Towards a Unifying Object Role Modelling Theory

S.J. Brouwer[1] C.L.J. Martens[1] G.H.W.M. Bronts[1] H.A. Proper[1]

PUBLISHED AS:

S.J. Brouwer, C.L.J. Martens, G.H.W.M. Bronts, and H.A. (Erik) Proper. Towards a Unifying Object Role Modelling Approach. In T.A. Halpin and R. Meersman, editors, *Proceedings of the First International Conference on Object–Role Modelling (ORM–1)*, pages 259–273, July 1994. ISBN 0867765658

Abstract

In this article we briefly present 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. [8]). This has led to the Methodology Jungle ([2]). In particular a wide range of data modelling techniques exists, for instance: ER based modelling techniques ([11], [13], [24]), and Object Role Modelling (ORM) techniques ([26], [3], [29], [22], [17]). 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 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. As an illustration of the relationship between the ORM kernel and other modelling techniques, consider figure 1. This figure also illustrates the possibility of using the textual language LISA-D ([20], [21]) as a way to represent models.

Figure 1: ORM Kernel

[1]Computing Science Institute, University of Nijmegen, Toernooiveld, 6525 ED Nijmegen, The Netherlands (All correspondence should be sent to E.Proper@acm.org)

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 [3] and [19], 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 [28] and [27]. Issues regarding internal representations of data models conforming to the ORM kernel have been studied in [4], [15] and [16]. Finally, in [20] and [21] 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.

To illustrate the elegance of the ORM kernel concept, we present the ORM kernel first, and then define an (E)ER ([11], [14]) and a NIAM ([26]) way of communicating on top of it as case studies. 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 proof of the generality of this kernel. It in particular shows that the ORM modelling concepts are expressive enough to cater for (E)ER based models, from which can be concluded that a data modelling kernel should at least support all ORM concepts.

This article only provides an overview of the ORM kernel. For a more elaborate discussion of the presented ideas, refer to the full paper: [6]. Furthermore, the Universities of Nijmegen and Queensland are heavily involved in joint research aiming at an integration between their respective versions of ORM (a first result can be found in [18]).

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. Sections 4 and 5 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. In the ORM kernel [22], an *information structure* $\mathcal{I}$ consists of the following basic components:

1. 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*. Instances of label, or value, types ($\mathcal{L}$) are directly denotable, whereas instances of the non-label object types ($\mathcal{N}$) are not directly denotable.

2. A finite set $\mathcal{P}$ of *predicators*.

3. A set $\mathcal{A} \subseteq \mathcal{O}$ of *atomic types*. Atomic types 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.

4. A partition $\mathcal{F}$ of the set $\mathcal{P}$. The elements of $\mathcal{F}$ are called *fact types*, or *relationship types*.

5. A set $\mathcal{G}$ of *power types*. Power types form a special class of object types ($\mathcal{G} \subseteq \mathcal{O}$). Power typing is a type constructor which leads to an object type that is instantiated with subsets over an underlying object type.

6. A set $\mathcal{S}$ of *sequence types*. Sequence types form a special class of object types ($\mathcal{S} \subseteq \mathcal{O}$). Sequences of an underlying object type are formed by sequence typing.

7. A set $\mathcal{C}$ of *schema types*: $\mathcal{C} \subseteq \mathcal{O}$. Schema typing provides the opportunity to describe an information structure in a top-down fashion. Schema types are instantiated by populations of the underlying schema.

8. A function $\mathsf{Base} : \mathcal{P} \rightarrow \mathcal{O}$. A fact type is an association between a number of object types. The involvement of an object type in a fact type is represented by a predicator. The base of a predicator is the object part of that predicator.

9. A function $\mathsf{Elt} : \mathcal{G} \cup \mathcal{S} \rightarrow \mathcal{O}$. This function yields the element type of power types and sequence types.

10. A relation $\prec \subseteq \mathcal{C} \times \mathcal{O}$. This relation describes the decomposition of schema types.

11. A partial order $\mathsf{IdfBy} \subseteq \mathcal{A} \times \mathcal{O}$ on object types, capturing the inheritance hierarchy.
12. A partial order $\mathsf{Spec} \subseteq \mathsf{IdfBy}$ specifying that part of the identification hierarchy concerned with specialisation.
13. A partial order $\mathsf{Gen} \subseteq \mathsf{IdfBy}$ specifying that part of the identification hierarchy concerned with generalisation. Generalisation and specialisation are different in nature, and originate from different axioms in set theory ([22]).

