DEPARTEMENT TOEGEPASTE ECONOMISCHE WETENSCHAPPEN

ONDERZOEKSRAPPORT NR 9630

Formal Specification Techniques in Object-Oriented Analysis: A Comparative View

by

Monique Snoeck

Jozef Wijsen

Guido Dedene

Katholieke Universiteit Leuven

Naamsestraat 69, B-3000 Leuven

ONDERZOEKSRAPPORT NR 9630

Formal Specification Techniques in Object-Oriented Analysis: A Comparative View

by

Monique Snoeck

Jozef Wijsen

Guido Dedene

D/1996/2376/30

Formal Specification Techniques in Object-Oriented Analysis: A Comparative View

Presented at the Workshop on Evaluation of Modeling Methods in Sytems Analysis and Design
*CAiSE*96, Crete, 20-21 May 1996*

Monique Snoeck*, Jozef Wijsen°, Guido Dedene*

*Katholieke Universiteit Leuven
Department of Applied Economic Sciences
Naamsestraat 69
B-3000 Leuven, BELGIUM
phone: (+) 32 16 32 66 12
fax: (+)32 16 32 67 32
e-mail: Monique.Snoeck@econ.kuleuven.ac.be
Guido.Dedene@econ.kuleuven.ac.be

°Vrije Universiteit Brussel
Department of Computer Science,
Pleinlaan 2
B-1050 Brussels, BELGIUM
phone: (+)32 2 629 34 87
fax: (+)32 2 629 34 95
e-mail: jwijsen@vub.ac.be

Abstract. During the last decade, object orientation has been advanced as a promising paradigm for software construction. In addition several authors have advocated the use of formal specification techniques during software development. Formal methods enable reasoning (in a mathematical sense) about properties of programs and systems. It is clear that also object oriented software development can benefit from the use of formal techniques.
But although the object oriented analysis (OOA) methods claim to provide the necessary concepts and tools to improve the quality of software development, they are in general informal. This is surprising as the modeling techniques used in OOA have a high potential for formalization. The purpose of this study is to compare the specification techniques used in current OOA-methods. In particular, the degree of formality provided by most of the methods is discussed and evaluated from a quality control perspective.

Introduction

In comparison with classical development methods, object oriented analysis (OOA) methods have the advantage that they allow for a seamless transition from analysis to design and implementation. Although seamless transition is a major advancement in software engineering practice, the quality of specifications in terms of correctness and consistency remains crucial from a quality control perspective. Several authors have advocated the use of formal specification techniques during software development [9]. Formal methods enable reasoning (in a mathematical sense) about properties of programs and systems [8]. Eventually, one may *prove* that certain anomalous system behaviour, such as deadlock, cannot occur [10, 19]. It is clear that also object oriented software development can benefit from the use of formal techniques.

Formal techniques are not necessarily mathematical specification languages but can be graphical techniques as well, provided that the syntax and semantics of these techniques are precisely described. This paper concentrates on OOA-methods which primarily use graphical specification techniques. The purpose of this study is to look to what extent these graphical specification techniques are formalised.

The next section lists the criteria used to compare the methods under consideration and motivates each one by illustrating some problems related to the use of informal specification techniques. Section 3 lists the object oriented analysis methods that are considered in this study and briefly describes how they apply well-known techniques. Section 4 addresses the actual evaluation of the methods. Finally, section 5 presents a conclusion and discussion.

Criteria for Comparison and Motivating Examples

An *objective* appraisal of methods for their level of formality is not an obvious task. The first step in eliminating subjectiveness is the definition of evaluation criteria. As explained in the introduction, the methods will be evaluated with a quality control perspective in mind. Quality of specifications is defined as internal consistency and correctness.

Internal consistency. In object oriented modeling, static, dynamic and interaction aspects are described with equal emphasis. The methods under consideration in this paper, all use different techniques for modeling each aspect. Even if these techniques model different

aspects of objects, they model the same Universe of Discourse and might have overlapping semantics. As a consequence, specifications must be checked for internal consistency.

