

A Unifying Object Role Modelling Theory

G.H.W.M. Bronts¹ S.J. Brouwer¹ C.L.J. Martens¹ H.A. Proper^{2,3}

¹Computing Science Institute
University of Nijmegen
Toernooiveld
6525 ED Nijmegen
The Netherlands

²Department of Computer Science
University of Queensland
Brisbane
Australia 4072
E.Proper@acm.org

June 23, 2004

PUBLISHED AS:

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.

Abstract

This article presents the idea of defining a kernel for object role modelling techniques, upon which different drawing styles can be based. We propose such a kernel (the ORM kernel) and define, as a case study, an ER and a NIAM drawing style on top of it.

One of the prominent advantages of such a kernel is the possibility to build a CASE-tool supporting multiple methods. Such a CASE-tool would allow users with different methodological backgrounds to use it and view the modelled domains in terms of their favourite method. This is illustrated using a running example of a concrete domain in which we use the ORM kernel in combination with the NIAM and ER drawing style.

1 Introduction

In the last decades, a plethora of modelling techniques for the design of information systems has been developed (see e.g. [Bub86]). In general, these modelling techniques provide only a crude and incomplete description of their syntax, and semantics ([HW92]). This situation has led to the Methodology Jungle ([Avi95]). In particular a wide range of data modelling techniques exists, for instance:

ER versions: [Che76], [TMHY80], [EWH85], [HNSE87], [HE92], [MSW92],

Object-Role Modelling versions: [VB82], [NH89], [Win90], [BHW91], [De 91], [SZ91], [HW93], [HO92], [JG87] and the more theoretical IFO: [AH87].

Quite often, the difference between contemporary data modelling techniques is limited to ‘cosmetic’ issues. Even if there are fundamental differences, most modelling techniques have more common than differing

³Part of this work has been supported by an Australian Research Council grant, entitled: “An expert system for improving complex database design”

aspects. This observation has led to the idea of defining a general ORM (Object Role Modelling) kernel as a greatest common divisor.

The definition of such a kernel has several advantages. CASE-tools supporting multiple data modelling techniques can be developed on top of the ORM kernel, e.g. leading to an ER and a NIAM view on the same ORM data model. A further result is that in one information system development project, multiple modelling techniques can be employed simultaneously. As a result, project members can use their own preferred modelling technique, and any investment in e.g. 'old' NIAM or ER models do not go to waste. The idea of defining a common repository for multiple CASE-tools is not new. The AD/Cycle project was also an attempt to define such a common repository. However, the idea of using a common repository to derive different views on the modelled domain is new. As an illustration of the relationship between the ORM kernel and other modelling techniques, consider figure 1. Besides a graphical representation, there is also the possibility of using a textual language such as LISA-D ([HPW93], [HPW94]) or FORM ([HH93]) as a way to represent models.

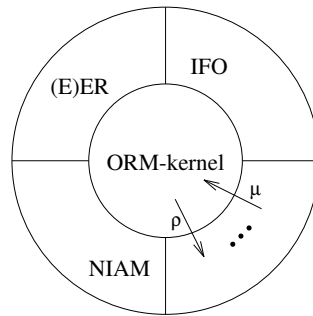


Figure 1: ORM Kernel

A further advantage of using a common ORM kernel is that research results based on this kernel may apply to all variants based on the kernel. In [BHW91] and [Hof93], theoretical results concerning identifiability of object types and populatability of data models are presented, which directly apply to the ORM kernel. Schema evolution of data models conforming to the ORM kernel is treated in [PW94]. Issues regarding internal representations of data models conforming to the ORM kernel have been, and are being, studied in [BW92], [Hal91] and [Hal92]. Finally, in [HPW93] and [HPW94] the query language LISA-D is presented, which allows for the formulation of queries in a semi-natural language format, closely following the naming conventions in the data model. The LISA-D language directly applies for data models conforming to the ORM kernel.

Before introducing the ORM kernel, we first propose a terminological framework to find our way in the methodology jungle. In this article, the framework for information system development methods depicted in figure 2 is used (see also [PW94] and [Pro94]). It makes a distinction between a way of thinking, a way of working, a way of modelling, a way of communicating, and a way of supporting. The framework is based on the framework which was originally presented in [WH90] and [SWS89]. In the framework, an information system development method is dissected into the following aspects:

1. The *way of thinking* verbalises the assumptions and viewpoints of the method on the kinds of problem domains, solutions and modellers. This notion is also referred to as *die Weltanschauung* ([Sol83]), *underlying perspective* ([Mat81]) or *philosophy* ([Avi95]).
2. The *way of working* structures the way in which an information system is developed. It defines the possible tasks, including sub-tasks, and orderings of tasks, to be performed as part of the development process. It furthermore provides guidelines and suggestions (heuristics) on how these tasks should be performed.
3. The *way of modelling* provides an abstract description of the underlying modelling concepts together with their interrelationships and properties. It structures the models which can be used in the information system development, i.e. it provides a language in which to express the models.

4. The *way of controlling* deals with managerial aspects of information system development. It includes such aspects as human resource management, quality and progress control, and evaluation of plans, i.e. overall project management (see [Ken84] and [Sol88]).
5. The *way of supporting* of a method, refers to the support of the method by (possibly automated) tools. A generally accepted name for computer based tools for methods is: *CASE-tools*, see for instance [McC89].
6. The *way of communicating* describes how the abstract notions from the *way of modelling* are visualised (communicated) to human beings, for instance in the style of a conceptual language (such as LISA-D [HPW93]). Usually, the way of communicating provides a graphical notation. It may very well be the case that different methods are based on the same *way of modelling*, and yet use a different graphical notation.

The arrows in figure 2 should be interpreted as: aspect x supports aspect y .

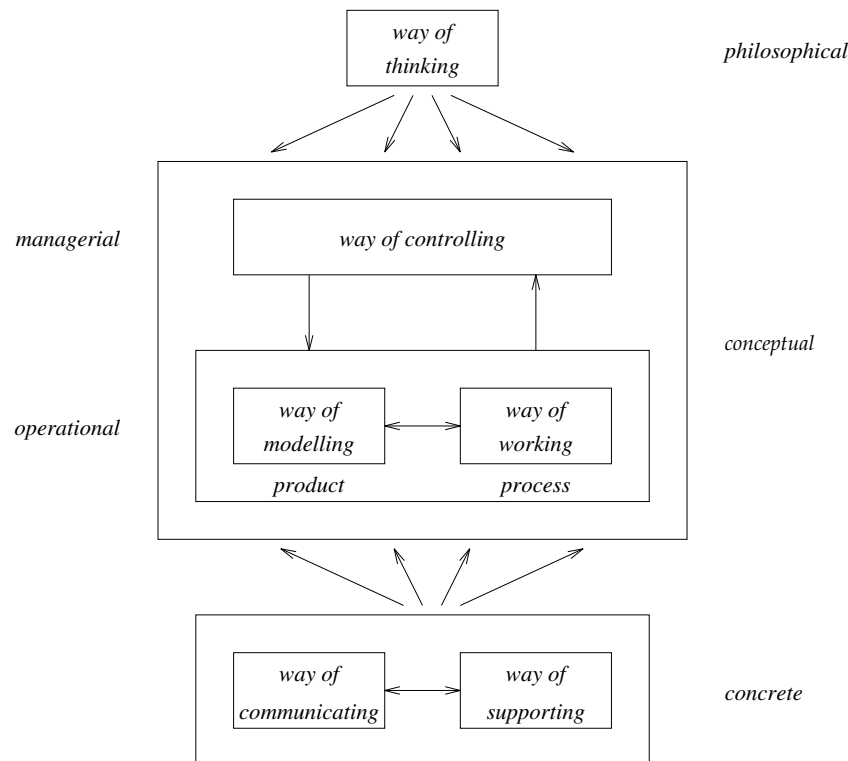


Figure 2: Framework for methodologies

The combination of a way of modelling and a way of communicating is usually referred to as a modelling technique. Several attempts to compare modelling methods and techniques with each other have been made. These attempts usually only *compare* the underlying way of thinking, the way of communicating, and the way of working. In this article, we try to take a more constructive approach, and define a common (kernel of a) way of modelling, which can then be used to define different ways of communicating and working. It is our belief that the basis of an information modelling method or technique is formed by its way of modelling.

To illustrate the elegance of the ORM kernel concept, we present the ORM kernel first, and then define an (E)ER ([Che76], [EGH⁺92]) and a NIAM ([NH89]) way of communicating on top of it as case studies. The formalisation of the ORM kernel proposed in this paper is inspired by the formalisation of PSM ([HW93], [HPW93]) and EVORM ([PW94]). We do not yet claim that the presented ORM kernel is general enough

to cover all different aspects of object role modelling techniques; nevertheless, the (E)ER and NIAM case studies already provide some empirical prove of the generality of this kernel. We also include a discussion on how the ORM kernel may be refined for less general variations of Object-Role Modelling.

The structure of this article is as follows. In section 2 we present the syntactical aspects of the ORM kernel, i.e. what is a model. Semantical issues of ORM models are addressed in section 3, i.e. what does a model mean. Some examples of how to tune the ORM kernel to different ORM (and ER) versions are provided in section 4. Sections 5 and 6 then provide the ways of communicating for (E)ER and NIAM respectively, together with an elaborated example.

2 Syntactical aspects of the ORM Kernel

In this section, the syntactical issues of the ORM kernel are addressed, i.e. what is a proper ORM data model.

2.1 Abstract and concrete object types

In data modelling there exists a distinction between objects that can be represented directly and objects that cannot be represented directly. This corresponds to the difference between label (or value) types and non-label types. Labels can be represented directly on a communication medium, while other objects depend on labels for their representation. As a result, label types are also called *concrete* object types, as opposed to the other object types which are referred to as *abstract* object types. Although these terms have been well accepted in literature ([BHW91], [HW93] and [HPW93]) we realise that some communities rather prefer the usage of other terms, such as *lexical* and *non-lexical*.

