The OO-Binary Relationship Model : A Truly Object Oriented Conceptual Model *

O. De Troyer
Tilburg University
Infolab
P.O. Box 90153
5000 LE Tilburg
The Netherlands

Abstract

Conventional conceptual models like the Binary Relationship Model (also known as NIAM) or the Entity-Relationship Model do not fit well with the promising object oriented database systems. In this paper we show that is possible to turn such a conventional conceptual model (in particular the Binary Relationship Model) into a truly object oriented conceptual model which combines the assets of the conventional model with the advantages of the object oriented approach.

1. Introduction

In recent years there has been a substantial influence of object-oriented (OO) programming languages to the Data Base Management System (DBMS) technology. This has resulted in the development of a number of object-oriented data models and systems [e.g. 1, 2, 3, 4, 9, 21, 29]. An object-oriented modelling method offers a number of important advantages, such as:

(1) All information is modelled through a single concept, namely **objects**. Each object has its own identity.
(2) The **encapsulation** mechanism that packs the data (**instance variables**) and behaviour (**methods**) of an object together, protects the data from corruption by other objects and hide low level implementation details from the rest of the system (**information hiding**).
(3) Similar objects are grouped together into **classes**. All objects belonging to the same class are described by the same instance variables and methods.
(4) The concept of class allows to define new **abstract data types** which enjoys the same rights and privileges of the built-in data types.
(5) Classes may be organized into **subclass hierarchies**. The instance variables and the methods specified for a class are inherited by all of its subclasses. In addition, the subclasses may have properties (instance variables as well as methods) of their own.

* This work was supported by the Basic Research Action IS-CORE of the Commission of the European Communities.

This generalization/specialization mechanism allows to abstract away the detailed differences of several class descriptions and to factor out the commonalities as a common superclass. The technique of **inheritance** makes it unnecessary to restate the properties of the superclass for the subclasses. In addition it will localize modifications due to changing requirements.

(6) Through the technique of **overloading** systems are simplified because the same method name may be used to activate an operation common to different types of objects but differently implemented for each type of object. It also makes the system easier to extend; if a new type of object, also bearing this operation is added, the existing objects are totally unaffected by this change.
A special case of overloading is **overriding**. This technique redefines a method of a class in a subclass. It allows special cases to be handled more efficiently and easily without having an impact on other objects.

(7) Objects can also contain other objects. Such objects are usually called **composite objects**. They are important because they can represent far more sophisticated structures than simple objects can.

On the other hand, **conceptual data models** (sometimes also called **semantic models**) are used during the information analysis stage in the development of an information system (IS) to translate the user's information requirements into a supposedly precise, complete and unambiguous description, the so-called **conceptual schema** (CS). One of the important functions of such a conceptual schema is to serve as a tool to communicate among all parties involved, the semantics of the Universe of Discourse (the application domain containing the given situation), abbreviated UoD. Therefore this description should ideally only deal with conceptual issues; i.e. the various object types of the UoD, their associations and the integrity constraints. No representation or implementation oriented details should be included. Looking too early into problems such as organization of the UoD objects into records, normalization, access strategies, representations, etc., distracts the specifiers' attention from mastering the real UoD problems. Well known examples of early conceptual models are Entity-Relationship models [e.g. 8], Binary Relationship Model [e.g. 26, 32], Object-Role Model [e.g. 14], Functional Model [e.g. 28].
Later on in the development of an information system, (at least for conventional DBMSs) the CS may become input to the design of a data base schema, which is (was) usually expressed in terms of record types, key fields and relationships between records. Until now, there is no clear connection of a conceptual schema to an OO-DBMS. It may even be questioned if a connection to an OO-system is possible and opportune. Some people will even argue that OO-data models makes conceptual models completely redundant .

Although most of the earlier mentioned conceptual models have already incorporated some of the attractive features of the OO-approach (e.g. subtype hierarchies together with inheritance of properties and aggregation hierarchies) [e.g. 18, 35] there is still a big gap between these conceptual models and OO-models. The main reasons for this are:

(1) Most of these conventional conceptual models do not support the specification of the behaviour of the IS, and if they support it there is usually a clear separation between the description of the information structure (the data) and the functionality (behaviour) of the IS. In OO-approaches specification of data and behaviour is inextricably bound.
(2) Conventional models describe the UoD as a bunch of entities and their relationships and interactions are described from a global viewpoint. The application programs are described in a top down way. The approach is usually based on functional decomposition- and modular programming techniques. This is completely different in the OO-approach. The OO-approach does not start with the tasks or functions to be performed by the system, but rather with the objects that are needed in order to perform the tasks. Once these objects are correctly represented, they can be used to solve a wide variety of tasks including the original one. Objects are specified as general-purpose building blocks and their relationships and interactions with other objects are described locally. Systems are built by assemble objects. In this sense, the OO-approach is rather a bottom up approach.
(3) OO-systems usual offers a large range of new data types like e.g. sets, arrays, bitmaps,... together with the possibility to define others. These are needed because of the complexity of the data of non-conventional information systems. Conventional conceptual models are not able to support the specification (at an abstract level) of the use or definition of such data types.

