolve into a derivation rule. In other words there is no magic; we do not allow for frogs to turn into princesses. This is formalised in the following axiom:

[MEW4] (history separation) If $e \in \mathcal{EID}$, then:

$$H(e)\downarrow t \wedge H(e)\downarrow t' \;\Rightarrow\; \mathsf{VClass}(H(e)(t)) = \mathsf{VClass}(H(e)(\triangleright t))$$

In concrete instances one may sometimes debate whether this rule should be enforced as a hard rule, or as a *deontic* rule. In the next section we will see that, although we distinguish between different types in a conceptual schema, an element evolution describing the evolution of a type is allowed to 'roam' in the set of all types.

The above axiom allows us to introduce a classification of element identifiers. This classification is defined in the context of a version history H using the existing classification provided by VClass:

$$\mathsf{EClass}_H : \mathcal{EID} \rightarrow \mathcal{ELC}$$
$$\mathsf{EClass}_H(e) \triangleq \text{ the unique } c \text{ such that } \forall_{H(e)\not\downarrow}\,[\mathsf{VClass}(H(e)(t)) = c]$$

Using the classification on element identifiers, we are also able to take a closer look at the kinds of reactions that can occur between element histories. The input and output of a reaction are identified by the functions:

$$\mathsf{Input}_H : \mathcal{REA} \rightarrow \wp(\mathcal{EID})$$
$$\mathsf{Input}_H(r) \triangleq \{e \in \mathsf{Reagents}(r) \mid H(e)\downarrow\mathsf{RTime}(r)\}$$
$$\mathsf{Output}_H : \mathcal{REA} \rightarrow \wp(\mathcal{EID})$$
$$\mathsf{Output}_H(r) \triangleq \{e \in \mathsf{Reagents}(r) \mid H(e)\downarrow\!\triangleright \mathsf{RTime}(r)\}$$

A catalyst is an element that partakes in a reaction without being changed itself. For our reactions we can identify catalysts by:

$$\mathsf{Catalyst}_H : \mathcal{REA} \rightarrow \wp(\mathcal{EID})$$
$$\mathsf{Catalyst}_H(r) \triangleq \{e \in \mathsf{Input}_H(r) \cap \mathsf{Output}_H(r) \mid H(e)(t) = H(e)(\triangleright t)\}$$

If a reaction r is stated to have class $\mathsf{RClass}(r)$ then this must be visible in the input and output of the reaction. So there must be some input or output of this class:

[MEW5] If $r \in R$, then:

$$c \in \mathsf{RClass}(r) \Rightarrow \exists_{h \in \mathsf{Input}_H(r) \cap \mathsf{Output}_H(r)}\,[\mathsf{EClass}_H(h) = c]$$

In a reaction some element histories of the same class may fuse, and some may be split. These respective elements are identified by:

$$\mathsf{Fusion}_H : \mathcal{REA} \rightarrow \wp(\mathcal{EID})$$
$$\mathsf{Fusion}_H(r) \triangleq \{h \in \mathsf{Input}_H(r) \mid \mathsf{HClass}_H(h) = \mathsf{RClass}(r)\}$$
$$\mathsf{Fission}_H : \mathcal{REA} \rightarrow \wp(\mathcal{EID})$$
$$\mathsf{Fission}_H(r) \triangleq \{h \in \mathsf{Output}_H(r) \mid \mathsf{HClass}_H(h) = \mathsf{RClass}(r)\}$$

Sometimes an element history may absorb element histories of another class, or vice versa. For instance a constraint may be absorbed when an objectification is transformed into a ternary. To identify such histories we introduce:

$$\mathsf{Absorption}_H : \mathcal{REA} \to \wp(\mathcal{EID})$$
$$\mathsf{Absorption}_H(r) \triangleq \mathsf{Input}_H(r) - \mathsf{Fusion}_H(r)$$

$$\mathsf{Emission}_H : \mathcal{REA} \to \wp(\mathcal{EID})$$
$$\mathsf{Emission}_H(r) \triangleq \mathsf{Output}_H(r) - \mathsf{Fission}_H(r)$$