The abstract and concrete object types are captured formally by two disjunct, nonempty, sets \mathcal{L} and \mathcal{N} of object types. Together ($\mathcal{O} \triangleq \mathcal{L} \cup \mathcal{N}$), they form the set of all *object types*.

2.2 Atomic types

A special class of types are the types without any (direct!) underlying structure. These types are the atomic types: $\mathcal{A} \subseteq \mathcal{O}$; They are not decomposable in other object types. The abstract atomic object types ($\mathcal{E} \triangleq \mathcal{N} \cap \mathcal{A}$) are referred to as entity types.

2.3 Fact typing

One of the key concepts in data modelling is the concept of fact type (or relationship type).” In this article we treat the terms relationship types and fact types as synonyms in this article. We do realise, however, that sometimes the term *fact type* is reserved for a special class of relationship type.

Generally, a fact type is considered to represent an association between object types. A fact type consists of a number of roles denoting the way object types participate in that fact type. The connection between an object type and a role is called a predicator. In the ORM kernel, a fact type is identified by a *set* of predicators. A fact type is therefore considered to be an association between predicators, rather than between object types. A fact type may be treated as an object type (*fact objectification*), and can therefore play a role in other fact types.

Formally, the set of predicators is provided as \mathcal{P} , and the set of fact types as a partition \mathcal{F} of the set \mathcal{P} . The auxiliary function $\text{Fact} : \mathcal{P} \rightarrow \mathcal{F}$ yields the fact type in which a given predicator is contained, and is defined by: $\text{Fact}(p) = f \iff p \in f$. The function $\text{Base} : \mathcal{P} \rightarrow \mathcal{O}$ yields the object type that plays the role of the predicator.

A bridge type is a special fact type, relating the abstract and the concrete worlds. In some ORM variations, they are considered the only way to bridge the gap between concrete and abstract world. The set of bridge types (\mathcal{B}) is identified by:

$$\mathcal{B} = \{ \{p, q\} \in \mathcal{F} \mid p \in \mathcal{P}_{\mathcal{L}} \wedge q \in \mathcal{P}_{\mathcal{N}} \}$$

where $\mathcal{P}_{\mathcal{X}} \triangleq \{p \in \mathcal{P} \mid \text{Base}(p) \in X\}$. In section 4 an axiom is presented which (optionally) enforces the idea that bridge types should be the single way to cross the gap between the concrete and abstract worlds.

The predicates that constitute a bridge type $b = \{p, q\}$ can be extracted by the operators *concr* and *abstr*. These operators are defined by

$$\text{concr}(b) \triangleq p \text{ such that } p \in b \cap \mathcal{P}_{\mathcal{L}} \quad \text{and} \quad \text{abstr}(b) \triangleq q \text{ such that } q \in b \cap \mathcal{P}_{\mathcal{N}}$$

respectively.

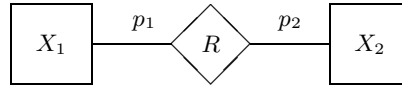


Figure 3: Example fact/relationship type

As an example fact type, consider the ER schema depicted in figure 3. In this schema we have: entity types $\mathcal{E} = \{X_1, X_2\}$, fact types $\mathcal{F} = \{R\}$, and predicates $\mathcal{P} = \{p_1, p_2\}$ where fact type $R = \{p_1, p_2\}$. Note that to provide an example, we already have to choose a more concrete way of communicating, i.e. the ER style.

2.4 Power typing

The concept of *power type* in ORM forms the data modelling pendant of power sets in conventional set theory ([Lev79]). This notion is the same as the notion of grouping introduced in the IFO data model ([AH87]). Power types form a special class of object types ($\mathcal{G} \subseteq \mathcal{O}$). An instance of a power type is a set of instances of its *element type*. Such an instance is identified by its elements, just as a set is identified by its elements in set theory (axiom of extensionality). The element type of a power type is found by the function $\text{Elt} : \mathcal{G} \rightarrow \mathcal{O}$. An example power type is provided in figure 4 in the PSM style of communicating. This example is concerned with convoys, being sets of ships.

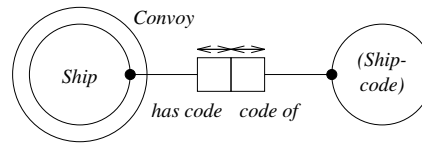


Figure 4: Convoys of ships

2.5 Sequence typing

Sequence typing offers the opportunity to represent sequences, built from an underlying element type. The sequence types are captured as the set $\mathcal{S} \subseteq \mathcal{O}$. Instances of a sequence type thus correspond to sequences of instances from its underlying element type. The element type of a sequence type is found by the function $\text{Elt} : \mathcal{S} \rightarrow \mathcal{O}$. Note that *Elt* is now an overloaded function symbol for a function: $\text{Elt} : \mathcal{S} \cup \mathcal{G} \rightarrow \mathcal{O}$.

2.6 Identification hierarchy

In the ORM kernel, we define the notion of *identification hierarchy*, which generalises generalisation and specialisation. Both generalisation and specialisation are treated as special *flavours* of inheritance in the identification hierarchy, and they will be introduced in the next two subsections.

The identification hierarchy is defined as a partial order (asymmetric and transitive) $\text{ldfBy} \subseteq \mathcal{A} \times \mathcal{O}$ on object types, with the convention that $a \text{ ldfBy } b$ is interpreted as: *a inherits its identification from b*. Note that an identification hierarchy only deals with inheritance of identification via specialisation or generalisation.

The nature of a partial order is expressed by:

[ORM1] (*asymmetry*) $x \text{ ldfBy } y \Rightarrow \neg y \text{ ldfBy } x$

[ORM2] (*transitivity*) $x \text{ ldfBy } y \text{ ldfBy } z \Rightarrow x \text{ ldfBy } z$

Note that $x \text{ ldfBy } y \text{ ldfBy } z$ is an abbreviation of $x \text{ ldfBy } y \wedge y \text{ ldfBy } z$.

We define ldfBy_1 as the one step counterpart of ldfBy :

$$x \text{ ldfBy}_1 y \triangleq x \text{ ldfBy } y \wedge \neg \exists z [x \text{ ldfBy } z \text{ ldfBy } y]$$

In the ORM kernel, all object types in the identification hierarchy have direct ancestors:

[ORM3] (*direct ancestors*) $x \text{ ldfBy } y \Rightarrow x \text{ ldfBy}_1 y \vee \exists p [x \text{ ldfBy}_1 p \text{ ldfBy } y]$

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

[ORM4] (*identification induction*) If F is a property for object types, such that:

$$\forall x: y \text{ ldfBy}_1 x [F(x)] \Rightarrow F(y) \text{ for any } y$$

then $\forall x \in \mathcal{O} [F(x)]$

The identification hierarchy is a result of specialisation and generalisation:

[ORM5] (*complete span*)

$$x \text{ ldfBy}_1 y \Rightarrow x \text{ Gen } y \vee x \text{ Spec } y$$

$$x \text{ Gen } y \vee x \text{ Spec } y \Rightarrow x \text{ ldfBy } y$$

2.7 Specialisation

Specialisation is a mechanism for representing one or more (possibly overlapping) subtypes of an object type. Specialisation is to be applied when only for specific instances of an object type certain facts are to be recorded.

A specialisation relation between a subtype and a supertype implies that the instances of the subtype are also instances of the supertype. For proper specialisation, it is required that subtypes be defined in terms of one or more of their supertypes. Such a decision criterion is referred to as *subtype defining rule* ([BHW91]), and can be specified by means of a LISA-D expression ([HPW93]). Identification of subtypes is derived from their supertypes. As an example specialisation consider figure 5. Possible subtype defining rules for *CD* and *Record* in the example are:

$$\begin{aligned} \text{CD} &= \text{Medium having Type 'CD'} \\ \text{Record} &= \text{Medium having Type 'Record'} \end{aligned}$$

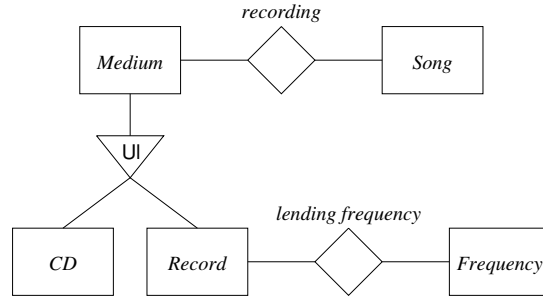


Figure 5: An example of specialisation

Objects inherit all properties from their ancestors in the specialisation hierarchy. This characteristic of specialisation excludes non-entity types (e.g. fact types) occurring as subtypes. Consider for example the case that a ternary fact type is a subtype of a binary fact type. Clearly this leads to a contradiction. No problems occur when non-entity types themselves are specialised. Consequently, non-entity types always act as pater familias. For an in depth discussion of specialisation, we refer to [HHO92].

The concept of specialisation is modelled as a partial order (asymmetric and transitive) Spec on object types, such that Spec is a part of the identification hierarchy. The intuition behind $a \text{ Spec } b$ is: a is a specialisation of b , or a is a subtype of b .

[ORM6] (*transitivity completeness*) If $x \text{ ldfBy } y \text{ ldfBy } z$ then:

$$x \text{ Spec } y \text{ Spec } z \iff x \text{ Spec } z$$

Note that the asymmetry of Spec follows from the asymmetry of ldfBy , as $\text{Spec} \subseteq \text{ldfBy}$.