Although in a recent paper [20] Maier argues that it is not possible to define a general object oriented data model because of the diversity of the features supported by the current available OO-DBMSs, it is our opinion that even in an OO-DBMS environment a conceptual schema is still desirable. Its function remains the same as in a conventional environment; to serve as a specification reference by giving an unambiguous description of the conceptual issues of the system which can also be understood by non-technical persons. The graphical representation technique used to represent the specifications plays an important role in the success of conceptual models as a communication tool between information analysts , database experts and non-technical persons.
Only, the earlier mentioned conceptual models as such cannot be combined with an OO-data model without sacrifying a number of the advantages of the OO-approach.
In this paper we show that it is possible to turn a conventional conceptual model (the Binary Relationship Model, also known as NIAM) into an OO-conceptual model without sacrifying the main assets of the model. The step from an OO-conceptual schema to an OO-implementation will then be much easier and the full power of the OO-approach can then be exploited. We have opted for the Binary Relationship Model (BRM) because it has a number of abstraction capabilities which are in particular suitable for conceptual information analysis.

The paper is organized as follows: In section 2 we briefly describe the BR model. Section 3 presents our object oriented version of the BR model and in section 4 we

summarize differences and similarities between the conventional and the OO-BR model and discuss the difference with other approaches.

2. The Binary Relationship Model (BR model)

The descriptions of the BR model (under different names) appear in several forms in the literature ([12, 16, 19, 22, 24, 25, 26, 32, 34 ...]).The Binary Relationship Model has proven to be successful in several large scale industrial projects [e.g. 13, 31].

Its main abstraction capabilities are :

(1) Abstraction from lexical representation.
The objects of the UoD are classified into lexical and non-lexical objects. Examples of non-lexical objects are employees, departments. The lexical objects are objects which may be used to represent the non-lexical objects of the UoD; e.g. employee names, employee numbers, department names. Although there is an explicit distinction between lexical and non-lexical objects no decision what so ever is made about which lexical object (or combination) will be used to represent a non-lexical object. Only possible lexical representations are identified.

(2) Abstraction from instance level.
This abstraction capability is well known and present in most conceptual models. It means that the building blocks are the object types (OT) instead of the individual objects.

(3) Abstraction from aggregation.
All associations between object types are expressed by means of binary relationships (facts). No fore-ordered data organization is imposed. No decision has to be taken whether an object is entity or attribute.

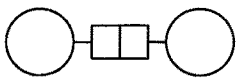
(4) Abstraction by generalization/specialization.
As in the OO-approach, the BR model supports the notion of object type sub-hierarchy and inheritance of properties along this hierarchy. An object type sub-hierarchy captures the "is-a" relationship between an object type and its subtypes. Multiple supertypes for a subtype are allowed.

In addition to these abstraction capabilities, the BR model explicitly addresses the issue of constraints. Constraints constitute the semantical component of the conceptual schema; they ensure that the schema meaningfully reflects the UoD. A large variety of constraint types are supported. Other constraints may be expressed in one of the constraint languages proposed in the literature [36, 32, 11].

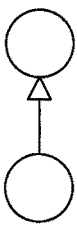
We adopt the well known "NIAM" graphical notation [12], [32] for the BR concepts :

a NOLOT -- (NOn-Lexical Object Type)

a LOT -- (Lexical Object Type). A LOT may be involved in one fact only, with a NOLOT.

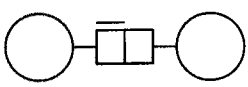


a Fact. The "boxes" are called Roles. Each Fact involves exactly two Object Types (which may be the same).

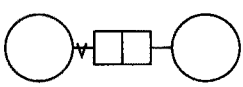


a Sublink - the subtype occurrences implicitly inherit all properties of the supertype. Subtypes need not be disjoint and not all of a NOLOT's occurrences need to be in one of its subtypes.

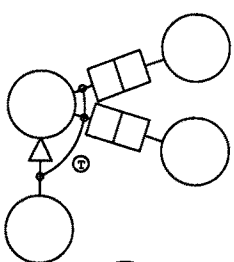
Certain constraint types occur so frequently and are so fundamental that they have a graphical representation as well. We only introduce the graphical representation of some constraint types:



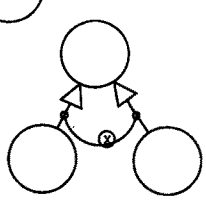
The Identifier constraint (simple functional dependency) is drawn as a line over the key-role.



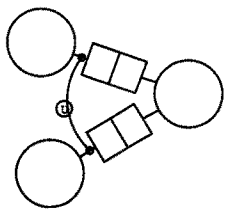
A Total Role constraint stating that each instance of an OT should participate in a given Role is represented by a "V" sign.



A Total Union constraint is a generalization of a Total Role constraint. Each instance of the OT should participate in at least one of the indicated Roles or Sublinks.



The Exclusion constraint expresses the mutually exclusion of a number of Subtypes.