Note that one may argue that power types, sequence types, and schema types are no atomical concepts. They may indeed be expressed in terms of other concepts (see e.g. [23]) if an extra class of constraints is introduced. However, from a formal and pragmatical point of view, their presence in the kernel is justified.

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:

[ORM1] (*disjunction*) $\mathcal{A}$, $\mathcal{F}$, $\mathcal{G}$, $\mathcal{S}$ and $\mathcal{C}$ form a partition of $\mathcal{O}$

In the remainder of this section, we elaborate more on the above constructs, and provide some examples.

2.1 Abstract and concrete objects 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. 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. The concrete and the abstract world may not be mixed:

[ORM2] (*strict separation*) $x = \mathsf{Elt}(y) \vee x \prec y \vee x\ \mathsf{IdfBy}\ y \ \Rightarrow\ x, y \in \mathcal{L} \vee x, y \in \mathcal{N}$

The only way to bridge the gap between concrete and abstract world, is by means of so called *bridge types*, being a special kind of (binary) fact type. The set of bridge types ($\mathcal{B}$) is:

$$\mathcal{B} \triangleq \left\{ \{p, q\} \in \mathcal{F} \;\middle|\; p \in \mathcal{P}_{\mathcal{L}} \wedge q \in \mathcal{P}_{\mathcal{N}} \right\}$$

where $\mathcal{P}_X \triangleq \left\{ p \in \mathcal{P} \;\middle|\; \mathsf{Base}(p) \in X \right\}$. Bridge types, should be the single way to cross the gap between the concrete and abstract worlds:

[ORM3] (*bridges only*) $f \in \mathcal{F} \ \Rightarrow\ f \subseteq \mathcal{P}_{\mathcal{L}} \vee f \subseteq \mathcal{P}_{\mathcal{N}} \vee f \in \mathcal{B}$

The predicators that constitute a bridge type $b = \{p, q\}$ can be extracted by the operators concr and abstr. These operators are defined by $\mathsf{concr}(b) \triangleq p$ such that $p \in b \cap \mathcal{P}_{\mathcal{L}}$ and, $\mathsf{abstr}(b) \triangleq p$ such that $p \in b \cap \mathcal{P}_{\mathcal{N}}$, respectively.

2.2 Fact typing

One of the key concepts in data modelling is the concept of fact type (or 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 objects types. A fact type may be treated as an object type (*fact objectification*), and can therefore play a role in other fact types. A bridge type is a special fact type, relating the abstract and the concrete worlds.

Figure 2: Example fact/relationship type

As an example fact type, consider the ER schema depicted in figure 2. In this schema we have: $\mathcal{E} = \{X_1, X_2\}$, $\mathcal{F} = \{R\}$, and $\mathcal{P} = \{p_1, p_2\}$ 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.c. the ER style.

2.3 Power typing

The concept of *power type* in ORM forms the data modelling pendant of power sets in conventional set theory. 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). An example power type is provided in figure 3, in the PSM style of communicating. This example is concerned with convoys, being sets of ships, where $\mathsf{Elt}(\mathsf{Convoy}) = \mathsf{Ship}$.

Figure 3: Convoys of ships

2.4 Sequence typing

Sequence typing offers the opportunity to represent sequences, built from an underlying element type. Instances of a sequence type thus correspond to sequences of instances from its underlying element type. The element type of a sequence type is also found by the function Elt.

2.5 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 4, an example of a schema type concerned with activity graphs and their decomposition is given.

Figure 4: Meta schema of activity graphs

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.

2.6 Identification hierarchy

In the ORM kernel, we define the notion of *identification hierarchy*, which generalises generalisation and specialisation. The identification hierarchy is defined as a partial order (asymmetric and transitive) IdfBy on object types, with the convention that a IdfBy 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:

[ORM4] $x \,\mathsf{IdfBy}\, y \Rightarrow \neg y \,\mathsf{IdfBy}\, x$ and $x \,\mathsf{IdfBy}\, y \,\mathsf{IdfBy}\, z \Rightarrow x \,\mathsf{IdfBy}\, z$

We define IdfBy_1 as the one step counterpart of the IdfBy relation:

$$x \,\mathsf{IdfBy}_1\, y \triangleq x \,\mathsf{IdfBy}\, y \wedge \neg\exists_z \,[x \,\mathsf{IdfBy}\, z \,\mathsf{IdfBy}\, y]$$

