Object-Role Modeling (ORM/NIAM)

Terry Halpin
Microsoft Corporation, USA

[This paper appeared as ch. 4 of *Handbook on Architectures of Information Systems*, eds P. Bernus, K. Mertins & G. Schmidt, Springer-Verlag, Berlin, 1998, and is reproduced here by permission. Details on the book are online at www.springer.de/cgi-bin/search_book.pl?isbn=3-540-64453-9.]

Abstract: Object-Role Modeling (ORM) is a method for modeling and querying an information system at the conceptual level, and mapping between conceptual and logical (e.g. relational) levels. ORM comes in various flavors, including NIAM (Natural language Information Analysis Method). This article provides an overview of ORM, and notes its advantages over Entity Relationship and traditional Object-Oriented modeling.

1 Introduction

1.1 ORM: what is it and why use it?

Object-Role Modeling (ORM) is primarily a method for modeling and querying an information system at the conceptual level. In Europe, the method is often called *NIAM* (Natural language Information Analysis Method). Since information systems are typically implemented on a DBMS that is based on some logical data model (e.g. relational, object-relational, hierarchic), ORM includes procedures for mapping between conceptual and logical levels. Although various ORM extensions have been proposed for process and event modeling, the focus of ORM is on data modeling, since the data perspective is the most stable and it provides a formal foundation on which operations can be defined.

For correctness, clarity and adaptability, information systems are best specified first at the conceptual level, using concepts and language that people can readily understand. Analysis and design involves building a formal model of the application area or *universe of discourse* (UoD). To do this properly requires a good understanding of the UoD and a means of specifying this understanding in a clear, unambiguous way. Object-Role Modeling simplifies this process by using natural language, as well as intuitive diagrams that can be populated with examples, and by expressing the information in terms of elementary relationships.

ORM is so-called because it pictures the world in terms of *objects* (entities or values) that play *roles* (parts in relationships). For example, you are now playing the role of reading, and this paper is playing the role of being read. In contrast to other modeling techniques such as Entity-Relationship (ER) and Object-Oriented (OO) approaches, ORM makes no explicit use of *attributes*. For example, instead of using countryBorn as an attribute of Person, we use the relationship type Person was born in Country. This has many important advantages. Firstly, ORM models and queries are more stable (attributes may evolve into entities or relationships). For example, if we decide to later record the population of a country, then our countryBorn attribute needs to be reformulated as a relationship. Secondly, ORM models may be conveniently populated with multiple instances (attributes make this too awkward). Thirdly, ORM is more uniform (e.g. we don't need a separate notation for applying the same constraint to an attribute rather than a relationship).

ORM is typically more expressive than ER or OO. Its role-based notation makes it easy to specify a wide variety of constraints, and its object types reveal the semantic domains that bind a schema together. One benefit of this is that conceptual queries may now be formulated in terms of schema paths, where moving from one role though an object type to another role amounts to a conceptual join (see later).

Unlike ORM or ER, popular OO models often duplicate information by wrapping facts up into pairs of inverse attributes in different objects. Moreover, OO notations have weak support for constraints (e.g. a constraint might have to be duplicated in different objects, or even ignored). Unfortunately, OO models are less stable than even ER models when the UoD evolves. For such reasons, OO models should be used only for implementation, not for analysis.

Although the detailed picture provided by ORM is desirable in developing and transforming a model, for summary purposes it is useful to hide or compress the display of much of this detail. Various abstraction mechanisms exist for doing this [e.g. CHP96]. If desired, ER and OO diagrams can also be used for providing compact summaries, and are best developed as views of ORM diagrams. For a simple discussion illustrating the points in this section, see [Hal96].

The rest of this article provides a brief history of ORM, summarizes the ORM notation, illustrates the conceptual design and relational mapping procedures, and mentions some recent extensions before concluding.

1.2 A brief history of ORM

In the 1970s, especially in Europe, substantial research was carried out to provide higher level semantics for modeling information systems. Abrial [Abr74], Senko [Sen75] and others discussed modeling with binary relationships. In 1973, Falkenberg generalized their work on binary relationships to n-ary relationships and decided that attributes should not be used at the conceptual level because they involved "fuzzy" distinctions and also complicated schema evolution. Later, Falkenberg proposed the fundamental ORM framework, which he called the "object-role model" [Fal76]. This framework allowed n-ary and nested relationships, but depicted roles with arrowed lines.

Nijssen [Nij76] adapted this framework by introducing the circle-box notation for objects and roles that has now become standard, and adding a linguistic orientation and design procedure to provide a modeling method called ENALIM (Evolving NAtural Language Information Model) [Nij77]. Nijssen led a group of researchers at Control Data in Belgium who developed the method further, including van Assche who classified object types into lexical object types (LOTs) and non-lexical object types (NOLOTs). Today, LOTs are commonly called "Entity types" and NOLOTs are called "Value types". Kent [Ken77] provided several semantic insights and clarified many conceptual issues.

Meersman added subtypes, and made major contributions to the RIDL query language [Mee82] with Falkenberg and Nijssen. The method was renamed "aN Information Analysis Method" (NIAM) and summarized in a paper by Verheijen and van Bekkum [VB82]. In later years the acronym "NIAM" was given different expansions, and is now known as "Natural language Information Analysis Method". Two matrix methods for subtypes were developed, one (the role-role matrix) by Vermeir [Ver83] and another by Falkenberg and others.

In the 1980s, Falkenberg and Nijssen worked jointly on the design procedure and moved to the University of Queensland, where the method was further enhanced by various academics. Halpin provided the first full formalization of the method [Hal89], including schema equivalence proofs, and made several refinements and extensions to the method. In 1989, Halpin and Nijssen co-authored a book on the method. A second edition of this book, authored by Halpin, was published in 1995 [Hal95]. Another book on the method, written by Wintraecken, was published in 1990 [Win90].