Since the identification hierarchy, and thus the specialisation hierarchy, is non-cyclic and finite, roots can be identified. For specialisation, these roots are referred to as *pater familias* (see [DMV88]), and are identified by: $\sqcap(x, y) \triangleq (x \text{ Spec } y \vee x = y) \wedge \neg \text{spec}(y)$ where $\text{spec}(x)$ is a shorthand for $\exists y [x \text{ Spec } y]$. Each specialisation hierarchy, contrary to generalisation, has a unique top element. This is stipulated by the following axiom:

[ORM7] (*unique pater familias*) $\sqcap(x, y) \wedge \sqcap(x, z) \Rightarrow y = z$

This axiom allows us to regard the pater familias relation \sqcap as a partial function, and to write $\sqcap(x) = y$ instead of $\sqcap(x, y)$.

2.8 Generalisation

Generalisation is a mechanism that allows for the creation of new object types by uniting existing object types. Generalisation is to be applied when different object types play identical roles in fact types. Contrary to what its name suggests, generalisation is *not* the inverse of specialisation. Specialisation and generalisation originate from different axioms in set theory ([HW93]) and therefore have a different expressive power. Specialisation is based on set comprehension ($\{x \in S \mid P(x)\}$ where P is the subtype defining rule), whereas generalisation corresponds to set union ($X \cup Y$).

An example of the use of a generalisation is provided in figure 6. This example is taken from [HP95]. In this schema, the type *fragment* is either a *graphic* or it is a sequence (a *paragraph*) of *sentences*.

For generalisation it typically is required that the generalised object type is covered by its constituent object types (or *specifiers*), i.e. the population of the generalisation equals the union of its specifiers. Therefore,

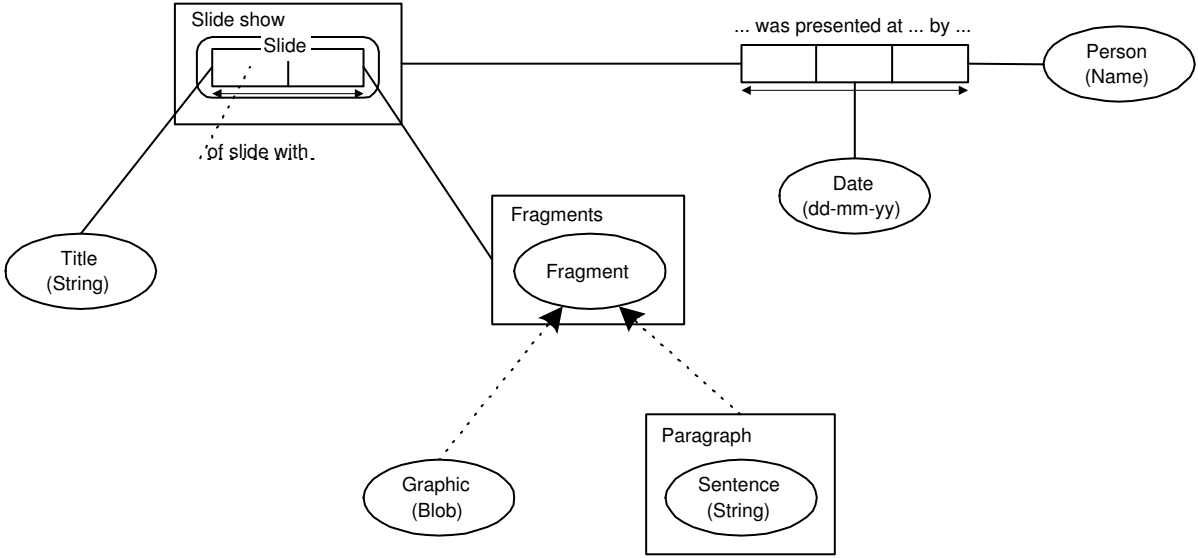


Figure 6: An example of generalisation

a decision criterion as in the case of specialisation (the subtype defining rule) is not necessary. Generalisation is typically only used when common facts need to be stored for types with different (polymorphic) underlying types. In [HP95] this is enforced by an explicit axiom (in section 4 we provide this axiom as an illustration of a refinement of the kernel). Even more, in [EN94] it is argued that a total generalisation (there called categorisation) can just as well be modelled using a specialisation. However, in [HP95] it is argued that this only makes sense if the underlying specifiers of the generalisation have the same underlying structure (*not* polymorph).

A further important difference between generalisation and specialisation is the fact that properties are inherited “upward” in a generalisation hierarchy instead of “downward”, which is the case for specialisation (see also [AH87]). This also implies that the identification of a generalised object type depends on the identification of its specifiers. From the nature of generalisation, it is apparent that a non-entity type cannot be a generalised object type.

The concept of generalisation is introduced as a partial order Gen . The expression $a \text{ Gen } b$ stands for: a is a generalisation of b , or b is a specifier of a . In the sequel $\text{gen}(x)$ will be used as an abbreviation for $\exists y [x \text{ Gen } y]$.

[ORM8] (*transitivity completeness*) If $x \text{ ldfBy } y \text{ ldfBy } z$ then:

$$x \text{ Gen } y \text{ Gen } z \iff x \text{ Gen } z$$

Generalisation and specialisation can be conflicting due to their inheritance structure. To avoid such conflicts, generalised object types are required to be pater familias:

[ORM9] $\text{gen}(x) \Rightarrow \neg \text{spec}(x)$

Basic specifiers of a generalised object type are defined analogously to the pater familias of a specialised object type: $\sqcup(x, y) \triangleq (x \text{ Gen } y \vee x = y) \wedge \neg \text{gen}(y)$. Note that uniqueness of basic specifiers is not required. As a shorthand, we will write $\sqcup(x)$ for the set $\{y \mid \sqcup(x, y)\}$.

2.9 Schema typing

A schema type is an object type with an underlying decomposition. The concept of *schema typing* allows for the decomposition of large schemata into, objectified, subschemata. The need for such a mechanism

has been generally recognised. In figure 7, an example of a schema type concerned with activity graphs and their decomposition is given. Note that the encircled dot indicates a so-called disjunctive total role

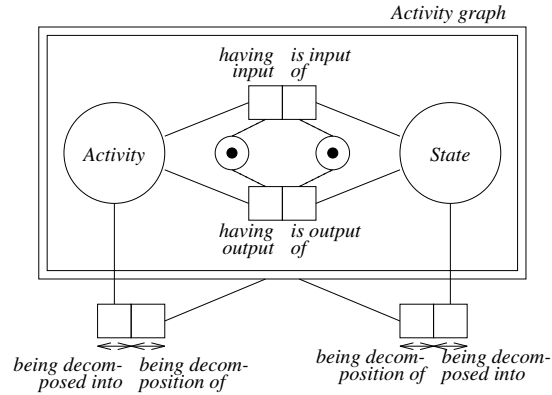


Figure 7: Meta schema of activity graphs

constraints. In the example, the left most total role constraint implies that each activity plays the *having input* or *having output* role (or both).

The set of schema types is presumed to be provided by $\mathcal{C} \subseteq \mathcal{O}$. Schema types can be decomposed into their underlying information structure via the relation $\prec \subseteq \mathcal{C} \times \mathcal{O}$, with the convention that $x \prec y$ is interpreted as x is decomposed into y or y is part of the decomposition of x . This underlying information structure \mathcal{I}_x for a schema type x is derived from the object types into which x is decomposed: $\mathcal{O}_x = \{y \in \mathcal{O} \mid x \prec y\}$. Analogously, the special object classes $\mathcal{F}_x, \mathcal{G}_x, \mathcal{S}_x, \mathcal{C}_x$ and \mathcal{A}_x can be derived. The functions $\text{Base}_x, \text{Elt}_x, \prec_x, \text{Spec}_x, \sqcap_x$ and Gen_x are obtained by restriction to object types within \mathcal{O}_x . In order to be a proper decomposition, the underlying information structure should form an information structure on its own:

[ORM10] (*structural nesting*) $x \in \mathcal{C} \Rightarrow \mathcal{I}_x$ is an ORM information structure.

The strict separation between concrete and abstract object types also goes for schema types. A schema type can either be a label type or a non-label type. For a schema type, which is a label type, this means that all the object types in the decomposition of it are also label types. An example of this is a date. A date can be regarded as a combination of a year, a month and a day, which can all be defined as label types.

2.10 Additional properties

Due to the different interpretation that will be given to atomic types, fact types, power types, sequence types and schema types, these object types are all considered to be different concepts:

[ORM11] (*disjunction*) $\mathcal{A}, \mathcal{F}, \mathcal{G}, \mathcal{S}$ and \mathcal{C} form a partition of \mathcal{O}

It is interesting to see that label types do not necessarily have to be atomic. If desired, however, an appropriate axiom can be added requiring label types to be atomic.

The concrete and the abstract world may not be mixed; therefore the following rule should hold:

[ORM12] (*strict separation*) $x = \text{Elt}(y) \vee x \prec y \vee x \text{ ldfBy } y \Rightarrow x, y \in \mathcal{L} \vee x, y \in \mathcal{N}$

The gap between the concrete and abstract worlds may only be bridged using fact types. In some versions of ORM this is even further limited to the bridge types only.

Even though it can be argued that power typing, sequence typing, and schema typing, are not elementary concepts ([HW93]) it is convenient to have them explicitly present in the kernel; since this allows for a more elegant formal treatment of these concepts.

Finally, we realise that the terminology used in this paper may not be agreed upon by everyone. In the field of conceptual modelling there is still an abundance of terminological disputes and differences. For instance, alternative terms for our notion of generalisation exist, such as: category and polymorphism. In [HP95] a terminological framework for ORM based modelling techniques is proposed as part of ongoing research between the authors of FORM ([HO92]) and PSM ([HW93]) variations of ORM. As such, this article, as well as [HP95], are all steps in the direction of a unifying Object-Role Modelling Theory.