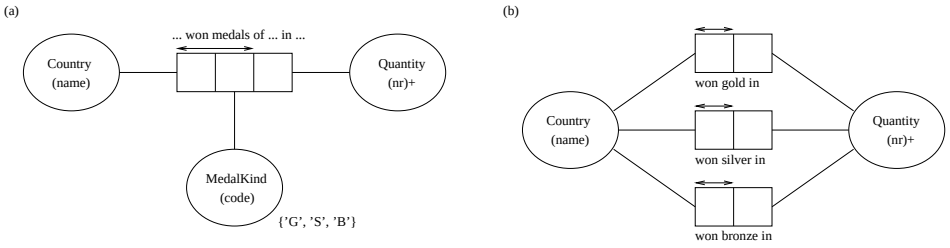


Figure 11: Olympic Games universe of discourse

As an example of what can happen during a schema transformation, consider the two Olympic Games schemas depicted in figure 11. The transformation from the first schema to the second schema is based on the observation that there are exactly three kinds of medals. So the ternary fact type may be specialised into three binaries, one for each medal kind. You may visualise the transformation from schema (a) into schema (b) thus:

Fusion of:
won medals of ... in, MedalKind (code)

Fission to:
won gold in, won silver in, and won bronze in

Absorption of:
uniqueness constraint over first two roles of won medals of ... in,
enumeration constraint { 'G', 'S', 'B' }

Emission of:
uniqueness constraints on first roles of won gold in, won silver in, and won bronze in

What we hope to illustrate with this example, despite its simplicity, is that explicit modelling of the interaction between elements during their evolution provides more insight into the evolution process. One could even explicitly include modelling decisions by storing a description of the schema transformation that caused the 'reaction', a short explanation of why it was applied, and what role each of the involved elements plays in the reaction. So:

$$\mathsf{ResultsFrom} : \mathcal{REA} \to \texttt{SchemaTransformation}$$
$$\mathsf{Reason} : \mathcal{REA} \rightarrowtail \texttt{Text}$$
$$\mathsf{Role} : \mathcal{REA} \times \mathcal{EID} \rightarrowtail \texttt{Text}$$

This, we believe, should provide modellers (and project managers for that matter) with the additional insight into the modelling process as needed according to [CKSI87, CKI88]. We hope that this way, the light will start shining on the other side of the street as well.

3.5 Derivation of Model Versions

A complete model snapshot at a given point in time t is easily derived from a complete evolution $\langle R, H \rangle$ by determining the set of version identifiers in the snapshot:

$$\mathsf{Snapshot}_H : \mathcal{TI} \rightarrow \wp(\mathcal{ELV})$$
$$\mathsf{Snapshot}_H(t) \triangleq \{ H(e)(t) \mid H(e)\downarrow t \}$$

3.6 Relation to Generic Version Management

We are now in a position to take closer look at the relationship between the version management system proposed in this article, and the more generic version management systems as discussed in e.g. [Kat90, BKKK87].

The version management system proposed in this article focuses on information system modelling processes, and in particular the evolution of the resulting models in the course of time. In doing so we, initially, do not cater for things like alternative versions of models. Initially we only acknowledge the existence of one time-line along which the models have evolved. Our focus is on an elegant description and construction of this model evolution in the context of information system design.

A generic version management framework is in our opinion a suitable (and efficient) implementation platform for the version management system discussed in this article. In doing so, we would obtain things like alternative versions and alternative courses of model evolutions for free. To do this we should regard a model evolution (like depicted in figure 1) as a (possible) configuration in the framework of Katz ([Kat90]). Alternative evolution processes then simply correspond to alternative configurations. The framework described in [Kat90] also provides ways to minimise storage of alternative evolutions by allowing for dynamic construction of configurations.