Many researchers have contributed to the ORM method over the years, and there is no space here to list them all. Today various versions of the method exist, but all adhere to the fundamental object-role framework. Although most ORM proponents favor n-ary relationships, some prefer Binary-Relationship Modeling (BRM), e.g. Shoval [SS93]. Habrias [Hab93] developed an object-oriented version called MOON (Normalized Object-Oriented Method). The Predicator Set Model (PSM) was developed mainly by ter Hofstede, Proper and van der Weide [HPW93], and includes complex object constructors. De Troyer and Meersman [DM95] developed another version with constructors called Natural Object-Relationship Model (NORM). Halpin developed an extended version called Formal ORM (FORM), and with Bloesch and others at InfoModelers Inc. developed an associated query language called ConQuer [BH97]; this work is being extended at Visio Corporation. Van der Lek and others [BZL94] allowed entity types to be treated as nested roles, to produce Fully Communication Oriented NIAM (FCO-NIAM). Embley and others [EKW92] developed Object-oriented Systems Analysis (OSA) which includes an "Object-Relationship Model" component that has much in common with standard ORM, with no use of attributes.

2 Data modeling in ORM

1.3 Notation

A modeling method includes both a notation and a procedure for using its notation. This subsection discusses notation, and later subsections discuss procedures. Each well-defined version of ORM includes a formal, textual specification language for both models and queries, as well as a formal, graphical modeling language. The textual languages are more expressive than the graphical languages, but are mentioned only briefly in this paper. Figure 1 summarizes most of the main symbols used in the graphical language. We now briefly describe each symbol. Examples of these symbols in use are given later.

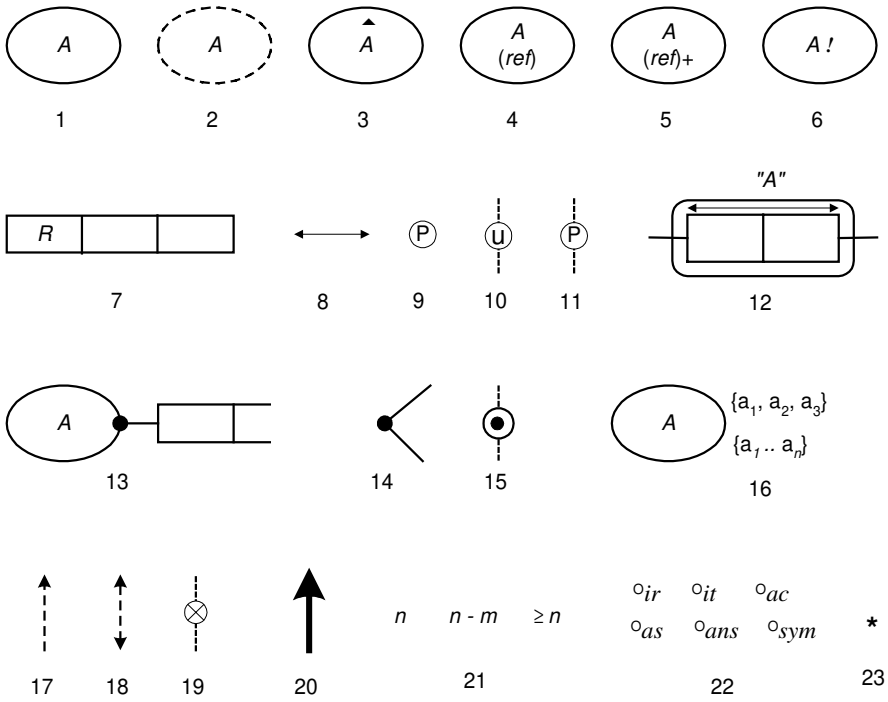


Figure 1 Main ORM symbols

The symbols are numbered for easy reference. An *entity type* is depicted as a named ellipse (symbol 1). A *value type* denotes a lexical object type (e.g. a character string or number) and is usually shown as a named, dotted ellipse (symbol 2). Another notation for value types encloses the value type name in parentheses. Object types that appear more than once in the schema may be tagged with an arrow tip (see symbol 3), that "points" to the existence of another occurrence. Each entity type must have at least one *reference scheme*, which indicates how each instance of the entity type may be mapped via predicates to a combination of one or more values. A simple injective (1:1 into) reference scheme maps entities to single values. For example, each country may be identified by a single country code (e.g. 'USA'). In such cases the reference scheme may be abbreviated as in symbol 4 by displaying the *reference mode* in parentheses beside the name of the entity type, e.g. Country(code). The reference mode indicates how values relate to the entities. Symbol 5 shows that a plus sign "+" may be added if the values are numeric, e.g. Mass(kg)+. Values are constants with a universally understood denotation, and hence require no reference scheme.

Although not strictly a conceptual issue, it is normal to require each entity type to have a *primary* reference scheme. Relationship types used for primary reference are then called *reference types*. The other relationship types are known as *fact types*. In symbol 6, an exclamation mark is added to declare that an entity type is *independent*. This means that instances of that type may exist without participating in any facts. By default, this is not the case (i.e. we don't normally introduce an object into the universe unless it takes part in some fact).

Symbol 7 shows a ternary *predicate*, comprised of three *roles*. Each role is depicted as a box, and must be played by exactly one object type. Roles are connected to their players by a line segment (see symbol 13). A predicate is basically a sentence with object holes in it, one for each role. The number of roles is called the *arity* of the predicate. Except for the BRM version, ORM allows predicates of any arity (1 = unary, 2 = binary, 3 = ternary etc.). Predicates are usually treated as ordered, as in predicate logic. In this case, the name of the predicate is written either in or beside the first role box, and if necessary each object hole may be shown as an ellipsis "…". Different readings may be provided so the information may be read in any direction. FORML allows mixfix predicates so objects may be placed at any position in the predicate. For example, the fact type Room at Time is used for Activity involves the predicate "... at ... is used for ...". Apart from facilitating natural verbalization of n-ary relationships, mixfix predicates allow binary relationships to be verbalized in languages where the verb is not in the infix position (e.g. in Japanese, verbs come at the end). In some versions of ORM, relationship types are given a name, and each role is also given a name, thus making order irrelevant.

Internal uniqueness constraints are depicted as arrow tipped bars (symbol 8), and are placed over one or more roles in a predicate to declare that instances for that role (combination) in the relationship type population must be unique. For example, adding a uniqueness constraint over the first role of Person was born in Country declares that each person was born in at most one country. A predicate may have one or more uniqueness constraints, at most one of which may be declared *primary* by adding a "P" (symbol 9). An *external uniqueness constraint* shown as a circled "u" may be applied to two or more roles from different predicates by connecting to them with dotted lines (symbol 10). This indicates that instances of the combination of those roles in the join of those predicates are unique. For example, to say that a state is identified by combining its statecode and country, we add an external uniqueness constraint to the roles played by Statecode and Country in the reference types: State has Statecode; State is in Country. To declare an external uniqueness constraint primary, use "P" instead of "u" (symbol 11). An object type may have at most one primary reference constraint.