3 Semantical aspects of the ORM kernel

In this section we first provide the semantics of data models in the ORM kernel. We distinguish two sorts of semantics. The first sort of semantics deals with the interpretation of both the user and information analyst. Thus far, object types and predicators in the ORM kernel are abstract concepts. We propose a naming mechanism for these abstract concepts, providing a means for human interpretation. The second sort of semantics is concerned with populations of data models in the ORM kernel, and constraints defined over these populations.

After providing the semantics of the data models, we provide some example axioms to ‘tune’ the ORM kernel to concrete data modelling techniques.

3.1 Naming of Concepts

In this article we define two classes of names for the abstract concepts in the ORM kernel. Object types, and combinations of predicators, may also receive a name. The set `Names` is used for all names that can be found in an information structure.

Object types are referenced by a unique name: $ONm : \mathcal{O} \rightarrow \text{Names}$, which is specified in the schema upon their introduction. The (partial) function $Obj : \text{Names} \rightarrow \mathcal{O}$ is the left-inverse of ONm , and relates object type names to their corresponding object type:

$$\forall_{x \in \text{dom}(ONm)} [Obj(ONm(x)) = x]$$

where $\text{dom}(ONm)$ denotes the domain of function ONm .

Besides naming of objects, ORM also allows naming of pairs of (different) predicators by means of *connector names*. The function combining such pairs with names is called $Connector : \mathcal{P} \times \mathcal{P} \rightarrow \text{Names}$. Names can only be given to pairs of predicators which are part of one single fact type:

$$Connector(p, q) \downarrow \Rightarrow p \in \text{Fact}(q)$$

Note that in some interpretations, *connector names* may correspond to the notion of role name. In figure 3, we could for instance have the following names:

$$\begin{array}{ll} ONm(X_1) = \text{Department} & Connector(p_1, p_2) = \text{has as coworker the} \\ ONm(X_2) = \text{Employee} & Connector(p_2, p_1) = \text{is a coworker of the} \end{array}$$

3.2 Constraints and Semantics

An information structure is used as a model for some part of the (real) world, the so-called universe of discourse ($U \circ D$). A *state* of the $U \circ D$ then corresponds to a so-called instantiation or population of the information structure, and vice versa.

In the ORM kernel, a population Pop of an information structure \mathcal{I} is a value assignment of sets of instances to the object types in \mathcal{O} , satisfying the population (POP) rules that follow in the remainder of this section. This is denoted as $\text{IsPop}(\mathcal{I}, \text{Pop})$. Pop then is a mapping $\text{Pop} : \mathcal{O} \rightarrow \wp(\Omega)$, where Ω is the universe of instances that can occur in the population of an information structure \mathcal{I} . This universe of instances is defined in definition 3.1.

An information structure can only be populated if a link is established between label types and concrete domains. The instances of label types come from their associated concrete domain. Formally, this link is established by the function $\text{Dom} : \mathcal{L} \cap \mathcal{A} \rightarrow D$. The range of this function, i.e. D , is the set of concrete domains (e.g. string, natno). The sets in D form the carriers of a many sorted algebra $\mathcal{D} = \langle D, F \rangle$, where F is the set of operations (e.g. +) on the sorts in D .

Definition 3.1

The universe of instances Ω is inductively defined as the smallest set satisfying:

1. $\bigcup D \subseteq \Omega$. Instances from the sorts in the many sorted algebra are elements of the universe of instances.
2. $\Theta \subseteq \Omega$, where Θ is an abstract (countable) domain of (unstructured) values that may occur in the population of entity types.
3. $x_1, \dots, x_n \in \Omega \wedge p_1, \dots, p_n \in \mathcal{P} \Rightarrow \{p_1 : x_1, \dots, p_n : x_n\} \in \Omega$. The set $\{p_1 : x_1, \dots, p_n : x_n\}$ denotes a mapping, assigning x_i to each predicator p_i . These mappings are intended for the population of fact types.
4. $x_1, \dots, x_n \in \Omega \Rightarrow \{x_1, \dots, x_n\} \in \Omega$. Sets of instances may occur as instances of power types.
5. $x_1, \dots, x_n \in \Omega \Rightarrow \langle x_1, \dots, x_n \rangle \in \Omega$. Sequences of instances are used as instances of sequence types (see the sequence type rule). The i -th element of a sequence $\langle x_1, \dots, x_n \rangle$, i.e. x_i , can be derived using projection, denoted as: $\langle x_1, \dots, x_n \rangle_{<i>$.
6. $X_1, \dots, X_n \subseteq \Omega \wedge O_1, \dots, O_n \in \mathcal{O} \Rightarrow \{O_1 : X_1, \dots, O_n : X_n\} \in \Omega$. Assignments of sets of instances to object types are also valid instances. They are intended for the populations of composition types.

□

For populations, certain conditions must hold. Some of these conditions are given below, for a complete set of conditions, refer to [HPW93]. *Root object types* are object types that are neither generalised, nor specialised. This is formalised as: $\text{Root}(x) \triangleq \neg \text{gen}(x) \wedge \neg \text{spec}(x)$. The population of a (root) label type is a set of values taken from its corresponding concrete domain:

$$\text{[POP1]} \quad x \in \mathcal{L} \wedge \text{Root}(x) \Rightarrow \text{Pop}(x) \subseteq \text{Dom}(x)$$

Note that $\text{Root}(x) \Rightarrow x \in \mathcal{A}$. The population of root entity types is a set of values, taken from the abstract domain Θ .

$$\text{[POP2]} \quad x \in \mathcal{E} \wedge \text{Root}(x) \Rightarrow \text{Pop}(x) \subseteq \Theta$$

The population of a fact type is a set of tuples. A tuple y in the population of a fact type x is a mapping of all its predicators to values of the appropriate type. This is referred to as the *conformity rule*:

$$\text{[POP3]} \quad x \in \mathcal{F} \wedge y \in \text{Pop}(x) \Rightarrow y : x \rightarrow \Omega \wedge \forall_{p \in x} [y(p) \in \text{Pop}(\text{Base}(p))]$$

The population of a power type consists of (nonempty) sets of instances of the corresponding element type. This is called the *power type rule*:

[POP4] $x \in \mathcal{G} \wedge y \in \text{Pop}(x) \Rightarrow y \in \wp(\text{Pop}(\text{Elt}(x))) - \{\emptyset\}$

The population of a sequence type consists of (nonempty) sequences of instances of the corresponding element type. This is called the *sequence type rule*:

[POP5] $x \in \mathcal{S} \wedge y \in \text{Pop}(s) \Rightarrow y \in \text{Pop}(\text{Elt}(x))^+$

The population of a composition type consists of populations of the underlying information structure.

[POP6] $x \in \mathcal{C} \wedge y \in \text{Pop}(x) \Rightarrow \text{IsPop}(\mathcal{I}_x, y)$

Respecting the identification hierarchy is reflected by the following rule:

[POP7] If $\text{gen}(x)$ or $\text{spec}(x)$, then:

$$\text{Pop}(x) \subseteq \bigcup_{y: x \text{ ldfBy } y} \text{Pop}(y)$$

Instances of a generalised or specialised object type originate from object types ‘higher’ in the identification hierarchy. In generalisation, all instances of a specifier are also instances of the generalised object type, contrary to specialisation where this is governed by the subtype defining rule.

[POP8] $x \text{ Gen } y \Rightarrow \text{Pop}(y) \subseteq \text{Pop}(x)$

In the UoD restrictions may have to be enforced on the populations. For example, a person can be stated to have at least one name. In some contexts, it can be stated that a person has a unique name. Such restrictions are called *constraints*, forming the set \mathcal{R} . In this section we define three classes of constraints, which are usually modelled graphically in data models. For more (complex) constraints see [HW92]. The first class of constraints is the *uniqueness constraint*. This class of constraints states that the population of the predicator (p), which it is enforced upon, may not have multiple occurrences of same instances:

$$\text{unique}(p) \triangleq \forall_{x,y \in \text{Pop}(\text{Fact}(p))} [x(p) = y(p) \Leftrightarrow x = y]$$

Another class of constraints is the *total role constraint*. Such a constraint states that all instances of the base of the predicator (p) must participate in the population of the fact of that predicator:

$$\text{total}(p) \triangleq \forall_{x \in \text{Pop}(\text{Base}(p))} \exists_{y \in \text{Pop}(\text{Fact}(p))} [y(p) = x]$$

The last class of constraints we define, is the *exclusion constraint*. An exclusion constraint states that the population of different object types may not have any instance in common:

$$\text{exclusion}(\{x_1, \dots, x_n\}) \triangleq \forall_{1 \leq i < j \leq n} [\text{Pop}(x_i) \cap \text{Pop}(x_j) = \emptyset]$$

4 Tuning the Kernel

Besides the POP and ORM axioms for the ORM kernel, more specific axioms can be formulated for models in ER and NIAM like modelling techniques. This allows us to tune the kernel for different versions of ER based techniques or ORM based techniques. Some of such extra rules are discussed in this section. We do not intend to be elaborate, but rather try to give an idea of such extra rules. The explicit identification of such rules should allow for a classification of ER and ORM based modelling techniques quite similar to the classification of epistemic logic systems such as the S4, S5 and K45 modal systems (see e.g. [Luk91]).

4.1 ER

The ER model presented in [Che76], does not deal with constructs like power typing, sequence typing and schema typing. This results in the following restriction:

$$\text{[ER1]} \text{ (no complex types)} \quad \mathcal{G} \cup \mathcal{S} \cup \mathcal{C} = \emptyset$$

Furthermore, pure ER does not support complex label types:

$$\text{[ER2]} \text{ (no complex labels)} \quad \mathcal{P}_{\mathcal{L}} \subseteq \cup \mathcal{B}$$

In pure ER it is also forbidden to use objectification:

$$\text{[ER3]} \text{ (no objectification)} \quad \text{ran}(\text{Base}) \cap \mathcal{F} = \emptyset$$

The ran in the above axiom returns the range of a function. Finally, pure ER does not allow for generalisation and specialisation of object types:

$$\text{[ER4]} \text{ (No inheritance)} \quad \text{ldfBy} = \emptyset$$

Extended versions of ER (see for example [EN94] or [EGH⁺92]) typically do allow for some of the constructs excluded by the above 4 extra axioms.