4 Evolution in the CDM Kernel

We have now discussed the two main components of the CDM Evolver: the *corpus evolutionis* and the *version manager*. All that remains to be done is to provide the *glue* to interconnect these two components. We do this by introducing a universe of data schemas. This universe provides the boundaries of schema evolution. The elements that make up this universe then provide the element versions needed to make the connection to the evolution mechanism. From this we can then also derive what a schema version within the universe is.

4.1 CDM Schema Universe

The universe of data schemas is built from the extra temporal versions of the schema components as identified in section 2. The notion of a universe in which data schemas evolve has been used before in e.g. [PW95, PW94].

Formally, the information structure universe is provided by:

$$\mathcal{IS} = \langle \mathcal{RL}, \mathcal{OB}, \mathcal{VL}, \mathcal{RO}, \mathcal{NO}, \mathsf{SubOf}, \mathsf{Roles}, \mathsf{Player}, \mathsf{Cluster}, \mathsf{CFlavour}, \mathsf{RoleLim}, \mathsf{Ident}, \mathsf{Edge} \rangle$$

The base sets, like $\mathcal{RL}$ and $\mathcal{OB}$, provide the set of all possible relationship types and object types respectively. The functions defining the fabric of the information structure, like $\mathsf{Player} : \mathcal{RO} \rightarrow \mathcal{OB}$ and $\mathsf{Roles} : \mathcal{RL} \rightarrow \wp(\mathsf{Roles})$, provide the players for any role that may be part of a schema and the set of roles that may ever be associated to any relationship type respectively.

Similarly, a complete data schema universe over a set of concrete domains $\mathcal{DO}$, is identified by the following (extra-temporal) components:

$$\mathcal{CS} \triangleq \langle \mathcal{IS}, \mathcal{CN}, \mathcal{OP}, \sigma, \mathsf{Ops}, \mathsf{DerRule}, \mathsf{UpdRule}, \mathsf{Encap}, \mathsf{Dom} \rangle$$

We presume that the atomic base sets of the data schema universe: $\mathcal{RL}, \mathcal{OB}, \mathcal{VL}, \mathcal{RO}, \mathcal{NO}, \mathcal{CN}, \mathcal{OP}$ are exclusive. Previously we stated that they are exclusive for each single version, but we now require this to be the case for the entire universe as a whole We can now formally glue the CDM Kernel to the evolution mechanism by stating:

$$\begin{aligned} \mathcal{ELV} &\triangleq \mathcal{TP} \cup \mathcal{RO} \cup \mathcal{NO} \cup \mathcal{CN} \cup \mathcal{OP} \\ \mathcal{ELC} &\triangleq \{\mathcal{TP}, \mathcal{RO}, \mathcal{NO}, \mathcal{CN}, \mathcal{OP}\} \\ \mathsf{VClass}(e) &\triangleq \mathbb{X} \in \{\mathcal{TP}, \mathcal{RO}, \mathcal{NO}, \mathcal{CN}, \mathcal{OP}\} \text{ such that } e \in \mathbb{X}. \end{aligned}$$

Note that we have combined all types into one single set ($\mathcal{TP}$). This means that for instance a relationship type is allowed to evolve into an object type; but not into a constraint.

From the above definition may seem to follow that the player of a role may not change in the course of time, or that an object type may not change its subtypes. However, the object types in the universe are simply treated as element *versions*. For example, changing a cluster of an object type, or change a subtyping, requires the involved object types *versions* to be replaced by 'fresh' object type *versions*. As shown above, and before in e.g. [PW95] and [PW94] this way of describing allows for elegant definitions. From an implementational point of view, one could regard an object type version from $\mathcal{OB}$ as the combination of its properties: name, supertypes, clustered types, etc. Whenever one of these properties changes, we would automatically have a 'fresh' object type. A possible class definition in an OO style would be:

```
VersionElement = CLASS
   Name:       STRING
END CLASS;

Type = CLASS (VersionElement)
   DerRule:    DerivationRules (OPTIONAL);
   UpdRule:    UpdateRules     (OPTIONAL);
END CLASS;

ObjectType = CLASS (Type)
   SuperTypes: SET  OF ObjectType;
   Encap:      SET  OF (Operation UNION Type);
   Ident:      LIST OF RolePair;
   Cluster:    FUNCTION Natno TO SET OF Type;
   Flavour:    FUNCTION Type  TO Flavours;
END CLASS;
```