A Uniqueness constraint is a generalization of the Identifier constraint. An instance of an Object Type is identified by an instances-combination of the indicated Object Types.

The classical BR model as such cannot be called object-oriented, because:
(1) Objects do not have an object identity. They only exist through their properties; objects with the same properties are considered to be the same.
(2) Behaviour of objects is not specified.
(3) Data cannot be encapsulated into an object.
(4) There is no way to define new abstract data types.
(5) There is no concept such as composite object.

3. An object oriented BR model (OO-NIAM)

In this section we introduce an object oriented version of the BR model. This is not done by simply adding extra features. Instead we build up the OO-BR model from scratch, but we follow most of the principles of the classical BR model and combine them with OO-principles.

3.1. Objects, Object Types and Subtypes

Following the OO-principles, an object is constituted by some private data (the state) and a set of operations (the behaviour) inextricably bound to it. In the classical BR model the concept of Object Type (OT) is only used to classify objects. In our OO version of the BR model, the description of state and behaviour of the objects are encapsulated into the OT. The OT defines the attribute relations, methods and constraints of the objects of this type. Attribute relations describe the state of an object, methods describe the behaviour of the object and the constraints restrict the possible states and the behaviour of an object. We use the term properties to denote attribute relations, methods and constraints.

In addition to the properties of the objects, an OT (which can also be considered as an object) may also described its own properties; they are called type-properties (type-attributes, type-methods and type-constraints).

As in regular NIAM, an OT will be represented by a circle.

An OT may be subtype of one or more supertypes. The properties of the supertypes are inherited by the subtypes. We keep the strict "is-a" meaning of the BR-model sublink concept; each object of the subtype is also an object of the supertype. We also keep the

graphical NIAM representation for a sublink; an arrow pointing from the subtype to the supertype. The BR-model exclusion and totality constraints expressible on sublinks keep their meaning.

As in Smalltalk [9, 17], we consider a single OT-sub-hierarchy for the entire schema. All OTs are ultimately subtype of the **pre-defined** OT "Object" which capture all objects of the UoD. According to the BR model principles, objects are classified into lexical and non-lexical objects. This means that the OT "Object" has two (disjoint) (pre-defined) subtypes "Lexical Object" and "Non-Lexical Object". The lexical objects may be further divided into e.g. "Boolean", "Number", "String" and "Text". See figure 1.

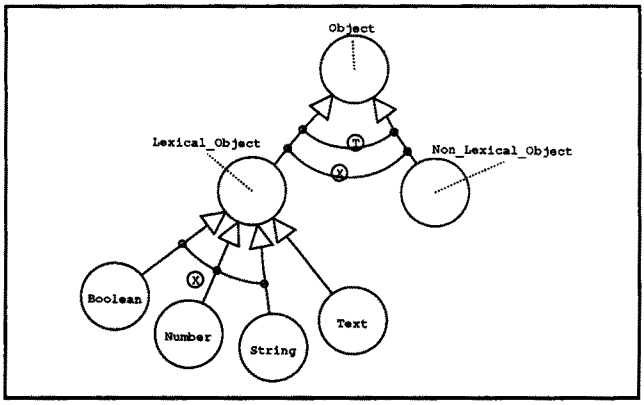


figure 1: pre-defined OT sub-hierarchy

Each **UoD-specific** OT is a subtype of either "Lexical Object" or "Non-Lexical Object". For example, document numbers are lexical objects while documents are non-lexical objects (see figure 2).

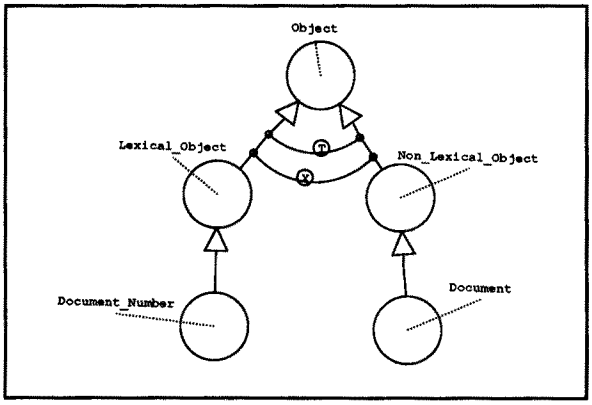


figure 2 : example UoD-specific OTs

We make the convention that UoD-defined OTs which are subtypes of "Lexical Object" will be represented by a dotted circle and we call them LOTs (similar as in regular NIAM). UoD-defined OTs which are subtype of "Non-Lexical Object" will be represented by a solid circle and are called NOLOTs (also similar as in regular NIAM) . By this convention the sublink arrow to the respectively supertype "Lexical Object" or "Non-Lexical Object" can be omitted (see figure 3).