4.2 ORM

The ORM variation presented in [NH89] (one of the NIAM variations), does not deal with power typing, sequence typing and schema typing either, so we would have:

$$\text{[NIAM1]} \text{ (no complex types)} \quad \mathcal{G} \cup \mathcal{S} \cup \mathcal{C} = \emptyset$$

Pure NIAM also does not support complex label types:

$$\text{[NIAM2]} \text{ (no complex labels)} \quad \mathcal{P}_{\mathcal{L}} \subseteq \cup \mathcal{B}$$

Finally, pure NIAM does not allow for generalisation of object types:

$$\text{[NIAM3]} \text{ (no generalisation)} \quad \text{Gen} = \emptyset$$

The ORM version presented in [SZ91] differs from pure NIAM in the fact that it only allows for binary fact types:

$$\text{[NIAM4]} \quad f \in \mathcal{F} \Rightarrow |f| = 2$$

In the NIAM version adopted in [Win90], bridge types, are the only way to cross the gap between the concrete and abstract worlds:

$$\text{[NIAM5]} \text{ (bridges only)} \quad f \in \mathcal{F} \Rightarrow f \subseteq \mathcal{P}_{\mathcal{L}} \vee f \subseteq \mathcal{P}_{\mathcal{N}} \vee f \in \mathcal{B}$$

In [BZL94], a so called “fully communication oriented” variation of NIAM is proposed in which no entity types are allowed. Each “entity type” should rather be an objectification of a fact type representing its identification (its way of communicating).

Finally, the following rule ([HP95]) allows us to limit the use of generalisation to the cases where it is really required due to polymorph specifiers:

$$\text{[NIAM6]} \text{ (polymorph generalisation)} \quad \text{gen}(x) \Rightarrow \exists_{y \in \mathcal{O} - \mathcal{A}} [x \text{ Gen } y]$$

5 ER way of communicating

When discussing the ER way of communicating, we actually refer to a group of modelling techniques. In this article we will use the name *pure ER* for the modelling technique which was introduced in [Che76]. Taking this technique as a base, we will define ER^+ by adding new symbols, part of which are taken from the EER technique as presented in [EGH⁺92].

5.1 Running example

Before discussing the ER way of communicating, we introduce a running example to illustrate this way of communicating. Now that we have defined the concepts in the ORM kernel, the naming of these concepts, the populations and constraints on the populations, we are able to use them in an example.

We employ the example of a zoo as a running example in the remainder of this article. Consider a company owning several zoo's in different cities. A zoo houses a number of animals (in this case only mammals and reptiles), and employs people to run the zoo. The persons working for the zoo each have a contract for a number of hours a week. Since the zoo is obliged to pay an environmental-tax for the amount of biological waste it produces, the zoo wants to know the amount (in kilos) of manure each mammal produces. For reptiles it is essential that their living environment is kept at a specific temperature, therefore, for each reptile this temperature is recorded. Each animal is accommodated in a home (together with other animals). Furthermore, each animal has an owner which is either a person or a zoo. Finally, all kinds of animals need to be fed at certain times, therefore a feeding table needs to be maintained.

If you consider the way of communicating we use to present the example UoD to be unreadable, and incomprehensible, we could not agree with you more! The readability and comprehensibility is exactly the reason why a graphical way of communicating for data models is to be preferred. For a preview on the resulting graphical models, the reader is advised to look at figure 8 and figure 9.

The example UoD can now be formulated in terms of the concepts of the ORM kernel. The object types in this example are, in abstract notation:

$$\begin{aligned} \mathcal{A} &= \{x_1, \dots, x_{20}\} & \mathcal{P} &= \{p_1, \dots, p_{32}\} \\ \mathcal{F} &= \{\{p_1, p_2\}, \dots, \{p_{31}, p_{32}\}\} & \mathcal{G} &= \{g\} \\ \mathcal{S} &= \{s\} & \mathcal{C} &= \{c\} \end{aligned}$$

Note that, in this example, all fact types are binary.

The object types in the example, which receive a name are:

ONm(x_1) = Name	ONm(x_2) = F id	ONm(x_3) = P id
ONm(x_4) = # Hours	ONm(x_5) = C id	ONm(x_6) = Kg
ONm(x_7) = °C	ONm(x_8) = Function	ONm(x_9) = Person
ONm(x_{10}) = Zoo keeper	ONm(x_{11}) = Home	ONm(x_{12}) = Time
ONm(x_{13}) = City	ONm(x_{14}) = Kind of animal	ONm(x_{15}) = Animal
ONm(x_{16}) = Manure	ONm(x_{17}) = Mammal	ONm(x_{18}) = Reptile
ONm(x_{19}) = Temperature	ONm(x_{20}) = Owner	ONm(g) = Animals
ONm(s) = Feeding table	ONm(c) = Zoo	ONm($\{p_1, p_2\}$) = Contract

The division in label types and non-label types is given by:

$$\begin{aligned} \mathcal{L} &= \{\text{Obj}(\text{Name}), \text{Obj}(\text{F id}), \text{Obj}(\text{P id}), \text{Obj}(\text{\# Hours}), \text{Obj}(\text{C id}), \text{Obj}(\text{Kg}), \text{Obj}(\text{°C})\} \\ \mathcal{N} &= \{\text{Obj}(\text{Function}), \text{Obj}(\text{Person}), \text{Obj}(\text{Zoo keeper}), \text{Obj}(\text{Home}), \text{Obj}(\text{Time}), \text{Obj}(\text{Kind of animal}), \\ &\quad \text{Obj}(\text{City}), \text{Obj}(\text{Animal}), \text{Obj}(\text{Manure}), \text{Obj}(\text{Mammal}), \text{Obj}(\text{Reptile}), \text{Obj}(\text{Temperature}), \\ &\quad \text{Obj}(\text{Animals}), \text{Obj}(\text{Feeding table}), \text{Obj}(\text{Zoo}), \text{Obj}(\text{Contract}), \{p_3, p_4\}, \dots, \{p_{31}, p_{32}\}\} \end{aligned}$$

The bases of the predicates are:

Base(p_1) = Obj(Function)	Base(p_2) = Obj(Person)
Base(p_3) = Obj(Person)	Base(p_4) = Obj(Name)
Base(p_5) = Obj(Contract)	Base(p_6) = Obj(# Hours)
Base(p_7) = Obj(Zoo keeper)	Base(p_8) = Obj(Kind of animal)
Base(p_9) = Obj(Home)	Base(p_{10}) = Obj(Animals)
Base(p_{11}) = Obj(Zoo)	Base(p_{12}) = Obj(City)
Base(p_{13}) = Obj(Feeding table)	Base(p_{14}) = Obj(Kind of animal)
Base(p_{15}) = Obj(Kind of animal)	Base(p_{16}) = Obj(Animal)
Base(p_{17}) = Obj(Manure)	Base(p_{18}) = Obj(Mammal)
Base(p_{19}) = Obj(Reptile)	Base(p_{20}) = Obj(Temperature)
Base(p_{21}) = Obj(Owner)	Base(p_{22}) = Obj(Animal)
Base(p_{23}) = Obj(Function)	Base(p_{24}) = Obj(F id)
Base(p_{25}) = Obj(Person)	Base(p_{26}) = Obj(P id)
Base(p_{27}) = Obj(Manure)	Base(p_{28}) = Obj(Kg)
Base(p_{29}) = Obj(Temperature)	Base(p_{30}) = Obj($^{\circ}$ C)
Base(p_{31}) = Obj(Animal)	Base(p_{32}) = Obj(Kind)

The element types of the Feeding table and Animals object types are:

$$\text{Elt}(\text{Obj}(\text{Animals})) = \text{Obj}(\text{Animal}) \text{ and } \text{Elt}(\text{Obj}(\text{Feeding table})) = \text{Obj}(\text{Time})$$

Almost all object types are in the decomposition of schema type Zoo:

$$\forall o \in \mathcal{O} - \{\text{Obj}(\text{Zoo}), \{p_{11}, p_{12}\}, \text{Obj}(\text{City}), \text{Obj}(\text{C id})\} [\text{Obj}(\text{Zoo}) \prec o]$$

In the example the following connectors are used:

Connector(p_1, p_2) = is done by	Connector(p_2, p_1) = works as
Connector(p_3, p_4) = has as	Connector(p_4, p_3) = is name of
Connector(p_5, p_6) = for	Connector(p_6, p_5) = belonging to
Connector(p_7, p_8) = takes care of	Connector(p_8, p_7) = is taken care by
Connector(p_9, p_{10}) = inhabits	Connector(p_{10}, p_9) = are inhabitants of
Connector(p_{11}, p_{12}) = is located in	Connector(p_{12}, p_{11}) = has as zoo
Connector(p_{13}, p_{14}) = contains times to feed	Connector(p_{14}, p_{13}) = must be fed at
Connector(p_{15}, p_{16}) = has as animal	Connector(p_{16}, p_{15}) = is kind of
Connector(p_{17}, p_{18}) = is produced by	Connector(p_{18}, p_{17}) = produces
Connector(p_{19}, p_{20}) = likes	Connector(p_{20}, p_{19}) = is favorite of
Connector(p_{21}, p_{22}) = owns	