Correctness. If behaviour of object types is modelled by means of Finite State Machines, executable systems are a set of Concurrent Finite State Machines. Concurrent State Machines are a well known specification technique in the domain of protocol validation, where they are used to check protocols for fairness and deadlock-freedom in a strictly formal way. As Finite State Machines are used in a different way in OOA, these algorithms for correctness checking cannot be transposed to OOA in a straightforward manner. This, however, does not mean that it is impossible to define algorithms that check object oriented specifications for correctness, as demonstrated in [10, 19].

A prerequisite to formal consistency and correctness checking is that syntax and semantics of the concepts employed by a particular method are rigourously and unambiguously defined, which is evaluated by the following two criteria:

Criterion 1: Syntax. Is the syntax of the method defined in a rigourous manner, or is it merely loosely described ?

Criterion 2: Semantics. Is each concept of the method provided with a formal semantics, or is the meaning of concepts only paraphrased in natural language ?

Many OOA-methods are superficial about the syntax and semantics of the concepts they use. This may compromise the quality of the specifications made by these methods. We give two examples, one from data modeling and one from process modeling, to illustrate the importance of precise syntax and semantics definitions.

Example. Conceptual schemas often look very natural and intuitively clear. Yet intuition can be misleading. It may suggest certain aspects which, in fact, are not modelled. The ER-schema of figure 1 [15] describes a mail course company. Each course consists of several parts, and students have to complete a homework per course part. Intuitively, the schema looks all-right. Nevertheless, a closer inspection reveals that the schema fails to model an

obvious constraint: a student must not receive parts of courses for which no subscription was made. One may believe that this constraint is implied by the schema, yet, in fact, it is not.

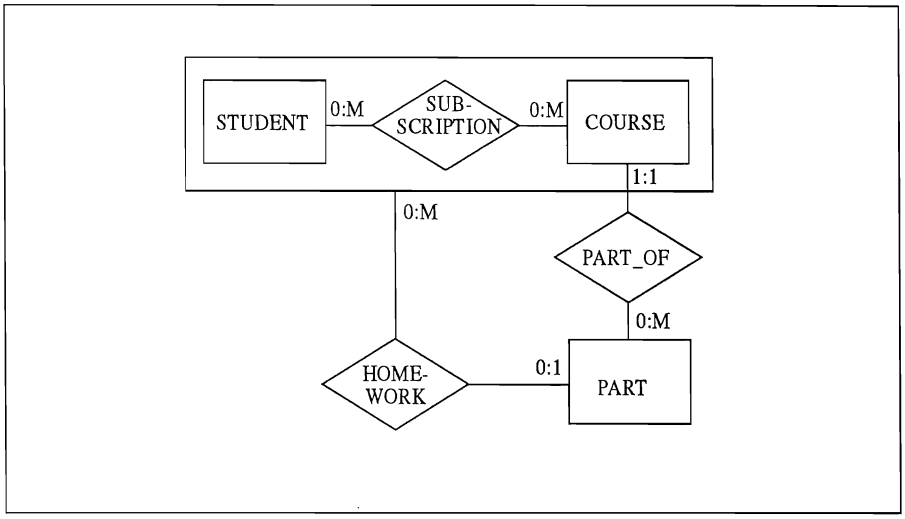

Fig. 1. Intuition may be misleading.

A rigourous definition of semantics allows to precisely determine what constraints are modelled by a particular schema.