If we want to talk about a relationship type we may *objectify* it (i.e. make an object out of it) so that it can play roles. Graphically, the objectified predicate is enclosed in either a rounded rectangle (symbol 12) or an ellipse, and named. Objectified predicates are also called *nested* object types. Typically the objectified predicate must have a spanning uniqueness constraint, but 1:1 cases may also be allowed [Hal93].

A *mandatory role constraint* declares that every instance in the population of the role's object type must play that role. It is usually shown as a black dot (see symbol 13) but a universal quantifier is sometimes used. Mandatory roles are also called *total* roles. A *disjunctive* mandatory constraint may be applied to two or more roles to indicate that all instances of the object type population must play *at least one* of those roles. This may often be shown by connecting the roles to a black dot on the object type (symbol 14) or in general by connecting the roles by dotted lines to a circled black dot (symbol 15).

To restrict an object type's population to a given list, the relevant values may be listed in braces (symbol 16, top). If the values are ordered, a range may be declared separating the first and last values by ".." (symbol 16, bottom). These constraints are called *value constraints*.

Symbols 17-19 denote *set comparison constraints*, and may only be applied between compatible role sequences (i.e. sequences of one or more roles, where the corresponding roles have the same host object type). A dotted arrow (symbol 17) from one role sequence to another is a *subset constraint*, restricting the population of the first sequence to be a subset of the second. A double-tipped arrow (symbol 18) is an *equality constraint*, indicating the populations must be equal. A circled "X" (symbol 19) is an *exclusion constraint*, indicating the populations are mutually exclusive. Exclusion constraints may be applied between two or more sequences.

A solid arrow (symbol 20) from one object type to another indicates that the first object type is a (proper) *subtype* of the other. For example, Woman is a subtype of Person. Totality (circled black dot) and exclusion (circled "X") constraints may also be displayed between subtypes, but are implied by other constraints if the subtypes are given formal definitions.

Symbol 21 shows three kinds of *frequency constraint*. Applied to a sequence of one or more roles, these indicate that instances that play those roles must do so exactly *n* times, between *n* and *m* times, or at least *n* times.

Symbol 22 shows six kinds of *ring constraint*, that may be applied to a pair of roles played by the same host type. These indicate that the binary relation formed by the role population must be irreflexive (ir), intransitive (it), acyclic (ac), asymmetric (as), antisymmetric (ans) or symmetric (sym).

Symbol 23 is an asterisk "*", which may be placed beside a fact type to indicate that it is derivable from other fact types. Not all versions of ORM support all these symbols, and some versions have a few more symbols. InfoModeler, a popular ORM tool, supports all of the symbols shown, as will a future release of Visio Professional.

1.4 Conceptual schema design procedure

The information systems life cycle typically involves several stages: feasibility study; requirements analysis; conceptual design of data and operations; logical design; external design; prototyping; internal design and implementation; testing and validation; and maintenance. ORM's *conceptual schema design procedure* (CSDP) focuses on the analysis and design of data. The conceptual schema specifies the information structure of the application: the *types of fact* that are of interest; *constraints* on these; and perhaps *derivation rules* for deriving some facts from others. With large applications, the UoD is divided into convenient modules, the CSDP is applied to each, and the resulting subschemas are integrated into the global conceptual schema.

Table 1 shows the CSDP used in FORM. Although different versions of the CSDP exist, they all agree on the importance of verbalization in terms of elementary facts, population checks, and thorough analysis of business rules. The rest of this section illustrates the basic working of this design procedure by means of an example. Because of space limitations, our treatment is necessarily brief. A much more detailed discussion of the same example can be electronically accessed from [Hal97].

Table 1 The conceptual schema design procedure (CSDP)

Step
1. Transform familiar information examples into elementary facts, and apply quality checks.
2. Draw the fact types, and apply a population check.
3. Check for entity types that should be combined, and note any arithmetic derivations.
4. Add uniqueness constraints, and check arity of fact types.
5. Add mandatory role constraints, and check for logical derivations.
6. Add value, set comparison and subtyping constraints.
7. Add other constraints and perform final checks.

Step 1 is the most important. Examples of the information required from the system are verbalized in natural language. Such examples are often available in the form of output reports or input forms, perhaps from a current manual version of the required system. If not, the modeler can work with the client to produce examples. To avoid misinterpretation, a UoD expert (a person familiar with the application) should perform or at least check the verbalization. As an aid to this process, the speaker imagines he/she has to convey the information contained in the examples to a friend over the telephone.

For our case study, we consider a fragment of an information system used by a university to maintain details about its academic staff and academic departments. One function of the system is to print an academic staff directory, as exemplified by the report extract shown in Table 2. Part of the modeling task is to clarify the meaning of terms used in such reports. The descriptive narrative provided here would thus normally be derived from a discussion with the UoD expert. The terms “empnr” and “extnr” abbreviate “employee number” and “extension number”.

A phone extension may have access to local calls only (“LOC”), national calls (“NAT”), or international calls (“INT”). International access includes national access, which includes local access. In the few cases where different rooms or staff have the same extension, the access level is the same. An academic is either tenured or on contract. Tenure guarantees employment until retirement, while contracts have an expiry date.

Table 2 Extract from a directory of academic staff

				Phone		*Tenured/*
Empnr	*EmpName*	*Dept*	*Room*	*Extnr*	*Access*	*Contract-expiry*
715	Adams A	Computer Science	69-301	2345	LOC	01/31/95
720	Brown T	Biochemistry	62-406	9642	LOC	01/31/95
139	Cantor G	Mathematics	67-301	1221	INT	tenured
430	Codd EF	Computer Science	69-507	2911	INT	tenured
503	Hagar TA	Computer Science	69-507	2988	LOC	tenured
651	Jones E	Biochemistry	69-803	5003	LOC	12/31/96
770	Jones E	Mathematics	67-404	1946	LOC	12/31/95
112	Locke J	Philosophy	1-205	6600	INT	tenured
223	Mifune K	Elec. Engineering	50-215A	1111	LOC	tenured
951	Murphy B	Elec. Engineering	45-B19	2301	LOC	01/03/95
333	Russell B	Philosophy	1-206	6600	INT	tenured
654	Wirth N	Computer Science	69-603	4321	INT	tenured
...	...	...	...	...	...	...