The information structure contains three specialisations and two generalisations:

$$\text{Obj}(\text{Zoo keeper}) \text{ Spec } \text{Obj}(\text{Person})$$

$$\text{Obj}(\text{Kind of Animal}) \text{ Spec } \text{Obj}(\text{Mammal}) \text{ and } \text{Obj}(\text{Kind of Animal}) \text{ Spec } \text{Obj}(\text{Reptile})$$

$$\text{Obj}(\text{Owner}) \text{ Gen } \text{Obj}(\text{Person}) \text{ and } \text{Obj}(\text{Owner}) \text{ Gen } \text{Obj}(\text{Zoo})$$

Finally, in the structure there are a number of constraints:

For each x in the following set we have $\text{unique}(x)$:

$$p_3, p_5, p_{14}, p_{16}, p_{19}, p_{22}, \dots, p_{32}$$

For each x in the following set we have $\text{total}(x)$:

$$p_3, \dots, p_8, p_{10}, \dots, p_{14}, p_{16}, p_{17}, p_{19}, p_{20}, p_{23}, p_{25}, p_{27}, p_{29}, p_{31}$$

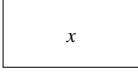

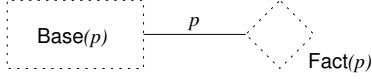
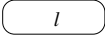
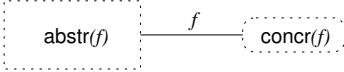
construct	symbol
$x \in \mathcal{A} \cap \mathcal{N}$	
$f \in \mathcal{F} - \mathcal{F}_O$	
$p \in \mathcal{P}_N - \cup \mathcal{B}$	
$x \in \mathcal{A} \cap \mathcal{L}$	
$f \in \mathcal{B}$	

Table 1: Relation between ORM constructs and ER symbols

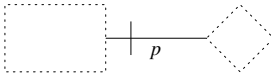
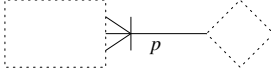
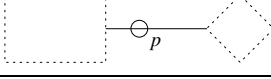
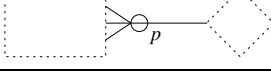
construct	symbol
$\text{unique}(p) \wedge \text{total}(p)$	
$\neg \text{unique}(p) \wedge \text{total}(p)$	
$\text{unique}(p) \wedge \neg \text{total}(p)$	
$\neg \text{unique}(p) \wedge \neg \text{total}(p)$	

Table 2: Constraints

5.2 Pure ER

As stated before, pure ER stands for the technique introduced in [Che76]. In this technique the following ORM constructs can be identified: entity types, fact types, predicates, label types and bridge types. In table 1 these constructs and their pure ER representation are given. For instance, an atomic non-label type (entity type) is represented as a rectangle, and an bridge type (attribute type) is represented by a line between the abstract part and the concrete part of the bridge type.

Although [Che76] does not give a graphical notation for attributes (bridge types), we will consider the notation shown in table 1 to be pure ER. The set \mathcal{F}_O corresponds to the objectified fact types, and is defined by:

$$\mathcal{F}_O \triangleq \text{ran}(\text{Base}) \cap \mathcal{F}$$

where ran returns the range of a function. Note that in pure ER: $\mathcal{P}_L \subseteq \cup \mathcal{B}$.

In table 2, the ER-symbols for uniqueness and total role constraints are given. The notation used for uniqueness constraints is based on the well known chicken feet notation. In the remainder, a predictor drawn in ER style without any constraints, is optional and not unique.


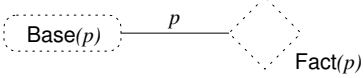
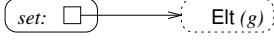
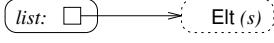
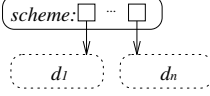
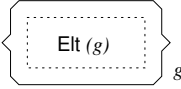
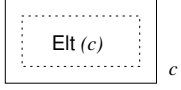
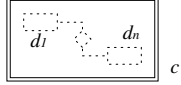
construct	symbol
$f \in \mathcal{F}_O$	
$p \in \mathcal{P}_C - \cup \mathcal{B}$	
$g \in \mathcal{L} \cap \mathcal{G}$	
$s \in \mathcal{L} \cap \mathcal{S}$	
$c \in \mathcal{L} \wedge \text{Dec}(c) = \{d_1, \dots, d_n\}$	
$g \in \mathcal{N} \cap \mathcal{G}$	
$s \in \mathcal{N} \cap \mathcal{S}$	
$c \in \mathcal{N} \wedge \text{Dec}(c) = \{d_1, \dots, d_n\}$	

Table 3: Relation between ER^+ symbols and ORM constructs

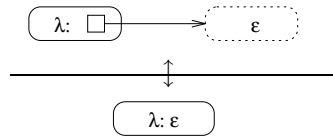
5.3 ER^+

Taking pure ER as a starting point, we can define ER^+ by adding more symbols. These extra symbols deal with fact objectification, generalisation, specialisation as well as power, sequence and schema types.

The symbol used for fact objectification is similar to the one used in NIAM or PSM, in that it is made by drawing an entity type symbol around a fact type symbol. For the representation of power and sequence types we distinct between concrete and abstract types. This representation is based on the representation of multi- and data-valued attributes in [EGH⁺92]. Except for generalisation and specialisation, all extra symbols are depicted in table 3. In this table, the Dec function is defined as:

$$\text{Dec}(c) \triangleq \text{if } c \in \mathcal{C} \text{ then } \{d \mid c \prec d\} \text{ else } \perp \text{ fi}$$

The notation for label types in \mathcal{G} and \mathcal{S} can be simplified, using the following derivation rule, in which λ stands for *list* or *set* and ε stands for an arbitrary label type.



Generalisation and specialisation are represented using one single symbol. This symbol is taken from [EGH⁺92], where the symbol represents only one single construct, called type construction. Despite that,

construct	symbol
$x \text{ IsGenOf } \{y_1, \dots, y_n\}$	

Table 4: Depicting generalisation in ER^+

it is possible to use this single symbol for representing the combination of the two constructs generalisation and specialisation.

Generalisation can be drawn in a very direct way (see table 4), using Gen_1 and $IsGenOf$:

$$\begin{aligned}
 x \text{ Gen}_1 y &\triangleq x \text{ Gen } y \wedge x \text{ ldfBy}_1 y \\
 x \text{ IsGenOf } Y &\triangleq Y = \{y \mid x \text{ Gen}_1 y\}
 \end{aligned}$$

For specialisation hierarchies, the drawing algorithm is less straightforward. As for generalisation, we introduce some predicates:

$$\begin{aligned}
 x \text{ Spec}_1 y &\triangleq x \text{ Spec } y \wedge x \text{ ldfBy}_1 y \\
 X \text{ AreSpecsOf } y &\triangleq X = \{x \mid x \text{ Spec}_1 y\}
 \end{aligned}$$

For drawing $Spec$ in ER^+ we execute, for any X and y such that $X \text{ AreSpecsOf } y$, the following algorithm:

```

E := {S | R ⊢ exclusion(S)} ∪ {{x} | x ∈ O}
while X ≠ ∅ do
  let {x1, ..., xn} ⊆ X such that {x1, ..., xn} ∈ E and n is maximised
  /* Note that the let makes a non-deterministic choice */
  draw
  
  X := X - {x1, ..., xn}
od

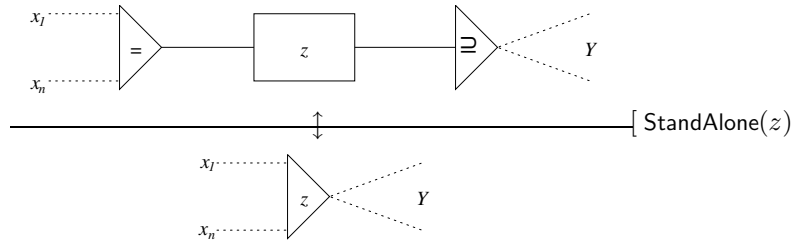
```

Note: with **maximised** is meant that the n must be chosen such that there is no larger set of x_i 's such that the conditions are met.

This algorithm also deals with exclusion constraints, since all output types of a type construction (the types drawn at the top of the triangle) have to be disjoint.

In [EGH⁺92], the general case of type construction consists of n input types and m output types (the types drawn at the base and the top of the triangle respectively), where n and m are greater than 1. The notations given above do not result in this general construct. In the following derivation rule, we show how this

general type construction relates to the constructions given above:



where the predicate StandAlone states that an object type is not used directly in the construction of another object type:

$$\text{StandAlone}(x) \triangleq x \notin (\mathcal{F}_O \cup \text{ran}(\text{Elt}))$$

The complete zoo example is now depicted in figure 8 in the ER⁺ style. Note that we did not depict all names of object types, and connectors, for reasons of clarity.