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

[ORM5] $x \,\mathsf{IdfBy}\, y \Rightarrow x \,\mathsf{IdfBy}_1\, y \vee \exists_p \,[x \,\mathsf{IdfBy}_1\, p \,\mathsf{IdfBy}\, y]$

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

[ORM6] If F is a property for object types, such that: $\forall_{x:y \,\mathsf{IdfBy}_1\, x} \,[F(x)] \;\Rightarrow\; F(y)$ for any y, then $\forall_{x \in \mathcal{O}} \,[F(x)]$

The identification hierarchy is a result of specialisation and generalisation:

[ORM7] $x \,\mathsf{IdfBy}_1\, y \Rightarrow x \,\mathsf{Gen}\, y \vee x \,\mathsf{Spec}\, y$ and $x \,\mathsf{Gen}\, y \vee x \,\mathsf{Spec}\, y \Rightarrow x \,\mathsf{IdfBy}\, y$

2.7 Specialisation

Specialisation is a mechanism for representing one or more (possibly overlapping) subtypes of an object type. 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* ([3]), and can be specified by means of a LISA-D expression ([20]). 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:

$\mathsf{CD} = \mathsf{Medium\ having\ Type\ 'CD'}$ and $\mathsf{Record} = \mathsf{Medium\ having\ Type\ 'Record'}$

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 $a \,\mathsf{Spec}\, b$ is: a is a specialisation of b, or a is a subtype of b.

[ORM8] If $x \,\mathsf{IdfBy}\, y \,\mathsf{IdfBy}\, z$ then: $x \,\mathsf{Spec}\, y \,\mathsf{Spec}\, z \iff x \,\mathsf{Spec}\, z$

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

Figure 5: An example of specialisation

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 ([22]) and therefore have a different expressive power. Properties are inherited "upward" in a generalisation hierarchy instead of "downward", which is the case for specialisation. This also implies that the identification of a generalised object type depends on the identification of its specifiers. The concept of generalisation is introduced as a partial order Gen. The expression $a \,\mathsf{Gen}\, b$ stands for: a is a generalisation of b, or b is a specifier of a.

Figure 6: An example of generalisation

[ORM9] (*transitivity completeness*) If x IdfBy y IdfBy z then: x Gen y Gen $z \iff x$ Gen z

Remark 2.1

A lot of confusion exists with regards to specialisation as opposed to generalisation. In this proposal for the ORM kernel, we have chosen for PSM's ([22]) notion of generalisation, which is not *simply the inverse of specialisation. Furthermore, [22] shows that the presented notions of generalisation and specialisation originate from different axioms from set theory.*

However, other views on generalisation do exist. In [13] the Entity Category Relationship (ECR) model is introduced, which uses the concept of sub-categories to specialize, and categories to build polymorphic types. A recent EER variant is presented in ([12]). In this EER version, the relationship between super types and their sub types is referred to as the super class/sub class relationship. This specialisation mechanism corresponds to NIAM's standard subtyping The only real difference is that subtypes not always have to be defined by a subtype defining rule in which case it is a user-defined subclass.

In ([12]) the authors also identify the problem that in some subclass hierarchies more than one common superclass (pater familias in PM terminology) is required. For these situations they introduces the notion of category (which is now slightly different from the category notion used in [13]). Categorization corresponds to the notion of union in set theory, and indeed corresponds to generalisation.

On the other hand, IFO ([1]) features a notion of generalisation and specialisation corresponding to PSM's approach. The specialisation and generalisation relations in IFO are depicted using two different arrows, with differing semantics.

More details on the discussion on generalisation vs specialisation can be found in [18]. □

3 Semantical aspects of the ORM kernel

In this section we briefly discuss 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. At the end of this section, we provide a running example which will be illustrated for the (E)ER and NIAM cases.