The information contained in this Table is to be stated in terms of *elementary facts*. Basically, an elementary fact asserts that a particular object has a property, or that one or more objects participate in a relationship, where that relationship cannot be expressed as a conjunction of simpler (or shorter) facts without introducing new object types [Hal93]. For example, to say that Bill Clinton jogs and is the president of the USA is to assert two elementary facts.

As a first attempt, one might read off the information on the first data row as the six facts f1-f6. Each asserts a binary relationship between two objects. For discussion purposes the *predicate* is shown in **bold** between the noun phrases that identify the objects, and object type names start with a capital letter. Some obvious abbreviations are used ("empnr", "EmpName", "Dept", "extnr"); when read aloud these can be expanded to "employee number", "Employee name", "Department" and "extension number". The second data row contains different instances of these six fact types. Row three, because of its final column, provides an instance f7 of a seventh fact type, a unary.

f1 The Academic with empnr 715 **has** EmpName 'Adams A'.
f2 The Academic with empnr 715 **works for** the Dept named 'Computer Science'.
f3 The Academic with empnr 715 **occupies** the Room with roomnr '69-301'.
f4 The Academic with empnr 715 **uses** the Extension with extnr '2345'.
f5 The Extension with extnr '2345' **provides** the AccessLevel with code 'LOC'.
f6 The Academic with empnr 715 **is contracted till** the Date with mdy-code '01/31/95'.
f7 The Academic with empnr 139 **is tenured**.

Different readings may be provided to allow relationships to be read in different directions. For example, the *inverse* reading of f4 is: The Extension with extnr '2345' **is used by** the Academic with empnr 715. To save writing, both the normal predicate and its inverse may be included in the same declaration, with the inverse predicate preceded by a slash "/". For example:

f4' The Academic with empnr 715 **uses /is used by** the Extension with extnr '2345'.

Predicate names are usually unique in the conceptual schema. In some cases (e.g. "has"), the same name may be used externally for different predicates: internally these have different identifiers.

As a quality check at Step 1, we ensure that objects are well identified. Values are identified by constants (e.g. 'Adams A', 715). *Entities* are "real world" objects that are identified by a definite description (e.g. the Academic with empnr 715). Fact f1 involves a relationship between an entity (a person) and a value (a name is just a character string). Facts f2-f6 specify relationships between entities. Fact f7 states a property (or unary relationship) of an entity.

As a second quality check at Step 1, we use our familiarity with the UoD to see if some facts should be split or recombined (a formal check on this is applied later). For example, suppose facts f1 and f2 were verbalized as: The Academic with empnr 715 and empname 'Adams A' **works for** the Dept named 'Computer Science'. The presence of the word "and" suggests that this may be split without information loss. The repetition of "Jones E" on different rows of Table 2 shows that academics cannot be identified just by their name. However the uniqueness of empnr in the sample population suggests that this suffices for reference. Since the "and-test" is only a heuristic, and sometimes a composite naming scheme is required for identification, the UoD expert is consulted to verify that empnr by itself is sufficient for identification. With this assurance obtained, the composite sentence is now split into f1 and f2.

As an alternative to specifying complete facts one at a time, the reference schemes may be declared up front and then assumed in later facts. For example, suppose we declare the following: Academic(empnr); EmpName(); Dept(name). The empty parentheses after EmpName indicates it is a value type and hence needs no reference scheme. Now facts f1 and f2 may be stated as: Academic 715 has EmpName 'Adams A'; Academic 715 works for Dept 'Computer Science'. Facts f1-f7 are instances of the following fact types:

F1 Academic has EmpName
F2 Academic works for Dept
F3 Academic occupies Room
F4 Academic uses Extension
F5 Extension provides AccessLevel
F6 Academic is contracted till Date
F7 Academic is tenured

Step 2 of the CSDP is to *draw a draft diagram of the fact types and apply a population check* (see Figure 2). As a check, each fact type has been populated with at least one fact, shown as a row of entries in the associated fact table, using the data from rows 1 and 3 of Table 2. The English sentences listed before as facts f1-f7, as well as other facts from row 3, may be read directly off this figure. Though useful for validating the model with the client and for understanding constraints, the sample population is not part of the conceptual schema itself.

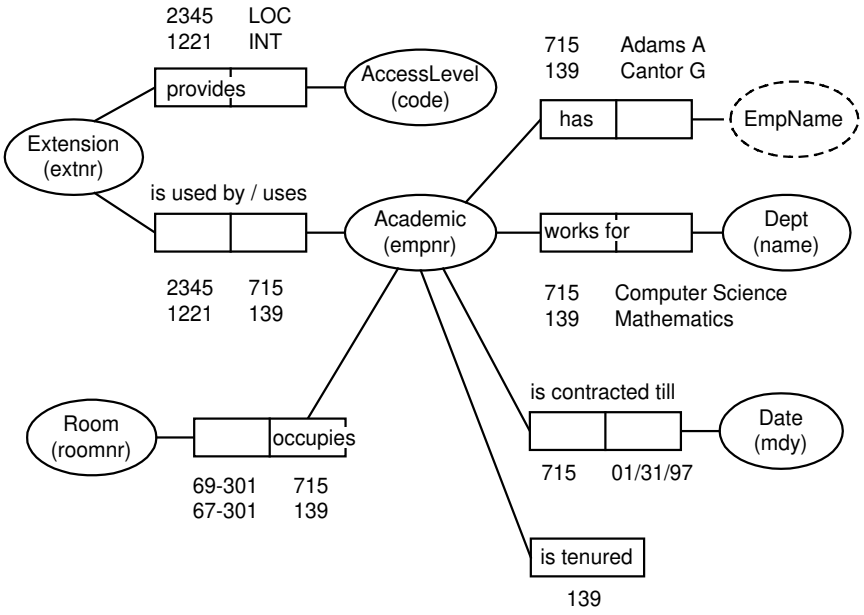


Figure 2 Draft diagram of fact types for Table 2 with sample population

Suppose the information system is also required to assist in the production of departmental handbooks. Figure 3 shows an extract from a page of one such handbook. In this university academic staff are classified as professors, senior lecturers or lecturers, and each professor holds a "chair" in a research area. To reduce the size of our problem, we have excluded many details that in practice would also be recorded (e.g. office phone and fax). To save space, details are shown here for only four of the 22 academics in that department. The data are, of course, fictitious.