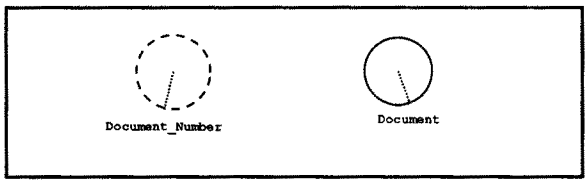


figure 3 : graphical convention for LOTs and NOLOTs

The (abstract) data type of a LOT may be specified by declaring the LOT a subtype of one of the pre-defined subtypes of "Lexical Object". For an example see figure 4; document numbers are defined to be numbers. Note that it is not necessary to specify the data type of LOT. It may be left unspecified till the implementation phase.

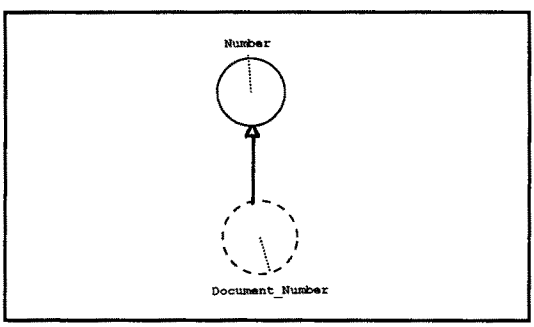


figure 4 : data type for a LOT

By introducing new subtypes of "Lexical Object" new abstract data types may be defined by the user himself. See for instance figure 5 where two new lexical types "Name" and "Date" are introduced.

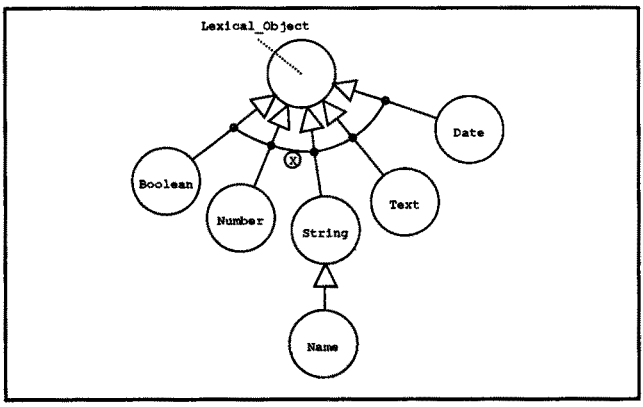


figure 5 : example of user defined abstract data types

Note that in figure 5 we have NOT used the dotted circle convention for "Name" and "Date" this to indicate that "Name" and "Date" are UoD *independent* OTs. This allows the user to extend the pre-defined subtype-hierarchy without limitations in a UoD independent way. These new defined subtypes can later be used in different conceptual schemas.

3.2. Object Type Definition

In the OO-version of the BRM, the OT concept is the mechanism to hide the description of the properties of the objects of this type. To describe these properties we will use a graphical representation technique very similar to the usual representation technique for the BR model.

The OT under description is represented as a circle embedded in a square. Attribute relations are considered to be special facts; they describe binary links between the objects of the OT under description and some other objects. The usual BRM constraints are used to restrict the possible instances of the attribute relations. For type-attributes the same graphical representation as for attribute relations is used but the role connected to the OT under description is labelled with a "T". Figure 6 represent the state part of the description of the OT "Document". The fact F1 is a type-attribute for the OT "Document". It specifies the "maximum size" of any document object. The graphical constraints specified for F1 state that there is only one maximum size and this value must be known even if there are no document objects.
The objects of "Size", "Title" and "Doc_Number" are encapsulated in Document objects; they are called internal OTs because no other object has access to a Document object's Size, Title or Doc-Number object.
In addition to internal attributes, an object may also have attributes (or type-attributes) which refer to external OTs. External OTs are defined independently of this OT definition. To distinguish an external OT from an internal OT, we put a dotted square around the

external OT. In the example of figure 6, the OT "Document" has an attribute relation with the external OT "Keyword".

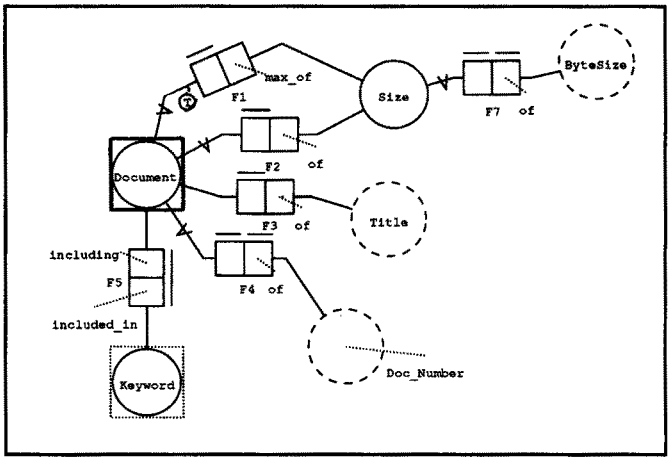


figure 6 : an OT definition

The difference between an object of an internal OT and an external OT is that an internal object can only be accessed by the objects with which it is involved in an attribute relation while an external object can be accessed (through its methods) by other objects as well. For an external object only the attribute relation is hidden, the objects themselves are not hidden. For instance in the example of figure 6, it is not possible for some object to access directly the Keyword objects contained in a given Document object because this link is hidden in the Document object. However, individual Keyword objects are accessible by other objects.