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: $\mathtt{ONm} : \mathcal{O} \rightarrowtail \mathtt{Names}$, which is specified in the schema upon their introduction. The (partial) function $\mathtt{Obj} : \mathtt{Names} \rightarrowtail \mathcal{O}$ is the left-inverse of ONm, and relates object

type names to their corresponding object type: $\forall_{x \in \mathsf{dom}(\mathsf{ONm})}\,[\mathsf{Obj}(\mathsf{ONm}(x)) = x]$, where $\mathsf{dom}(\mathsf{ONm})$ 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 $\mathsf{Conn} : \mathcal{P} \times \mathcal{P} \rightarrowtail \mathtt{Names}$. Names can only be given to pairs of predicators which are part of one single fact type: $\mathsf{Conn}(p,q)\!\downarrow \Rightarrow p \in \mathsf{Fact}(q)$. Note that in some interpretations, *connector names* may correspond to the notion of role name. In figure 2, we could for instance have the following names:

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

3.2 Constraints and Semantics

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 $\mathcal{O}$: $\mathsf{Pop} : \mathcal{O} \rightarrow \wp(\Omega)$, where Ω is the set of values that can occur in the population.

Constraints 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. In this article we only consider one class of constraints, the *exclusion constraint*. An exclusion constraint states that the population of different object types may not have any instance in common: $\mathsf{exclusion}(X) \triangleq \forall_{x,y \in X}\,[x \neq y \Rightarrow \mathsf{Pop}(x) \cap \mathsf{Pop}(y) = \varnothing]$

In the full version of this paper ([6]), we also provide some example axioms to tune the above presented ORM kernel to concrete ER and NIAM versions (e.g. binary NIAM).

3.3 Running example

We employ the example of a zoo as a running example in the remainder of this article. Consider a company owning several zoos 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). 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, which will be employed in the remainder of this section, for the example U∘D to be unreadable, and incomprehensible, we could not agree with you more. The readability and comprehensability 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 7 and figure 8.

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

$$\begin{array}{lll} \mathcal{A} = \{x_1,\ldots,x_{19}\} & \mathcal{P} = \{p_1,\ldots,p_{20}\} & \mathcal{F} = \{\{p_1,p_2\},\ldots,\{p_{19},p_{20}\}\} \\ \mathcal{G} = \{g\} & \mathcal{S} = \{s\} & \mathcal{C} = \{c\} \end{array}$$

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

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

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

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

$$\begin{aligned}\mathcal{L} &= \{\mathsf{Obj(Name)}, \mathsf{Obj(F\ id)}, \mathsf{Obj(P\ id)}, \mathsf{Obj(\#\ Hours)}, \mathsf{Obj(C\ id)}, \mathsf{Obj(Kg)}, \mathsf{Obj(^oC)}\}\\ \mathcal{N} &= \{\mathsf{Obj(Function)}, \mathsf{Obj(Person)}, \mathsf{Obj(Zoo\ keeper)}, \mathsf{Obj(Home)}, \mathsf{Obj(Time)}, \mathsf{Obj(Kind\ of\ animal)}\\ &\quad \mathsf{Obj(City)}, \mathsf{Obj(Animal)}, \mathsf{Obj(Manure)}, \mathsf{Obj(Mammal)}, \mathsf{Obj(Reptile)}, \mathsf{Obj(Temperature)},\\ &\quad \mathsf{Obj(Animals)}, \mathsf{Obj(Feeding\ table)}, \mathsf{Obj(Zoo)}, \mathsf{Obj(Contract)}, \{p_3, p_4\}, \ldots, \{p_{19}, p_{20}\}\}\end{aligned}$$

The bases of the predicators are:

$$\begin{array}{lll}\mathsf{Base}(p_1) = \mathsf{Obj(Function)} & \mathsf{Base}(p_2) = \mathsf{Obj(Person)} & \mathsf{Base}(p_3) = \mathsf{Obj(Person)}\\ \mathsf{Base}(p_4) = \mathsf{Obj(Name)} & \mathsf{Base}(p_5) = \mathsf{Obj(Contract)} & \mathsf{Base}(p_6) = \mathsf{Obj(\#\ Hours)}\\ \mathsf{Base}(p_7) = \mathsf{Obj(Zoo\ keeper)} & \mathsf{Base}(p_8) = \mathsf{Obj(Kind\ of\ animal)} & \mathsf{Base}(p_9) = \mathsf{Obj(Home)}\\ \mathsf{Base}(p_{10}) = \mathsf{Obj(Animals} & \mathsf{Base}(p_{11}) = \mathsf{Obj(Zoo)} & \mathsf{Base}(p_{12}) = \mathsf{Obj(City)}\\ \mathsf{Base}(p_{13}) = \mathsf{Obj(Feeding\ table)} & \mathsf{Base}(p_{14}) = \mathsf{Obj(Kind\ of\ animal)} & \mathsf{Base}(p_{15}) = \mathsf{Obj(Kind\ of\ animal)}\\ \mathsf{Base}(p_{16}) = \mathsf{Obj(Animal)} & \mathsf{Base}(p_{17}) = \mathsf{Obj(Manure)} & \mathsf{Base}(p_{18}) = \mathsf{Obj(Mammal)}\\ \mathsf{Base}(p_{19}) = \mathsf{Obj(Reptile)} & \mathsf{Base}(p_{20}) = \mathsf{Obj(Temperature)} & \end{array}$$

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

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

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

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

In the example the following connectors are used:

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

The information structure contains one specialisation and two generalisations:

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

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

4 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 [11]. 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 [14].