Department: Computer Science
Home phone of Dept head: 9765432

Chairs *Professors (5)*

Databases Codd EF BSc (UQ); PhD (UCLA) *(Head of Dept)*
Algorithms Wirth N BSc (UQ); MSc (ANU); DSc (MIT)
…

Senior Lecturers (9)

Hagar TA BInfTech (UQ); PhD (UQ)
…

Lecturers (8)

Adams A MSc (OXON)
…

Figure 3 Extract from Handbook of Computer Science Department

It appears from the handbook example that within a single department, academics may be identified by their name. Let us assume this is verified by the UoD expert. However the complete application requires us to handle all departments in the same information system, and to integrate this subschema with the directory subschema considered earlier. Hence we must replace the academic naming convention used for the handbook example by the global scheme used earlier (i.e. empnr).

We use this report to illustrate *Step 3* of the CSDP: *check for entity types that should be combined, and note any arithmetic derivations*. Suppose we verbalized the degree information in terms of the three

ternary fact types: Professor obtained Degree from University; SeniorLecturer obtained Degree from University; Lecturer obtained Degree from University. The common predicate suggests that the entity types Professor, SeniorLecturer and Lecturer should be collapsed to the single entity type Academic, with this predicate now shown only once. To preserve the original information about who is a professor, senior lecturer or lecturer we introduce the fact type: Academic has Rank. Let's use the codes "P", "SL" and "L" for the ranks of professor, senior lecturer and lecturer.

The second aspect of Step 3 is to see if some fact types can be derived from others by arithmetic. Since we now record the rank of academics as well as their departments, we can compute the number in each rank in each department simply by counting. So the fact type Dept employs academics of Rank in Quantity is *derivable*. If desired, derived fact types may be included on a schema diagram if they are marked with an asterisk "*". At any rate, a *derivation rule* must be supplied. This may be written below the diagram (see Figure 4). Here "**iff**" abbreviates "if and only if".

Step 4 of the CSDP is to *add uniqueness constraints and check the arity of the fact types*. For example, we add a uniqueness constraint to the first role of works for to ensure each academic works for at most one department. An arity check ensures each uniqueness constraint on an n-ary spans at least n-1 roles.

Step 5 of the CSDP is to *add mandatory role constraints, and check for logical derivations*. For example, we need a disjunctive mandatory constraint to declare that each academic *either* is contracted till some date *or* is tenured. Roles that are not mandatory are *optional*. If an object type plays only one fact role in the global schema, then by default this is mandatory, but a dot is not normally shown.

Suppose that departmental handbooks include a building directory, which lists the names as well as the numbers of buildings. A sample fact might be: Building '67' has Buildingname 'Priestly'. Earlier we identified rooms by a single value. For example "67-301" was used to denote the room in building 67 which has room number "301". Now that buildings are to be talked about in their own right, we replace the simple reference scheme by a composite one that shows the full semantics (see Figure 4). Here Roomnr now means just the number (e.g. "301") used to identify the room within its building.

To illustrate nesting, suppose the application requires reports about teaching commitments, an extract of which is shown in Table 3. Not all academics currently teach. If they do, their teaching in one or more subjects may be evaluated and given a rating. Some teachers serve on course curriculum committees. Here the new fact types may be schematized as shown in Figure 4. The nested object type Teaching plays only one role, and this role is optional, so Teaching is an *independent* object type (as shown by the "!").

Table 3 Extract of report on teaching commitments

Empnr	*Emp. name*	*Subject*	*Rating*	*Committees*
715	Adams A	CS100 CS101	5	
430	Codd EF			
654	Wirth N	CS300		BSc-Hons CAL Advisory

The second stage of Step 5 is to check for *logical derivations* (i.e. can some fact type be derived from others without the use of arithmetic?). One strategy here is to ask whether there are any relationships (especially functional relationships) which are of interest but which have been omitted so far. Another strategy is to look for transitive patterns of functional dependencies. Suppose that our client confirms that the rank of an academic determines the access level of his/her extension. For example, suppose a current business rule is that professors get international access while lecturers and senior lecturers get local access. This rule might change in time (e.g. senior lecturers might be arguing for national access). To minimize later changes to the schema, we store the rule as data in a table. So it can be updated as required by an authorized user without recompiling the schema. The relevant rule is shown at the bottom of Figure 4.

In *Step 6* of the CSDP we add *any value, set comparison and subtyping constraints*. One value constraint is that Rankcode is restricted to {'P','SL','L'}. In Figure 4, a pair-subset constraint runs from the heads predicate to the works for predicate, indicating that a person who heads a department must work for the same department. The rule that nobody can be tenured and contracted at the same time is captured by an exclusion constraint. Subtyping is determined as follows. Each optional role is inspected: if the role is played only by some well-defined subtype, a subtype node is introduced with this role attached.