Example. Most OOA-methods use Finite State Machines (FSMs) or Harel Statecharts (an extension of FSMs) to model object behaviour. Regular Expressions, Regular Languages and Finite State Machines have been extensively used and formalised for the development of programming languages and their parsers [1]. However, their use in the context of object oriented analysis requires that the semantics of Finite State Machines be refined to model the concept of a lifecycle more accurately. For example, the FSM shown in figure 2(a) follows the syntax of a FSM, but is unacceptable from our viewpoint. Indeed, as there is no path from the initial to the final state, the FSM is not meaningful in the context of modeling object lifecycles. Reversing all transitions results in a new FSM with a path from the initial to the final state (figure 2(b)). If other diagramming techniques of the method allow to define the events create_P, destroy_P and modify_P as creator, modifier or destructor of class occurrences respectively, the FSM does not fit our intuition about a meaningful lifecycle (the destroy event precedes the create event). This example shows that the basic semantics of FSM deserve further extensions.

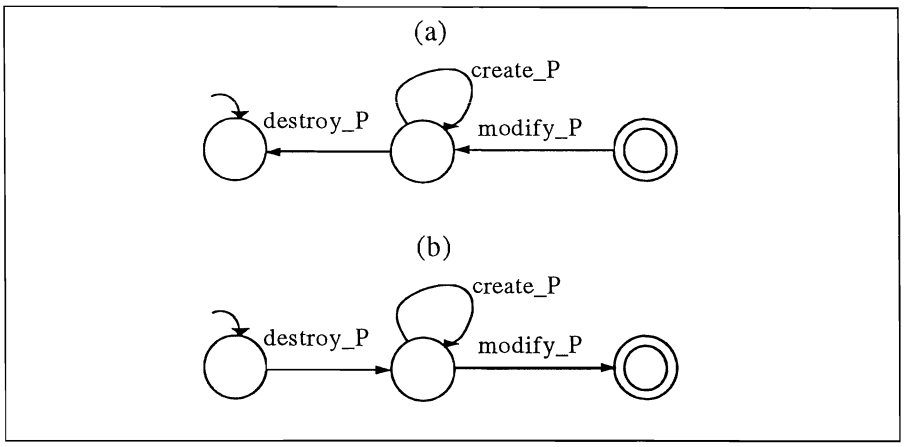

Fig. 2. A Finite State Machine for an object type *P*.

Criterion 3: Consistency between schemas. Static, dynamic and interaction aspects of objects are generally modelled by different techniques. Nevertheless, the resulting schemas are likely to be interrelated. The relation between static, dynamic and interaction schemas should be made explicit and checked for consistency. The third criterion looks whether a particular method defines a formal procedure to check the consistency between subschemas.

Example. The possibility to define generalisation/specialisation hierarchies is seen as a key element in the OO-paradigm. However, the question of how behaviours of generalisation and specialisation relate to each other deserves special attention. Examples of relevant questions are [20, 19]:

- Does a specialisation inherit the statemachine of its parent ?
- Can it refine this statemachine by adding, removing or redefining states, transitions or events ?
- Can it restrict the behaviour of its parent or extend it or both ?

Many methods do not answer these questions in a precise fashion. For example, in OOSA [18] the life-cycle of a subtype corresponds to a part of the life-cycle of its supertype. This definition violates the broadly accepted notion of inheritance where subtypes inherit data *and behaviour* of their supertype. [20] gives an in-depth study of this particular problem.

Criterion 4: Overall system behaviour. Once consistency between schemas is established, it must be possible to derive a global system behaviour from the individual schemas. More

particularly, it must be possible to compose individual object behaviour and interaction descriptions into a single system behaviour specification.

Criterion 5: Anomalous system behaviour. If the overall system behaviour is specified, can we check it for desirable properties such as deadlock-freedom and fairness ?

Example. Most conceptual schemas define more than one object type and allow different object types to synchronize somehow on some event types. This means that different object types might impose conflicting sequence restrictions on event types. In such a case a deadlock occurs. This can be illustrated by an example taken from Belgian legislation:

> *When a house is acquired, a deed of sale has to be signed. Property law prescribes full payment of the transaction's amount on the spot. In order to obtain the necessary funding, most buyers need to contract a loan. In general, banks are only willing to contract a loan provided a first mortgage can be held on a property. In its turn a mortgage can only be held on a property that is already owned by the mortgagor.*

Hence, anyone who acquires his first property and needs a loan to fund the acquisition is faced with a problem of circular prerequisites: the deed can not be signed before funding is available, but the funding can not be made available before the deed has been signed. This kind of circular prerequisite is not easy to detect if no formal consistency checking procedure for the overall implications of the constraints is available.