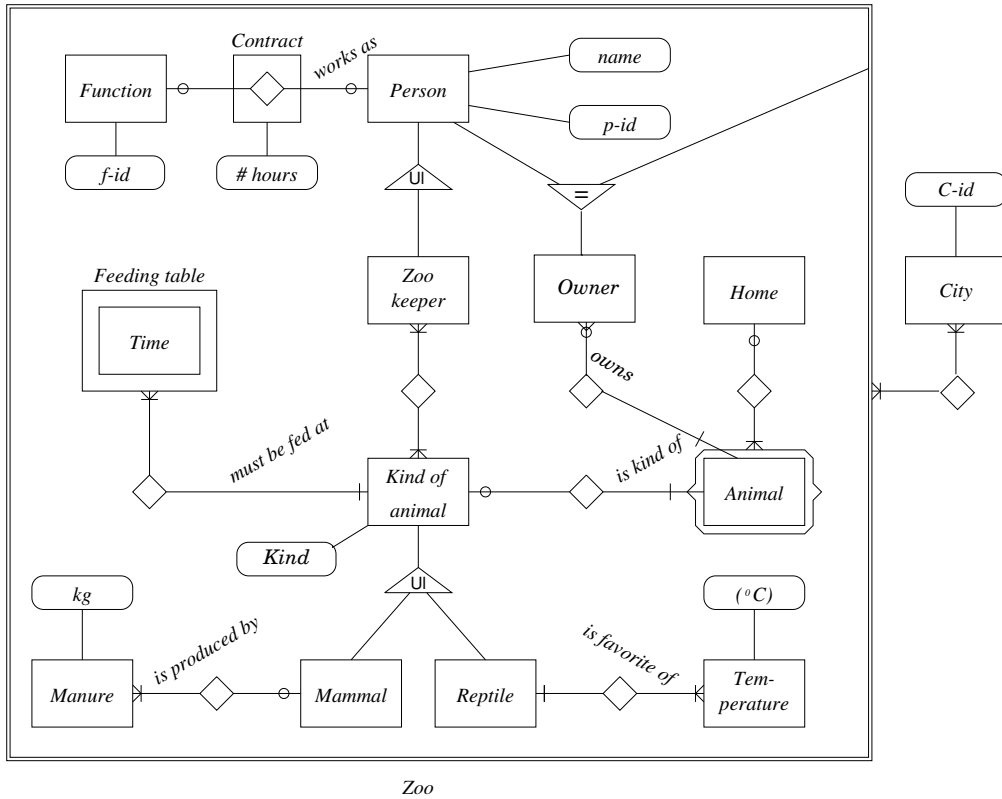


Figure 8: Complete zoo example in ER⁺

6 NIAM way of communicating

The techniques, which use a NIAM way of communicating are mostly extensions of the NIAM modelling technique presented in [NH89]. The Predicator Set Model, which is presented in [HW93], is also such an extension of NIAM. In this section we make a distinction between the concepts used in NIAM and the concepts used in PSM.


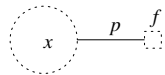


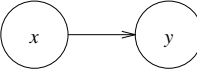
construct	symbol
$x \in \mathcal{N} \cap \mathcal{A}$	
$p \in \mathcal{P}$	
$f \in \mathcal{F}$	
$l \in \mathcal{L} \cap \mathcal{A}$	
$x \text{ Spec } y$	

Table 5: Relation between ORM constructs and pure NIAM symbols

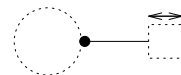
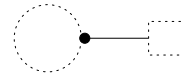


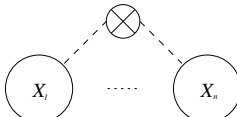
construct	symbol
$\text{unique}(p) \wedge \text{total}(p)$	
$\neg \text{unique}(p) \wedge \text{total}(p)$	
$\text{unique}(p) \wedge \neg \text{total}(p)$	
$\neg \text{unique}(p) \wedge \neg \text{total}(p)$	
$\text{exclusion}(\{X_1, \dots, X_n\})$	

Table 6: Constraints

6.1 Pure NIAM

In pure NIAM the following ORM concepts can be identified: object types (\mathcal{O}), entity types (\mathcal{E}), roles or predicates (\mathcal{P}), fact types (\mathcal{F}), and label types (\mathcal{L}). In table 5 the relation between the ORM constructs and NIAM are provided. In table 6, the NIAM-symbols for uniqueness, total role and exclusion constraints are given.

6.2 PSM

Not all of the constructs of ORM can be represented using NIAM. Therefore, NIAM was extended with some constructs, resulting in PSM (see [HW93]). In PSM, all concepts of the ORM kernel can be visu-

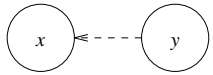
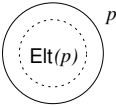

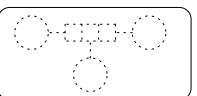
construct	symbol
$x \text{ Gen } y$	
$p \in \mathcal{G}$	
$s \in \mathcal{S}$	
$\text{Dec}(c) = \{d_1, \dots, d_n\}$	

Table 7: Relation between the ORM constructs and the PSM symbols

alised. The additional concepts are: generalisation, power typing, sequence typing, and schema typing.

In table 7 the PSM symbols corresponding to the ORM constructs are given. The complete zoo example is depicted in figure 9.

6.3 Evaluation

From our running example follows that most schemas denoted with the ER way of communicating can be directly represented in the NIAM way of communicating using the ORM kernel as an intermediate. However, the reverse will usually not be as easy. A NIAM schema usually contains more details than an ER schema does; in particular with regards to attributes. NIAM schemas tend to be more elaborate.

Some authors, and we tend to agree with this, suggest to regard ER schemas as an abstraction from NIAM schemas (and ORM in general) ([CH94], [Cam94]). We expect no problems in defining the abstraction mechanisms provided in [CH94] in terms of an ORM kernel.

7 Conclusions

By making a clear distinction between the different aspects of methods, we were able to demonstrate the possibility to build a CASE-tool with different ways of communicating, using a single way of modelling. We have presented a first attempt for an ORM kernel, general enough to contain both ER-like and NIAM-like models, together with two appropriate ways of communicating. Furthermore, we have shown how the general ORM kernel can be *tuned* to match one's own flavour of ER or NIAM.

As a next step, an actual CASE-tool should be build supporting the ORM kernel and multiple ways of communicating. Furthermore, the completeness of the ORM kernel should be validated. e.g. can object oriented data models be 'linked' into the kernel. So far, we do not yet know whether the ORM kernel is indeed general enough to include object oriented data models, and this certainly requires more investigation. A new avenue in this research effort is discussed in [FHL97] where data modelling techniques are expressed as categories in category theory ([BW90]).

The ORM kernel further illustrates that there is little difference in the way of modelling of ER and NIAM (more extensive comparisons can be found in [Bro93a], [Mar93] and [Bro93b]). The main difference

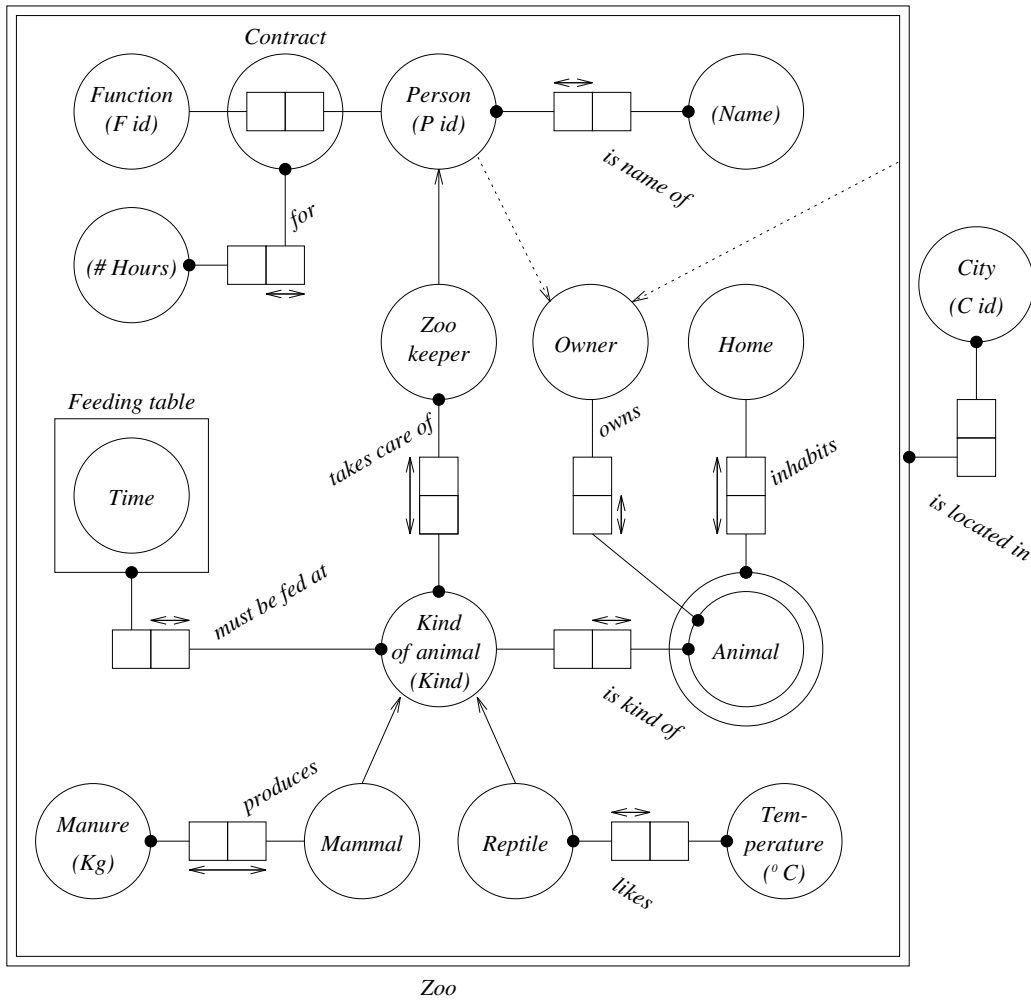


Figure 9: Complete zoo example in PSM

between ER and NIAM lies in their respective ways of working, any research concerned with the underlying way of modelling is interchangeable between both ‘worlds’.

Acknowledgements

We would like to thank the anonymous referees for their comments and suggestions, which have lead to improvements of the original article.