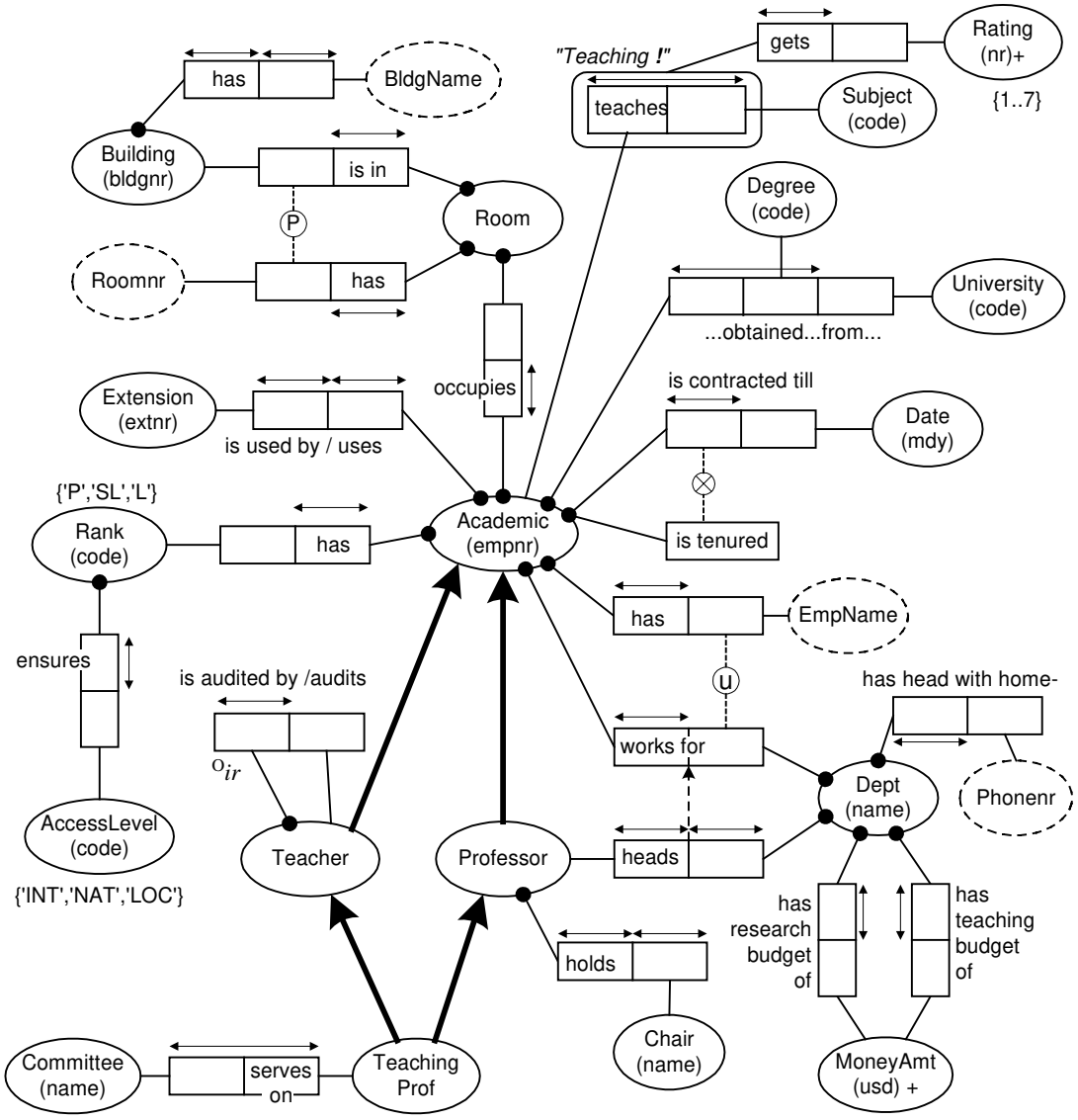


Figure 4 The final conceptual schema

Subtype links and definitions are added. Figure 4 contains three subtypes: Teacher; Professor; and TeachingProfessor. In this university, each teacher is audited by another teacher. Moreover, only professors may be department heads, and only teaching professors can serve on curriculum committees (not all universities work this way).

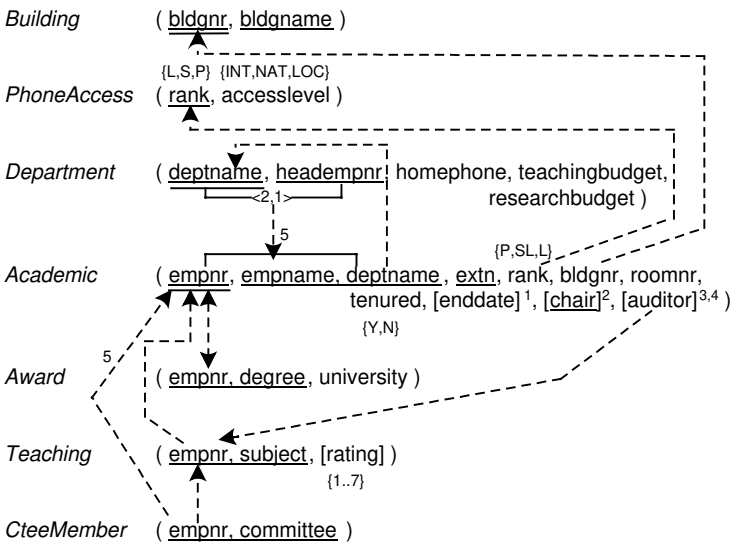
Step 7 of the CSDP adds other constraints and performs final checks. For example, auditing is *irreflexive* (no teacher audits himself/herself). Suppose we also need to record the teaching and research budgets of the departments. We might schematize this using the ternary Dept has for Activity a budget of MoneyAmt, where Activity has the value constraint {'Teaching', 'Research'} and the first role is mandatory and constrained to a *frequency* of 2.

Once the global schema is drafted, and the target DBMS decided, some optimization can often be performed to improve the efficiency of the logical schema obtained by mapping. Assuming the conceptual schema is to be mapped to a relational database schema, the ternary fact type about budgets will map to a separate table all by itself, leading to extra joins for some queries. We can avoid this problem by transforming the ternary into the following two binaries before we map: Dept has teaching budget of MoneyAmt; Dept has research budget of MoneyAmt. These binaries have simple keys, and will map to the "main" department table. Another optimization may be performed which moves the home phone information to Dept instead of Professor. Figure 4 includes these optimizations. Such *conceptual schema transformations* require a rigorous theory of schema equivalence and optimization strategies. For details on such topics, see [Hal95, ch. 9; HP95b; DeT93].

2.3 Logical Mapping

Once the conceptual schema has been specified, the target data model is selected and the mapping is done. For example, the Rmap algorithm [RH93; Hal95] maps our conceptual schema to the relational schema shown in Figure 5 (domains omitted). If the conceptual fact types are elementary (as they should be), then the mapping is guaranteed to be free of redundancy, since each fact type is grouped into only one table, and fact types which map to the same table all have uniqueness constraints based on the same attribute(s).

Keys are underlined. If alternate keys exist, the primary key is doubly-underlined. A mandatory role is captured by making its corresponding attribute mandatory in its table (not null is assumed by default), by marking as optional (in square brackets) all optional roles for the same object type which map to the same table, and by running an equality/subset constraint from those mandatory/optional roles which map to another table. The ⟨2,1⟩ in the pair-subset constraint indicates the source pair should be reversed before the comparison. Subtyping is captured by qualified optionals or qualified subset constraints. The word "exists" means "a non-null value exists".