4.2 CDM Schema Versions

Now that we have glued the evolution mechanism to the CDM schema universe, we can derive versions of the schemas for a given schema evolution $\langle R, H \rangle$. Using $\mathsf{Snapshot}_H(t)$ the schema version at t in history H can be derived by separating out the different base sets. For each of the base sets $\mathbb{X} \in \{\mathcal{TP}, \mathcal{RL}, \mathcal{OB}, \mathcal{VL}, \mathcal{RO}, \mathcal{NO}, \mathcal{CN}, \mathcal{OP}\}$ we can define:

$$\mathbb{X}_t \triangleq \mathbb{X} \cap \mathsf{Snapshot}_H(t)$$

The remaining components (all functions and predicates) of a schema version can now simply be derived by limiting the extra temporal domains to the current versions of the base sets, so:

$$\mathsf{SubOf}_t \triangleq \{\langle x, y \rangle \in \mathsf{SubOf} \mid x, y \in \mathcal{OB}_t\}$$

$$\begin{aligned} \mathsf{Roles}_t &\triangleq \{\langle x, R\rangle \in \mathsf{Roles} \mid x \in \mathcal{RL}_t \cup \mathcal{NO}_t \wedge R \subseteq \mathcal{RO}_t\} \\ \mathsf{Player}_t &\triangleq \{\langle r, x\rangle \in \mathsf{Player} \mid r \in \mathcal{RO}_t \wedge x \in \mathcal{OB}_t\} \\ &\ldots \end{aligned}$$

A direct result of this definition is, what might be expected intuitively, that for each schema component $\mathbb{X} \in \{\mathcal{RL}, \mathcal{OB}, \mathcal{VL}, \mathcal{RO}, \mathcal{NO}, \mathsf{SubOf}, \mathsf{Roles}, \mathsf{Player}, \mathsf{Cluster}, \mathsf{CFlavour}, \mathsf{RoleLim}, \mathsf{Ident}, \mathsf{Edge}, \mathcal{CN}, \mathcal{OP}, \sigma, \mathsf{Ops}, \mathsf{DerRule}, \mathsf{UpdRule}, \mathsf{Encap}, \mathsf{Dom}\}$ we have:

$$\mathbb{X}_t \subseteq \mathbb{X}$$

One can quite easily see that it is not hard to extend this framework when taking additional aspects of conceptual schemas into consideration. For example, verbalisations (names) of the object types and relationship types in a schema. In this article we have not elaborated on these issues, however, during the evolution of a conceptual schema there is also the need to change the names of object types or the verbalisation of fact types. For a detailed study of fact verbalisation and extensions needed to the formal model of a conceptual schema, refer to e.g. [HPW94], [HPW96].

4.3 Well-formedness of Evolution

FInally, besides well-formedness rules on versions, one might want to formulate rules that limit the evolution of a data schema, i.e. transition oriented constraints. Specifying such rules, however, can be a rather arbitrary process which depends on the kind of evolution one has in mind. In [PW95], [PW95] and [PW94] some example rules have been formulated in the context of schema evolution due to evolution of the universe of discourse. However, for evolution during the design phase of an information system, one would typically like to be more liberal.

Although we do not formulate such rules explicitly here, one might imagine having a set of extra rules EW (**E**volution **W**ell-formedness) that govern the evolution of data schemas. Then we can now refine the $\mathsf{IsModEvol}$ predicate to the CDM Kernel specific definition:

$$\begin{aligned} \mathsf{IsCDMEvol}(R, H) \quad &\Leftrightarrow \quad \mathsf{IsModEvol}(R, H) \ \wedge \ \forall_{t \in \mathcal{TI}} \left[\mathsf{IsSch}(\mathcal{CS}_t)\right] \\ &\wedge \quad \langle R, H\rangle \text{ obeys the EW rules} \end{aligned}$$

5 Conclusions

In this article we have defined a version management system for schema evolution: the CDM Evolver. It can be seen as the finishing touch to the existing CDM Kernel. This version management system allows us to model the evolution of data schemas in the CDM Kernel. Two important features are the ability to describe the evolution of data schemas through the entire modelling process including the internal representation, and the fact that it can model the interaction between model elements in the course of time. This latter property is believed to be crucial to provide a better insight into schema design processes. Both for modellers themselves as well as their managers.