In addition to the graphical constraints, text constraints to restrict the states of an OT may be specified as well. Such a text constraint may access internal objects directly (or through local methods), but properties hidden in the definition of external OTs should be accessed through proper methods. An example of a simple text constraint for the OT "Document" would be

ByteSize of Size of a Document is <= ByteSize of Size max of Document

3.3 Methods

In addition to attribute relations and constraints the definition of an OT also includes a description of the methods of the objects of this type. The definition of a method may be given using some OO-language. As for the constraint language, methods may directly access the attribute relations and the internal objects of the OT under description, all external

objects should be accessed through the proper methods. A description of this language will be given elsewhere. An example of such a language can be found in e.g. [33, 11].

A method will be graphically represented by a rectangle. A fat arrow points to the OT of which it is a method. The OTs of the in- and return-objects are specified by connecting the method box and those OTs by in- resp. out-going arrows. The arrows may be labelled with a name to distinguish two different in- or return-objects of the same OT. To make the difference between an object-method and a type-method, we put a dot inside the OT for an object-method. Figure 7 is an example of the graphical representation of the object-method "Info" for the OT "Document" Figure 8 is an example of a type-method for the same OT.

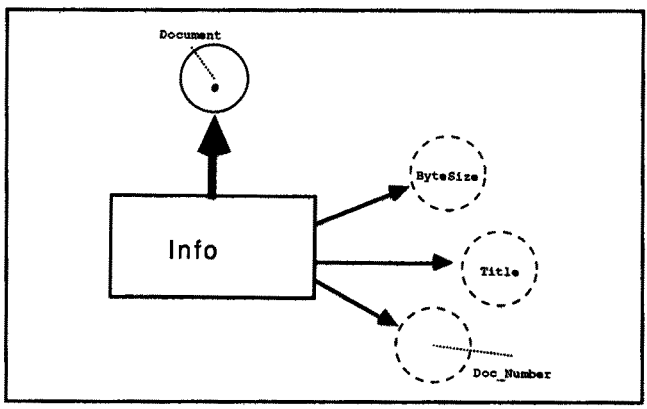


figure 7 : an object-method

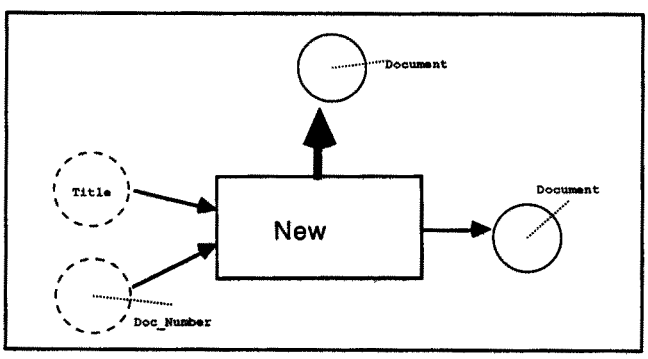


figure 8 : a type-method

3.4. Fact Types

In most OO-system, instance variables are the only means for relating objects. We have chosen to support also *explicit* relations. Explicit relations are not encapsulated in some object. We have found that during information analysis the explicit representation of

relationships between objects may be of great value. Very often there is no conceptual reason for considering one object attribute of the other object or vice versa. Introducing a new object to represent the relationship between the objects is not always a desirable conceptual solution. Representing the relationship in both objects is a kind of redundancy which we want to avoid as much as possible during conceptual modelling. In our opinion, explicit relations will also meet the so called "ravioli code" problem [30]. "Ravioli code" is the OO-version of "spaghetti code", it refers to lots of tiny well structured objects that are easy to understand in isolation, but whose interactions are nearly impossible to decipher.

Other OO-approaches which also support explicit relations are e.g.. [5] and [27].

As in regular NIAM, we use fact types to model explicit relations. Figure 9 shows an example of an explicit relation between the OTs "Document" and "Folder"; a Folder object may contain several Document objects and a Document object may be placed in different Folder objects.

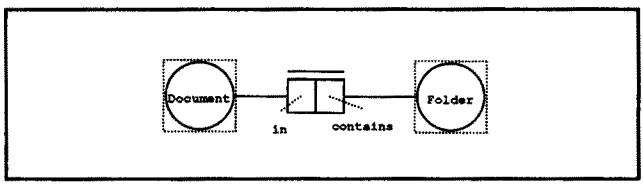


figure 9 : example of an explicit relation

3.5. Inheritance

A subtype inherits all properties of its supertypes. Contrary to most OO-systems, we only allow a limited form of overriding of the inherited properties. Along the subtype hierarchy, inherited attribute relations and constraints can only become more specific. For attribute relations inherited by a subtype this means that additional constraints may be specified which do not apply to the supertype. Also the OT of the attribute may become more specific; i.e. only a subtype of the original OT may be used.