[1] **exists iff** tenured = 'N'
[2] **exists iff** rank = 'P'
[3] <> empnr
[4] **exists iff** empnr **in** Teaching.empnr
[5] **only where** rank = 'P'

* *Provides* (extn, accesslevel) ::= extn, accesslevel **from**
Academic **natural join** PhoneAccess

* *Employs* (deptname, rank, nrstaff) ::= deptname, rank, **count**(*)
from Academic
group by deptname, rank

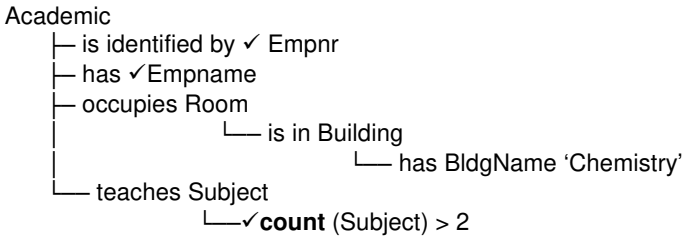
Figure 5 The relational schema mapped from Figure 4

3 Recent extensions

3.1 Conceptual queries

Besides information modeling, ORM is also ideal for information querying. The first significant ORM query language was RIDL [Mee82], a hybrid language with both declarative and procedural components. Temporal aspects were added later to form TRIDL. Currently, research is being carried out on at least three ORM query languages: LISA-D [HPW93]; OSM-QL [EWPC96]; and ConQuer [BH96]. Of these ConQuer (CONceptual QUERy) is the only one to be commercially released. A more powerful version, ConQuer-II [BH97], is currently under development at Visio Corporation.

Using ConQuer, an ORM model may be queried directly without prior knowledge of either the conceptual schema or the corresponding relational schema, by dragging object types onto the query pane, selecting predicates of interest, applying restrictions and functions as desired, and ticking the items to be listed. As a simple example, consider the following English query on our academic database: list the empnr, empname and number of subjects taught for each academic who occupies a room in the Chemistry building and teaches more than two subjects. This may be formulated by drag-and-drop basically as follows:



Notice how easily the conceptual joins are made. A verbalization of the query is automatically generated, as well as SQL code. Formulating queries in terms of objects and predicates is much easier than deciphering the semantics of the relational schema and coding in SQL or QBE. A major benefit of such queries is their semantic stability. For example, ConQuer queries are unaffected by most schema changes (e.g. addition of fact types, or changes to constraints). In contrast, such changes often require the corresponding SQL or ER query to be reformulated, since they depend on attribute structures.

3.2 Other extensions

Researchers are actively investigating several extensions to the basic ORM framework. These include abstraction mechanisms to allow users to control the amount of detail seen at any given time [CHP96], reverse engineering [SS93; CH94], support for complex objects [HW93; DM95], process-event modeling [Hof93], external schema generation [CH93], schema evolution [Pro94], schema optimization [HP95b; Bom94], meta-modeling [FO94], subtype extensions [HP95a], null handling [HR92], object-oriented mapping [ME96], unary nesting [BZL94], and empirical research [Eve94].

Although various versions of ORM have added support for complex objects, they differ in their approaches. Currently there seems to be a growing agreement that constructors (e.g. set, bag, sequence) should only be added after a "flat" ORM model is first developed. There are also different opinions on whether such constructors should be considered part of the conceptual model, or regarded as mapping annotations. Commercial developers of ORM tools are also extending the method. For example, InfoModeler includes extra constructs for mapping to object-relational databases, and extensions of this technology are being incorporated into future Visio products.

4 Conclusion

This article has provided only a brief sketch of the ORM method, emphasizing its fundamental features and touching on some of its advantages. Apart from its sound theoretical basis, the method has been used successfully in many countries, on applications from the small to the very large. The recent emergence of intuitive and powerful ORM tools has led to wider adoption of the method, which is now being successfully taught as early as high school level. Perhaps the greatest strengths of ORM are that it lifts the communication between modeler and client to a level where they can readily understand and validate the

application model using simple sentences, and that it has been designed from the ground up to facilitate schema evolution. This second advantage is very relevant to today's business world where change is ongoing.

In an article this brief, several aspects of ORM have necessarily been glossed over. The reader who is interested in pursuing the area further should consult the cited references, which are included at the end of the handbook.

References

[Abr74] Abrial, J.R. 1974, 'Data Semantics', *Data Base Management*, eds J.W. Klimbie and K.L. Koffeman, North-Holland, Amsterdam, The Netherlands, pp. 1-60.

[BZL94] Bakema, G.P., Zwart, J.P.C. & Lek, H. van der 1994, 'Fully Communication Oriented NIAM', *NIAM-ISDM 1994 Conf. Working papers*, eds G.M. Nijssen & J. Sharp, Albuquerque, NM USA, pp. L1-35.

[BH96] Bloesch, A.C. & Halpin, T.A. 1996, 'ConQuer: a conceptual query language', *Proc. ER'96: 15th Int. Conf. on conceptual modeling*, Springer LNCS, vol. 1157, pp. 121-33.

[BH97] Bloesch, A.C. & Halpin, T.A. 1997, 'Conceptual queries using ConQuer-II', *Proc. ER'97: 16th Int. Conf. on conceptual modeling*, Springer LNCS, vol. 1331, pp. 113-26.

[Bom94] Bommell, P. van 1994, 'Implementation selection for Object-Role models', *Proc. First Int. Conf. On Object-Role Modeling (ORM-1)*, eds T.A. Halpin & R.M. Meersman, Magnetic Island, Australia, pp. 103-12.

[CH93] Campbell, L. & Halpin, T.A. 1993, 'Automated Support for Conceptual to External Mapping', *Proc. 4th Workshop on Next Generation CASE Tools*, eds S. Brinkkemper & F. Harmsen, Univ. Twente Memoranda Informatica 93-32, pp. 35-51, Paris (June).

[CH94] Campbell, L. & Halpin, T.A. 1994, 'The reverse engineering of relational databases', *Proc. 5th Workshop on Next Generation CASE Tools*, Utrecht (June).

[CHP96] Campbell, L.J., Halpin, T.A. & Proper, H.A.1996 'Conceptual Schemas with Abstractions: making flat conceptual schemas more comprehensible', *Data and Knowledge Engineering,* vol. 20, no. 1, pp. 39-85.

[DeT93] De Troyer, O. 1993, 'On data schema transformations', PhD thesis, University of Tilburg (K.U.B.), Tilburg, The Netherlands.