References

- [AH87] S. Abiteboul and R. Hull. IFO: A Formal Semantic Database Model. *ACM Transactions on Database Systems*, 12(4):525–565, December 1987.
- [Avi95] D.E Avison. *Information Systems Development: Methodologies, Techniques and Tools*. McGraw-Hill, New York, New York, 2nd edition, 1995. ISBN 0077092333

- [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.
- [Bro93a] G.H.W.M. Bronts. Formalization of an Object Model. Master’s thesis, University of Nijmegen, Nijmegen, The Netherlands, 1993.
- [Bro93b] S.J. Brouwer. PSM vs ‘the rest of the world’, Vergelijking tussen ER en PSM. Master’s thesis, University of Nijmegen, Nijmegen, The Netherlands, 1993. In Dutch.
- [Bub86] J.A. Bubenko. Information System Methodologies - A Research View. In T.W. Olle, H.G. Sol, and A.A. Verrijn-Stuart, editors, *Information Systems Design Methodologies: Improving the Practice*, pages 289–318. North-Holland/IFIP WG8.1, Amsterdam, The Netherlands, EU, 1986.
- [BW90] M. Barr and C. Wells. *Category Theory for Computing Science*. Prentice-Hall, Englewood Cliffs, New Jersey, 1990.
- [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.
- [BZL94] G.P. Bakema, J.P.C. Zwart, and H. van der Lek. Fully Communication Oriented NIAM. In *Proceedings of NIAM-ISDM 2*, pages 1–35, Albuquerque, New Mexico, August 1994.
- [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.
- [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.
- [Che76] P.P. Chen. The entity-relationship model: Towards a unified view of data. *ACM Transactions on Database Systems*, 1(1):9–36, March 1976.
- [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.
- [DMV88] O.M.F. De Troyer, R. Meersman, and P. Verlinden. RIDL* on the CRIS Case: A Workbench for NIAM. In T.W. Olle, A.A. Verrijn-Stuart, and L. Bhabuta, editors, *Information Systems Design Methodologies: Computerized Assistance during the Information Systems Life Cycle*, pages 375–459, Amsterdam, The Netherlands, EU, 1988. North-Holland/IFIP WG8.1.
- [EGH⁺92] G. Engels, M. Gogolla, U. Hohenstein, K. Hülsmann, P. Löhr-Richter, G. Saake, and H-D. Ehrich. Conceptual modelling of database applications using an extended ER model. *Data & Knowledge Engineering*, 9(4):157–204, 1992.
- [EN94] R. Elmasri and S.B. Navathe. Advanced data models and emerging trends. In *Fundamentals of Database Systems*, chapter 21. 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.
- [FHL97] P.J.M. Frederiks, A.H.M. ter Hofstede, and E. Lippe. A Unifying Framework for Conceptual Data Modelling Concepts. *Information and Software Technology*, 39(1):15–25, January 1997.

- [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.
- [Hal92] 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.
- [HE92] U. Hohenstein and G. Engels. SQL/EER-syntax and semantics of an entity-relationship-based query Language. *Information Systems*, 17(3):209–242, 1992.
- [HH93] T.A. Halpin and J. Harding. Automated Support for Verbalization of Conceptual Schemas. In S. Brinkkemper and F. Harmsen, editors, *Proceedings of the Fourth Workshop on the Next Generation of CASE Tools*, pages 151–161, Paris, France, June 1993.
- [HHO92] T.A. Halpin, J. Harding, and C-H. Oh. Automated Support for Subtyping. In B. Theodoulidis and A. Sutcliffe, editors, *Proceedings of the Third Workshop on the Next Generation of CASE Tools*, pages 99–113, Manchester, United Kingdom, May 1992.
- [HNSE87] U. Hohenstein, L. Neugebauer, G. Saake, and H-D. Ehrich. Three-Level-Specification of Databases using an extended Entity-Relationship Model. In R.R. Wagner, R. Traunmüller, and H.C. Mayr, editors, *Informationsbedarfsermittlung und -analyse für den Entwurf von Informationssystemen*, pages 58–88, Berlin, Germany, 1987. Springer-Verlag.
- [HO92] T.A. Halpin and M.E. Orłowska. Fact-oriented modelling for data analysis. *Journal of Information Systems*, 2(2):97–119, April 1992.
- [Hof93] A.H.M. ter Hofstede. *Information Modelling in Data Intensive Domains*. PhD thesis, University of Nijmegen, Nijmegen, The Netherlands, 1993.
- [HP95] T.A. Halpin and H.A. Proper. Subtyping and Polymorphism in Object-Role Modelling. *Data & Knowledge Engineering*, 15:251–281, 1995.
- [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.
- [HPW94] A.H.M. ter Hofstede, H.A. Proper, and Th.P. van der Weide. A Conceptual Language for the Description and Manipulation of Complex Information Models. In G. Gupta, editor, *Seventeenth Annual Computer Science Conference*, volume 16 of *Australian Computer Science Communications*, pages 157–167, Christchurch, New Zealand, January 1994. University of Canterbury. ISBN 047302313
- [HW92] A.H.M. ter Hofstede and Th.P. van der Weide. Formalisation of techniques: chopping down the methodology jungle. *Information and Software Technology*, 34(1):57–65, January 1992.
- [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.
- [JG87] D.A. Jardine and J.J. van Griethuysen. A logic-based information modelling language. *Data & Knowledge Engineering*, 2:59–81, 1987.
- [Ken84] F. Kensing. Towards Evaluation of Methods for Property Determination: A Framework and a Critique of the Yourdon-DeMarco Approach. In Th.M.A. Bemelmans, editor, *Beyond Productivity: Information Systems Development for Organizational Effectiveness*, pages 325–338. North-Holland, Amsterdam, The Netherlands, 1984.
- [Lev79] A. Levy. *Basic Set Theory*. Springer-Verlag, Berlin, Germany, 1979.

- [Luk91] W. Lukaszewicz. *Non-Monotonic Reasoning; Formalization of Commonsense Reasoning*. Ellis Horwood series in Artificial Intelligence, 1991.
- [Mar93] C.L.J. Martens. PSM vs ‘the rest of the world’, Vergelijking tussen IFO en PSM. Master’s thesis, University of Nijmegen, Nijmegen, The Netherlands, 1993. In Dutch.
- [Mat81] L. Mathiassen. *Systemudvikling og Systemudviklings-Metode*. PhD thesis, Aarhus University, Aarhus, Denmark, 1981. (In Danish).
- [McC89] C.L. McClure. *CASE is Software Automation*. Prentice-Hall, Englewood Cliffs, New Jersey, 1989. ISBN 0131193309
- [MSW92] P. McBrien, A.H. Seltviet, and B. Wangler. An Entity-Relationship Model Extended to describe Historical Information. In A.K. Majumdar and N. Prakash, editors, *Proceedings of the International Conference on Information Systems and Management of Data (CISMOD 92)*, pages 244–260, Bangalore, India, July 1992.
- [NH89] G.M. Nijssen and T.A. Halpin. *Conceptual Schema and Relational Database Design: a fact oriented approach*. Prentice-Hall, Sydney, Australia, 1989. ASIN 0131672630
- [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
- [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.
- [Sol83] H.G. Sol. A Feature Analysis of Information Systems Design Methodologies: Methodological Considerations. In T.W. Olle, H.G. Sol, and C.J. Tully, editors, *Information Systems Design Methodologies: A Feature Analysis*, pages 1–7. North-Holland/IFIP WG8.1, Amsterdam, The Netherlands, EU, 1983. ISBN 0-444-86705-8
- [Sol88] H.G. Sol. Information Systems Development: A Problem Solving Approach. In *Proceedings of 1988 INTEC Symposium Systems Analysis and Design: A Research Strategy*, Atlanta, Georgia, 1988.
- [SWS89] P.S. Seligmann, G.M. Wijers, and H.G. Sol. Analyzing the structure of I.S. methodologies, an alternative approach. In R. Maes, editor, *Proceedings of the First Dutch Conference on Information Systems*, Amersfoort, The Netherlands, EU, 1989.
- [SZ91] P. Shoval and S. Zohn. Binary-Relationship integration methodology. *Data & Knowledge Engineering*, 6(3):225–250, 1991.
- [TMHY80] D. Teichrow, P. Macasovic, E.A. Hershey, and Y. Yamamoto. Application of the entity-relationship approach to information processing systems modeling. In P.P. Chen, editor, *Entity-Relationship Approach to Systems Analysis and Design*, pages 15–38, Amsterdam, The Netherlands, 1980. North-Holland.
- [VB82] G.M.A. Verheijen and J. van Bekkum. NIAM: an Information Analysis Method. In T.W. Olle, H.G. Sol, and A.A. Verrijn-Stuart, editors, *Information Systems Design Methodologies: A Comparative Review*, pages 537–590. North-Holland/IFIP WG8.1, Amsterdam, The Netherlands, EU, 1982.
- [WH90] G.M. Wijers and H. Heijes. Automated Support of the Modelling Process: A view based on experiments with expert information engineers. In B. Steinholz, A. Solvberg, and L. Bergman, editors, *Proceedings of the Second Nordic Conference CAiSE’90 on Advanced Information Systems Engineering*, volume 436 of *Lecture Notes in Computer Science*, pages 88–108, Stockholm, Sweden, EU, 1990. Springer-Verlag. ISBN 3540526250
- [Win90] J.J.V.R. Wintraecken. *The NIAM Information Analysis Method: Theory and Practice*. Kluwer, Deventer, The Netherlands, EU, 1990.

Contents

1	Introduction	1
2	Syntactical aspects of the ORM Kernel	4
2.1	Abstract and concrete object types	4
2.2	Atomic types	4
2.3	Fact typing	4
2.4	Power typing	5
2.5	Sequence typing	5
2.6	Identification hierarchy	6
2.7	Specialisation	6
2.8	Generalisation	7
2.9	Schema typing	8
2.10	Additional properties	9
3	Semantical aspects of the ORM kernel	10
3.1	Naming of Concepts	10
3.2	Constraints and Semantics	10
4	Tuning the Kernel	12
4.1	ER	13
4.2	ORM	13
5	ER way of communicating	14
5.1	Running example	14
5.2	Pure ER	16
5.3	ER ⁺	17
6	NIAM way of communicating	19
6.1	Pure NIAM	20
6.2	PSM	20
6.3	Evaluation	21
7	Conclusions	21