In this way, at least for the structural part, we avoid problems with multiple inheritance. Multiple inheritance raises problems if an OT inherits the same property of two (or more) supertypes and the property is differently defined for each supertype. Since we only allow specialization, the subtype inherits the specializations given for each supertype. This will not cause any problems; in the worst case no object will ever satisfy the specialization. However, such a situation can be detected by a CASE (Computer Aided DEsign) tool. For an example of such a tool see [10, 12].

An example of inheritance of attribute relations is given in figure 10. In the graphical representation, the inherited attribute relation are shadowed. The additional constraints and the more specific OT are drawn in full lines.

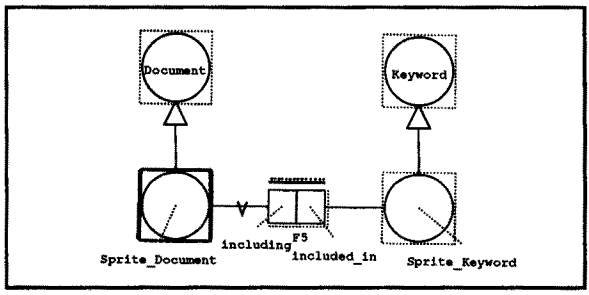


figure 10 : example of inheritance and specialization of attribute relations

Figure 10 shows an example; a Sprite_Document is defined as a subtype of a Document (see figure 6). They may only include Keywords which are registrated as Sprite_Keywords (OT-specialization of F5). Each Sprite_Document must refer to at least one Sprite_Keyword (extra constraint for F5).

Another example is given in figure 11. A Template_Document is a subtype of Document (see figure 6). A Template_Document includes a Style_Definition (extra attribute) but has no Keywords (extra constraint for F5). The latter is graphically represented by putting a vertical bar on the role connecting the OT "Keyword" to the OT "Template_Document".

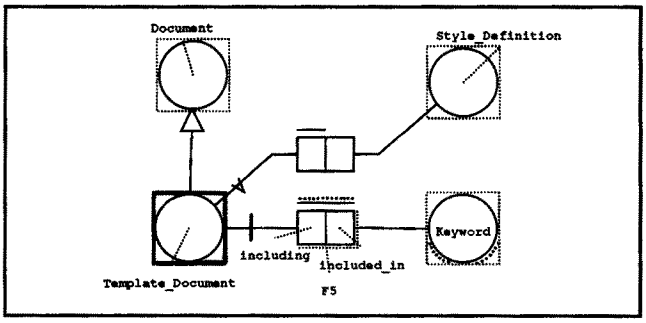


figure 11 : example of subtype definition

The implementation of the methods can be overwritten completely. So for methods we have to deal with the multiple inheritance problem. For a possible solution we refer to the literature [6, 7].

3.6. Type constructors

It is possible to see type constructors such as collection, bag, set, tree, list,... as objects (see figure 12).

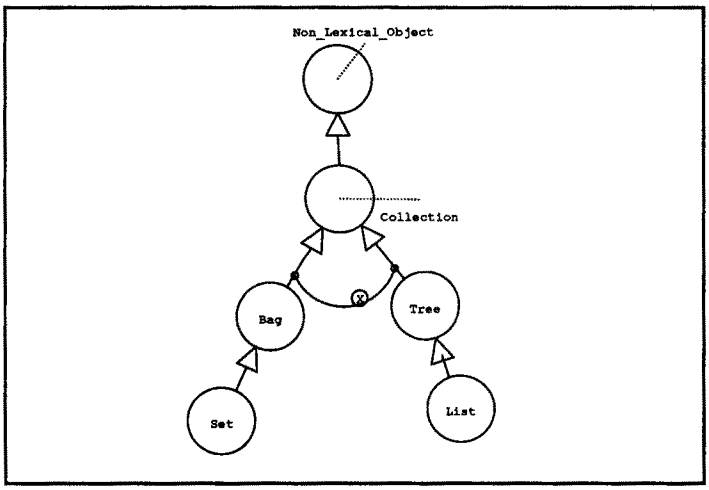


figure 12 : pre-defined type constructors

The definition of these OTs as pre-defined OTs make it possible for the user to define UoD-specific subtypes of these OTs which inherit all built-in methods such as e.g. first/next for "List". Usually, UoD-specific subtypes of the OTs of the "Collection" family-tree (such as e.g. "Keyword_List", "Document_Set") only restrict the elements of these collection to a certain UoD-specific OT. If this is the case, we make the convention that the OT is not represented as a subtype of "Collection" or one of its subtypes, but the name of the OT is composed of the name of the chosen type constructor and the name of the OT of the elements, e.g. "List_of_Keyword" is a List Collection of Keyword objects.

Of course the user is free to define new UoD independent subtypes of "Collection" or other type constructors (like e.g. bitmaps) in order to accommodate his needs.

3.7. Meta Object Types

Ots, fact types, constraints, methods, etc. may also be considered as objects of some (meta-)OTs. These meta-OTs together form the meta-schema which describe the OO-BR model.