The version management system has been setup such that it is extendible and easy to adapt to other models. This means that as such it is not only useful in the context of the CDM Kernel, but can be applied to other modelling techniques.

Currently, work is underway in implementing the CDM Kernel, and the version management system presented here will find its place in this implementation as well.

Acknowledgments

We would like to thank A.H.M. ter Hofstede for his valuable comments the first drafts of this article

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.

[BHW91] P. van Bommel, A.H.M. ter Hofstede, and Th.P. van der Weide. Semantics and verification of object-role models. *Information Systems*, 16(5):471–495, October 1991.

[BKKK87] J. Banerjee, W. Kim, H.J. Kim, and H.F. Korth. Semantics and Implementation of Schema Evolution in Object-Oriented Databases. *SIGMOD Record*, 16(3):311–322, December 1987.

[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.

[Bom95] P. van Bommel. *Database Optimization: An Evolutionary Approach*. PhD thesis, University of Nijmegen, Nijmegen, The Netherlands, 1995.

[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.

[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] P.N. Creasy and W. Hesse. Two-level NIAM: A way to get it object-oriented. In T.W. Olle A.A. Verrijn-Stuart, editor, *Proceedings of the IFIP WG 8.1 CRIS-94 Conference on Methods and Associated Tools for the Information Life Cycle*, pages 209–221, Maastricht, The Netherlands, September 1994.

[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.

[CJA90] C.R. Carlson, W. Ji, and A.K. Arora. The Nested Entity-Relationship Model. In F.H. Lochovsky, editor, *Proceedings of the Eight International Conference on Entity-Relationship Approach*, Entity-Relationship Approach to Database Design and Querying, pages 43–57, Toronto, Canada, 1990. Elsevier Science Publishers.

[CKI88] B. Curtis, H. Krasner, and N. Iscoe. A Field Study of the Software Design Process for Large Systems. *Communications of the ACM*, 31(11):1268–1287, November 1988.

[CKSI87] B. Curtis, H. Krasner, V. Shen, and N. Iscoe. On Building Software Process Models Under the Lamppost. In *Proceedings of the 9th International Conference on Software Engineering*, pages 96–103, Monterey, California, 1987. IEEE Computer Society Press.

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

[De 91] O.M.F. De Troyer. The OO-Binary Relationship Model: A Truly Object Oriented 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 561–578, Trondheim, Norway, May 1991. Springer-Verlag.

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

[DJ93] O.M.F. De Troyer and R. Janssen. On Modularity for Conceptual Data Models and the Consequences for Subtyping, Inheritance & Overriding. In E.K. Elmagarmid and E.J. Neuhold, editors, *Proceedings of the 9th IEEE Conference on Data Engineering (ICDE 93)*, pages 678–685. IEEE Computer Society Press, 1993.

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

[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.

[Hai91] J.-L. Hainaut. Entity-generating Schema Transformation for Entity-Relationship Models. In *Proceedings of the 10th International Conference on the Entity-Relationship Approach*, Lecture Notes in Computer Science, San Mateo, California, 1991. Springer-Verlag.

[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.

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

[HCTJ93] J-L. Hainaut, M. Chandelon, C. Tonneau, and M. Joris. Contribution to a Theory of Database Reverse Engineering. In *Proceedings of the IEEE Working Conference on Reverse Engineering*, Baltimore, Massachusetts, May 1993. IEEE Computer Society Press.