[DM95] De Troyer, O. & Meersman, R. 1995, 'A logic framework for a semantics of object oriented data modeling', *OOER'95: Object-Oriented and Entity-Relationship Modeling*, Springer LNCS, vol. 1021, pp. 238-49.

[EKW92] Embley, D.W., Kurtz, B.D. & Woodfield, S.N. 1992, *Object-Oriented Systems Analysis*, Prentice Hall, Englewood Cliffs, NJ.

[EWPC96] Embley, D.W., Wu, H.A., Pinkston, J.S. & Czejdo, B. 1996, 'OSM-QL: a calculus-based graphical query language', Tech. Report, Dept of Comp. Science, Brigham Young Univ., Utah.

[Eve94] Everest, G. 1994, 'Experiences teaching NIAM/OR modeling', *NIAM-ISDM 1994 Conf. Working Papers*, eds G.M. Nijssen & J. Sharp, Albuquerque, NM USA, pp. N1-26.

[Fal76] Falkenberg, E.D. 1976, 'Concepts for modelling information', *Proc. 1976 IFIP Working Conf. on Modelling in Data Base Management Systems*, ed. G.M. Nijssen, Freudenstadt, Germany, North-Holland Publishing, pp. 95-109

[FO94] Falkenberg, E.D. & Oei, J.L.H. 1994, 'Meta-model hierarchies from an Object-Role Modeling perspective', *Proc. First Int. Conf. On Object-Role Modeling (ORM-1)*, eds T.A. Halpin & R.M. Meersman, Magnetic Island, Australia, pp. 218-227.

[Hab93] Habrias, H. 1993, 'Normalized Object Oriented Method', in *Encyclopedia of Microcomputers*, vol. 12, Marcel Dekker, New york, pp. 271-85.

[Hal89] Halpin, T.A. 1989, 'A Logical Analysis of Information Systems: static aspects of the data-oriented perspective', PhD thesis, University of Queensland.

[Hal93] Halpin, T.A. 1993, 'What is an elementary fact?', *Proc. First NIAM-ISDM Conf.*, eds G.M. Nijssen & J. Sharp, Utrecht, (Sep), 11 pp.

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

[Hal96] Halpin, T.A. 1996, 'Business Rules and Object-Role Modeling', *Database Prog. & Design*, vol. 9, no. 10, Miller Freeman, San Mateo CA, pp. 66-72.

[Hal97] Halpin, T.A. 1997, 'Object-Role Modeling: an overview', electronic paper available on website www.orm.net.

[HP95a] Halpin, T.A. & Proper, H.A. 1995a, 'Subtyping and polymorphism in Object-Role Modeling', *Data and Knowledge Engineering*, vol. 15, pp. 251-81, Elsevier Science.

[HP95b] Halpin, T.A. & Proper, H.A. 1995b, 'Database schema transformation and optimization', *OOER'95: Object-Oriented and Entity-Relationship Modeling*, Springer LNCS, vol. 1021, pp. 191-203.

[HR92] Halpin, T.A. & Ritson, P.R. 1992, 'Fact-Oriented Modelling and Null Values', *Proc. 3rd Australian Database Conf.*, eds. B. Srinivasan & J. Zeleznikov, World Scientific, Singapore.

[Hof93] Hofstede, A.H.M. ter 1993, 'Information modelling in data intensive domains', PhD thesis, University of Nijmegen, The Netherlands.

[HPW93] Hofstede, A.H.M. ter, Proper, H.A. & Weide, th.P. van der 1993, 'Formal definition of a conceptual language for the description and manipulation of information models', Information Systems, vol. 18, no. 7, pp. 489-523.

[HW93] Hofstede A.H.M. ter & Weide, th.P. van der 1993, 'Expressiveness in conceptual data modelling', *Data and Knowledge Engineering,* vol. 10, no. 1, pp. 65-100.

[Ken77] Kent, W. 1977, 'Entities and relationships in Information', *Proc. 1977 IFIP Working Conf. on Modelling in Data Base Management Systems*, ed. G.M. Nijssen, Nice, France, North-Holland Publishing, pp. 67-91.

[Mee82] Meersman, R. 1982, 'The RIDL conceptual language', Research report, Int. Centre for Information Analysis Services, Control Data Belgium, Brussels.

[ME96] Mok, W.Y & Embley, D.W. 1996, 'Transforming conceptual model to object-oriented database designs: practicalities, properties and peculiarities', *Proc. ER'96: 15th Int. Conf. on conceptual modeling*, Springer LNCS, vol. 1157, pp. 309-24.

[Nij76] Nijssen, G.M. 1976, 'A gross architecture for the next generation database management systems', *Proc. 1976 IFIP Working Conf. on Modelling in Data Base Management Systems*, ed. G.M. Nijssen, Freudenstadt, Germany, North-Holland Publishing, pp. 1-24.

[Nij77] Nijssen, G.M. 1977, 'Current issues in conceptual schema concepts', *Proc. 1977 IFIP Working Conf. on Modelling in Data Base Management Systems*, ed. G.M. Nijssen, Nice, France, North-Holland Publishing, pp. 31-66.

[Pro94] Proper, H.A. 1994, 'A theory of conceptual modelling of evolving application domains', PhD thesis, University of Nijmegen, The Netherlands.

[RH93] Ritson, P.R. & Halpin, T.A. 1993, 'Mapping Integrity Constraints to a Relational Schema', *Proc. 4th ACIS*, Brisbane (Sep.), pp. 381-400.

[Sen75] Senko, M.E. 1975, 'Information systems: records, relations, sets, entities and things', *Information Systems*, vol. 1, no. 1, Jan. 1995, Pergamon Press, pp. 3-13.

[SS93] Shoval, P. & Shreiber, N. 1993, 'Database reverse engineering: from the relational to the binary relational model', *Data and Knowledge Engineering,* vol. 10, pp. 293-315.

[VB82] Verheijen, G.M.A. & van Bekkum, J. 1982, 'NIAM: an information analysis method', *Information systems Design Methodologies: a comparative review, Proc. IFIP WG8.1 Working Conf.*, Noordwijkerhout, The Netherlands, North Holland Publishing, pp. 537-90.

[Ver83] Vermeir, D. 1983, 'Semantic hierarchies and abstractions in conceptual schemata', *Information systems*, vol. 8, no. 2, pp. 117-24.

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