We can consider the meta-schema as an integral part of the model, deriving a self-describing system. This has several advantages, for a detailled discussion see e.g. [23]. One of the advantages is that OTs may then be used as in- and return-objects for methods. In this way objects may be asked for their type(s) (OTs), OTs may be asked for their subtypes, methods, attribute relations and so on.

6. Conclusions

In the previous section we have presented an object oriented version of the well known Binary Relationship Model (also known as NIAM). The new model has been built from scratch following most of the principles of the conventional BRM in combination with OO-principles. The differences and similarities with the conventional BR model can be summarized as follows.

In contrast with the conventional BRM, the OO-BRM supports:
(1) encapsulation of properties,
(2) specialization of inherited properties,
(3) specification of (abstract) data types for lexical object types,
(4) definition of new abstract data types,
(5) type constructors as object types,
(6) definition of new type constructors,
(7) specification of the behaviour of objects as an integrated part.

The following BRM characteristics still apply for the OO-BRM:
(1) same graphical representation,
(2) same support for constraint specification (see also below),
(3) distinction between lexical and non-lexical object types,
(4) explicit relationship between object types,
(5) same subtype mechanism.

In principle, for supporting the specification of constraints the OO-BRM has the same capabilities as the conventional BRM. However because of the encapsulation principles of the OO-BRM, constraints which involve properties of multiple object types have to be expressed differently. A constraint defined in the context of one object type can only access the properties of an other object type through the methods of this object type. As a consequence, it will be less obvious which are all the properties involved in a certain constraint. On the other hand, the definition of OTs may be changed without affecting the constraints attached to other OTs. This is not always the case for the conventional BRM.

Related work can be divided into two groups. To the first group belong the newly defined object models e.g. [15, 33, 37, 38, 39]. These models are fully object oriented but have the disadvantage to be new and not be related with a well-tried and widely accepted model. Therefore it will be hard to introduce them in the sort term into big companies.

The second group contains work that extends the conventional models (mainly the E-R model) with OO-features such as complex objects, subtypes, generalizations and new data types e.g. [18, 35]. Although these extensions are valuable contributions to the models, they do not turn the model into a real object oriented model. We would rather call them object directed.

We argue that our model is truly object-oriented. Very important in achieving this goal was the introduction of a single subtype hierarchy, including type constructors and abstract data types, of which each user-defined object type is implicitly a subtype.

We believe that the proposed OO-BR model is a valuable contribution in order to reduce the gap between conventional conceptual models and the promising OO-DBMSs. In conventional environments the BR model has proven to be successful. By turning it into a truly OO model without actually affecting its main characteristics, it will also be possible to benefit from the advantages offered by the OO-approach when it is used in a non-conventional environment which demands for an OO solution.

Bibliography and References

[1] Andrews T., Harris C., "Combing Language and Database Advances in an Object-Oriented Development Environment". In Proceedings 2nd ACM OOPSLA Conf. Oct 1987.

[2] Atwood T.M., "An Object-Oriented DBMS for Design Support Applications". In Proceedings IEEE COMPINT 85, Montreal Canada, pp. 299 - 307.

[3] Bancilhon, et al., "The Design an Implementation of O2, an Object Oriented Database System". In Advances in Object Oriented Database Systems, proceedings 2nd Intl workshop on OO Database Systems. Ed. Dittrich K., Sept 1988. Lecture Notes in Computer Science 334, Springer Verlag.

[4] Banerjee J., Chou H.T., Garza J.F. , Kim W., Woelk D., Ballou N., Kim H.J., "Data Model Issues for Object-Oriented Applications". In ACM Trans. On Office Information Systems, Vol 5, N 1, 1987.

[5] Bratsberg S.E., "FOOD: Supporting Explicit Relations in a Fully Object Oriented Database". In proc. Object Oriented Databases (DS-4) North-Holland, 1990.

[6] Cardelli L., "A Semantics of multiple inheritance". In Intl Symposium on Semantics of Data Types, Sophie-Antipolis, France 1984. Lecture Notes in Computer Science 173, Springer Verlag.

[7] Cardelli L., Wegner P.,"On understanding types, data abstraction and polymorphism". In ACM Computing Surveys 17:4, 1985.

[8] Chen P.P., "The Entity-Relationship Model - Towards a Unified View of Data". In ACM trans. on Database Systems 1(1) pp.9-36 (1976).

[9] Copeland G., Maier D., "Making Smalltalk a Database System". In Proc. ACM SIGMOD Intl. Conf. on Management of Data, Boston, Mass., June 1984.

[10] De Troyer O., "RIDL*: A Tool for the Computer-Assisted Engineering of Large Databases in the Presence of Integrity Constraints". In Proceedings of the ACM-SIGMOD "International Conference on Management of Data", Oregon 1989.

[11] De Troyer O., Meersman R., Ponsaert F., "RIDL User Guide", Control Data DMRL Research Memorandum (1983) [available from the authors].

[12] De Troyer O., Meersman R., Verlinden P., "RIDL* on the CRIS Case: A Workbench for NIAM". In "Computerized Assistance During the Information System Life Cycle". Proceedings IFIP CRIS-88 Conference, eds. Olle T.W., Verrijn-Stuart A.A., Bhabuta L., North-Holland (1988).