[HEH+94] J-L. Hainaut, V. Englebert, J. Henrard, J-M. Hick, and D. Roland. Database Evolution: the DB-MAIN Approach. In P. Loucopoulos, editor, *Proceedings of the 13th International Conference on the Entity-Relationship Approach*, volume 881 of *Lecture Notes in Computer Science*, pages 112–131, Manchester, United Kingdom, December 1994. Springer-Verlag.

[HP95] T.A. Halpin and H.A. Proper. Database schema transformation and optimization. In M.P. Papazoglou, editor, *Proceedings of the OOER'95, 14th International Object-Oriented and Entity-Relationship Modelling Conference*, volume 1021 of *Lecture Notes in Computer Science*, pages 191–203, Gold Coast, Australia, December 1995. Springer-Verlag.

[HPW94] A.H.M. ter Hofstede, H.A. Proper, and Th.P. van der Weide. Grammar Based Information Modelling. Technical Report CSI-R9414, Submitted for publication, Computing Science Institute, University of Nijmegen, Nijmegen, The Netherlands, October 1994.

[HPW96] A.H.M. ter Hofstede, H.A. Proper, and Th.P. van der Weide. Exploring Fact Verbalisations for Conceptual Query Formulation. In *Proceedings of the Second International Workshop on Applications of Natural Language to Databases (NLDB'96)*, pages 40–51, Amsterdam, The Netherlands, June 1996. IOS Press.

[HTJC93a] J-L. Hainaut, C. Tonneau, M. Joris, and M. Chandelon. Schema Transformation Techniques for Database Reverse Engineering. In *Proceedings of the 12th International Conference on the Entity-Relationship Approach*, Lecture Notes in Computer Science, Dallas, Texas, December 1993. Springer-Verlag.

[HTJC93b] J-L. Hainaut, C. Tonneau, M. Joris, and M. Chandelon. Transformation-based Database Reverse Engineering. In *Proceedings of the 12th International Conference on the Entity-Relationship Approach*, Lecture Notes in Computer Science, Dallas, Texas, December 1993. Springer-Verlag.

[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.

[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.

[Kat90] R.H. Katz. Toward a Unified Framework for Version Modelling in Engineering Databases. *ACM Computing Surveys*, 22(4):375–408, 1990.

[Kob86] I. Kobayashi. Classification and transformations of binary relationship relation schemata. *Information Systems*, 11(2):109–122, 1986.

[Kri94] G. Kristen. *Object Orientation, the KISS Method: From Information Architecture to Information System*. Addison-Wesley, Reading, Massachusetts, 1994.

[KS92] G. Kappel and M. Schrefl. Local referential integrity. In G. Pernul and A.M. Tjoa, editors, *11th International Conference on the Entity-Relationship Approach*, volume 645 of *Lecture Notes in Computer Science*, pages 41–61, Karlsruhe, Germany, October 1992. Springer-Verlag.

[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.

[OPF94] J.L.H. Oei, H.A. Proper, and E.D. Falkenberg. Evolving Information Systems: Meeting the Ever-Changing Environment. *Information Systems Journal*, 4(3):213–233, 1994.

[PH95] 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, Australia, July 1995.

[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.

[PW95] 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.

[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.

[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.

[Sho85] P. Shoval. Essential information structure diagrams and database schema design. *Information Systems*, 10(4):417–423, 1985.

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

[TS92] M.T. Tresch and M.H. Scholl. Meta Object Management and its Application to Database Evolution. In G. Pernul and A.M. Tjoa, editors, *11th International Conference on the Entity-Relationship Approach*, volume 645 of *Lecture Notes in Computer Science*, pages 299–321, Karlsruhe, Germany, October 1992. Springer-Verlag.

[TWBK89] T.J. Teorey, G. Wei, D.L. Bolton, and J.A. Koenig. ER Model Clustering as an Aid for User Communication and Documentation in Database Design. *Communications of the ACM*, 32(8):975–987, August 1989.

[Ver83] D. Vermeir. Semantic Hierarchies and Abstractions in Conceptual Schemata. *Information Systems*, 8(2):117–124, 1983.