4.1 Pure ER

As stated before, pure ER stands for the technique introduced in [11]. In this technique the following ORM constructs can be identified: entity types, fact types, predicators, label types and bridge types. In table 1 these constructs and their pure ER representation are given. Although [11] 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 $\mathcal{F}_\mathcal{O}$ corresponds to the objectified fact types, and is defined by: $\mathcal{F}_\mathcal{O} \triangleq \mathsf{ran(Base)} \cap \mathcal{F}$ where ran returns the range of a function. Note that in pure ER: $\mathcal{P}_\mathcal{L} \subseteq \bigcup \mathcal{B}$.

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

Table 1: Relation between ORM constructs and ER symbols

construct	symbol
$f \in \mathcal{F}_{\mathcal{O}}$	
$p \in \mathcal{P}_{\mathcal{L}} - \cup \mathcal{B}$	
$g \in \mathcal{L} \cap \mathcal{G}$	
$s \in \mathcal{L} \cap \mathcal{S}$	
$c \in \mathcal{L} \wedge \mathsf{Dec}(c) = \{d_1, \ldots, d_n\}$	
$g \in \mathcal{N} \cap \mathcal{G}$	
$c \in \mathcal{N} \cap \mathcal{S}$	
$c \in \mathcal{N} \wedge \mathsf{Dec}(c) = \{d_1, \ldots, d_n\}$	
$x \ \mathsf{IsGenOf} \ \{y_1, \ldots, y_n\}$	

Table 2: Relation between ER$^+$ symbols and ORM constructs

4.2 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 distinguish between concrete and abstract types. This representation is based on the representation of multi- and data-valued attributes in [14]. Except for generalisation and specialisation, all extra symbols are depicted in table 2. In this table, the Dec function is defined as: $\mathsf{Dec}(c) \triangleq$ **if** $c \in \mathcal{C}$ **then** $\{d \mid c \prec d\}$ **else** $\perp$ **fi**. Generalisation can be drawn in a very direct way, using Gen_1 and $\mathsf{IsGenOf}$:

$$x\ \mathsf{Gen}_1\ y \triangleq x\ \mathsf{Gen}\ y \wedge x\ \mathsf{IdfBy}_1\ y \text{ and } x\ \mathsf{IsGenOf}\ Y \triangleq Y = \{y \mid x\ \mathsf{Gen}_1\ y\}$$

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

$$x\ \mathsf{Spec}_1\ y \triangleq x\ \mathsf{Spec}\ y \wedge x\ \mathsf{IdfBy}_1\ y \text{ and } X\ \mathsf{AreSpecsOf}\ y \triangleq X = \{x \mid x\ \mathsf{Spec}_1\ y\}$$

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

$E := \{S \mid \mathcal{R} \vdash \mathsf{exclusion}(S)\} \cup \{\{x\} \mid x \in \mathcal{O}\}$
while $X \neq \varnothing$ **do**
  let $\{x_1, \ldots, x_n\} \subseteq X$ **such that** $\{x_1, \ldots, x_n\} \in E$ **and** n **is maximal**
  /* Note that the **let** makes a non-deterministic choice */

  draw

  $X := X - \{x_1, \ldots, x_n\}$
od

The expression $\mathcal{R} \vdash \mathsf{exclusion}(S)$ means: the exclusion constraint $\mathsf{exclusion}(S)$ is part of the set of constraints $\mathcal{R}$ defined on the schema. As a result, this algorithm also deals with exclusion constraints, since all output types of a type construction (the types at the top of the triangle) have to be disjoint.

The complete zoo example is now depicted in figure 7 in the ER$^+$ style. Note that we did not depict all names of object types, and connectors, for reasons of clarity.