[13] Dijkstra J, De Troyer O., Meersman R., Weigand H, "RIDL* as a software engineering aid - some practical results". In Proc. 4th Filin Conf of methods and tools as aids to design information systems. Ed. Habrias, Nantes Sept 1990.

[14] Falkenberg E., "Concepts for Modelling Information". In "Modelling in Data Base Management Systems", Proceedings of IFIP TC-2 Conference, North Holland, 1976.

[15] Fishman D., et al., "IRIS: an Object-Oriented Database Management System". In ACM Trans. on Office Information Systems, Vol. 5, N. 1, 1987.

[16] Gadre S., "The Enterprise and Information Model". In "Database Programming and Design", Volume 1 (1), 1987.

[17] Goldberg A., Robson D "Smalltalk-80: The Language and its implementation". Addison-Wesley, Reading, Mass. 1983.

[18] Hohenstein U., Gogolla M., "A Calculus for an Extended Entity-Relationship Model Incorporating Arbitrary Data Operations and Aggregate Functions". In "Entity-Relationship Approach", ed. C. Batini, Elsevier Science Publ. (North Holland), 1989.

[19] International Standards Organisation, "Concepts and Terminology for the Conceptual Schema and the Information Base". ISO TR#9007 (also as: N695; Ed. J.J. van Griethuysen) (1982).

[20] Maier D., "Why isn't there an object oriented Data Model", In Information Processing 1989. Ed. G.X. Ritter, Elsevier Science Publisher IFIP 1989.

[21] Maier D, Stein J., Otis A., Purdy A., "Development of an Object Oriented DBMS", Proc. of the ACM OOPSLA Conf., Portland Oregon Sept 1986.

[22] Mark L., "What is the Binary Relationship Approach?". In "Entity-Relationship Approach to Software Engineering", Ed. Davis, North Holland 1983.

[23] Mark L., Roussopoulos N. "Integration of Data, Schema and Meta-schema in the Context of Self-Documenting Data Models". In "Entity-Relationship Approach to Software Engineering", Ed. Davis, North Holland 1983.

[24] Meersman R., "Towards Formal Models for Reasoning about Conceptual Database Design". In "Data and Knowledge", Proceedings of the IFIP Working Conference DS-2, Eds: R. Meersman, A. Sernadas, North Holland (1988).

[25] Nijssen G.M., "A Gross Architecture for the Next Generation Database Management Systems". In Modelling in Database Management Systems; proceedings of the IFIP TC-2 Conference, Ed. G.M. Nijssen. North Holland (1976).

[26] Nijssen G.M., Halpin T.A., "Conceptual Schema and Relational Database Design", Prentice Hall 1989.

[27] Rumbaugh J., "Relations as Semantic Constructs in an Object Oriented Language". In Proc. of OOPSLA 1987.

[28] Shipman D.W., "The Functional Data Model and the Data Language DAPLEX". In ACM trans. on Database Systems 6(1), pp. 140-173, March 1981.

[29] Stonebraker M., Rowe L., "The Design of POSTGRES". In Proc ACM SIGMOD Intl. Conf. on Management of Data, Washingthon D.C., May 1986.

[30] Taylor David, "Object Oriented Technology: A Manager's Guide". Servio Corporation 1990.

[31] Vanparys R., "Modulaire decompositie van informatiestructuren". In Informatie Vol. 29, No. 6 pp.493-568, 1987.

[32] Verheijen G., van Bekkum J., "NIAM: An Information Analysis Method". In Proceedings of IFIP TC-8 Conference on Comparative Review of Information Systems Methodologies (CRIS-1), Eds. Verrijn-Stuart A., Olle T.W., Sol H., North Holland (1982).

[33] Weigand H., "An Object Oriented Approach in a Multi Media Database Project". In Proc. Object Oriented Databases (DS-4), Eds. Kent W., Meersman R., North Holland 1990.

[34] Wintraecken J.J. "NIAM in Theorie en Praktijk", Academic Service, 1986 (Book in Dutch).

[35] Lipeck U., Neuman K. "Modelling and manipulating objects in geo scientific databases". In E-R approach, ed. Spaccapietra S. Elsevier (North Holland) 1987.

[36] Nienhuys-Cheng "Classification and Syntax of Constraints in Binary Semantical Networks". In Information Systems Vol 15, No 5, pp497-513, 1990.

[37] Su S.Y.W. "An Object-Oriented Semantic Association Model (OSAM*)". In AI in Industrial Engineering and Manifacturing: Theoretical Issues and Applications", eds. Kumara, Kashuap, Soyster. American Institute of Industrial Engineers 1988.

[38] Costa J.-F., Sernadas A., Sernadas C. "Oblog: User's Manual", Technical Report INESC 1989.

[39] Sernadas C., Fiadeiro J., Sernadas A. "Object-Oriented Conceptual Modelling From Law". In The Role of AI in Databases and Information Systems, eds. C.-H. Kung and R. Meersman, North-Holland 1990.