5 NIAM way of communicating

The techniques, which use a NIAM way of communicating are mostly extensions of the NIAM modelling technique presented in [26]. The Predicator Set Model, which is presented in [22], 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.

5.1 Pure NIAM

In pure NIAM the following ORM concepts can be identified: object types ($\mathcal{O}$), entity types ($\mathcal{E}$), roles or predicators ($\mathcal{P}$), fact types ($\mathcal{F}$), and label types ($\mathcal{L}$). In table 3 the relation between the ORM constructs and NIAM are provided.

5.2 PSM

Not all of the constructs of ORM can be represented using NIAM. Therefore, NIAM was extended with some constructs, resulting in PSM (see [22]). In PSM, all concepts of the ORM kernel can be visualised. The additional concepts are: generalisation, power typing, sequence typing, and schema typing.

In table 4 the PSM symbols corresponding to the ORM constructs are given. The complete zoo example is depicted in figure 8.

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

Table 3: Relation between ORM constructs and pure NIAM symbols

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

Table 4: Relation between the ORM constructs and the PSM symbols

Figure 7: Complete zoo example in ER^+

6 Conclusions

By making a clear distinction between the different aspects of methods, we were able to demonstrate the possibility to built 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. More research in the refinement of the ORM kernel is underway, in the form of joint research between the Universities of Nijmegen and Queensland. Furthermore, the ER view on an ORM model could be further refined and tuned, by applying abstraction mechanisms for ORM schemas as discussed in [9] and [10].

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.

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 [5], [25] and [7]). The main difference 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 on previous versions of this article.

References

[1] S. Abiteboul and R. Hull. IFO: A Formal Semantic Database Model. *ACM Transactions on Database Systems*, 12(4):525–565, December 1987.

[2] D.E. Avison. *Information Systems Development: Methodologies, Techniques and Tools*. McGraw–Hill, New York, New York, USA, 2nd edition, 1995. ISBN: ISBN 0077092333

Figure 8: Complete zoo example in PSM

[3] 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.

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

[5] G.H.W.M. Bronts. Formalization of an Object Model. Master's thesis, University of Nijmegen, Nijmegen, The Netherlands, EU, 1993.

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

[7] S.J. Brouwer. PSM vs 'the rest of the world. Master's thesis, University of Nijmegen, Nijmegen, The Netherlands, EU. In Dutch.

[8] 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, Amsterdam, The Netherlands, EU*, pages 289–318. North–Holland/IFIP WG8.1, Amsterdam, The Netherlands, EU, 1986.

[9] L.J. Campbell and T.A. Halpin. Automated Support for Conceptual to External Mapping. In S. Brinkkemper and F. Harmsen, editors, *Proceedings of the Fourth Workshop on the Next Generation of CASE Tools*, pages 35–51, June 1993.

[10] 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.

[11] P.P. Chen. The Entity–Relationship Model: Towards a Unified View of Data. *ACM Transactions on Database Systems*, 1(1):9–36, March 1976.

[12] R. Elmasri and S.B. Navathe. Advanced Data Models and Emerging Trends. In *Fundamentals of Database Systems*, chapter 21. Benjamin Cummings, Redwood City, California, USA, 1994. Second Edition.

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

[14] 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.

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

[16] 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, July 1992.

[17] T.A. Halpin and M.E. Orlowska. Fact–oriented modelling for data analysis. *Journal of Information Systems*, 2(2):97–119, April 1992.

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

[19] A.H.M. ter Hofstede. *Information Modelling in Data Intensive Domains*. PhD thesis, University of Nijmegen, Nijmegen, The Netherlands, EU, 1993.

[20] A.H.M. ter Hofstede, H.A. (Erik) 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.

[21] A.H.M. ter Hofstede, H.A. (Erik) 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: ISBN 047302313

[22] 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.

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

[24] 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, Berlin, Germany, EU*, pages 58–88, Berlin, Germany, EU, 1987. Springer.

[25] C.L.J. Martens. PSM vs 'the rest of the world. Master's thesis, University of Nijmegen, Nijmegen, The Netherlands, EU. In Dutch.

[26] G.M. Nijssen and T.A. Halpin. *Conceptual Schema and Relational Database Design: a fact oriented approach*. Prentice–Hall, Englewood Cliffs, New Jersey, USA, 1989. ISBN: ASIN 0131672630

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

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

[29] P. Shoval and S. Zohn. Binary–Relationship integration methodology. *Data & Knowledge Engineering*, 6(3):225–250, 1991.