The Methods Evaluated

Figure 3 shows the list of object-oriented analysis methods that were taken into consideration in this study together with the consulted references.

Figure 4 shows the methods under consideration together with the techniques they use for representing static, dynamic and object interaction aspects.

AUTHORS	METHOD	REFERENCES
Embley et al.	OSA	[11]
Kappel, Schrefl	Object/Behaviour (O/B)	[13]
Hayes, Coleman et al.	FUSION	[12, 6, 7]
Rumbaugh et al.	OMT	[16]
Shlaer, Mellor	OOSA	[17, 18, 14]
Coad, Yourdon	OOA	[5]
Booch	OOD	[2, 3]

Fig. 3. Object-oriented analysis methods considered in this study.

Data modelling. For data modelling purposes, most methods use the basic concepts of the EER-model [4] but add a number of proprietary concepts. These proprietary concepts are illustrated by one or more examples but are usually not defined in a formal way.

Process modelling. All methods (except for O/B) use the category of Regular Languages, which can be represented either by FSMs, Harel Statecharts or Regular Expressions, for the purpose of process modeling. Again, "using" any of these formalisms is mostly limited to using the same notations but ignoring the syntax and semantics of the original technique. We illustrate this for OOA and for OMT. Coad and Yourdon explain the use of the FSM technique in OOA in their handbook by means of only *one* example [5, p. 146]. In this example, the FSM shows only states and legal transitions (transitions are not labelled) and it has no final state. In OMT, State diagrams can consist of a set of concurrent FSMs. Apparently, FSMs do not always have an initial state and/or a final state ([16], figure 5.1, p.107). The vagueness with which the technique of FSMs is used in OMT seems to be a consequence of the fact that the authors of this method have not given a precise definition of FSMs. As a result, guide-lines for checking the dynamic model for consistency and completeness such as [16, p. 179]:

> *"Check for completeness and consistency at the system level when the state diagrams for each class are complete. ... States without predecessor or successor are suspicious ...",*

are too general and unprecise. Formal techniques should provide a designer with precise criteria about completeness, consistency and the correctness of each construct.

METHOD	STATIC MODEL	DYNAMIC MODEL	INTERACTION MODEL
OSA	EER with extensions	FSM	Interaction Diagram
O/B	Own formalism	Petri-Nets	Complex Activity
FUSION	EER with extensions	Harel-Statecharts Regular expressions	Object Interaction Graph
OMT	EER with extensions	Harel-Statecharts	Event Trace Diagram
OOSA	EER with extensions	FSM	Object Communication Model Object Access Model
OOA	Generalization/Specialization Whole/Part	FSM	Message Connections
OOD	Variant of EER	Harel-Statecharts	Timing Diagrams

Fig. 4. Techniques used to model static, dynamic and interaction aspects.

Interaction modelling. A variety of techniques, mostly based on the concept of message passing are proposed for interaction modelling. Except for O/B and Fusion, the semantics of object interaction is never precisely defined. As a result, correctness checking for interaction schemes is reduced to recommendations such as [16, p.179]:

"...; beware of synchronization errors where input occurs at an awkward time. Make sure that corresponding events on different state diagrams are consistent. ...".

The question is what is precisely meant by "an awkward time" and "consistent".

Finale

We have used the above criteria for comparing the object-oriented analysis methods under consideration. Figure 5 shows the result. Possible scores are *high* (■), *medium* (▮), *low* (|), and *absent* (.). The given scores are motivated as follows.

Syntax and semantics. For the first two criteria, a *high* score is attributed to methods with a proprietary definition of syntax and semantics. A method without formal definition of syntax and semantics gets a *medium* score if it uses a standard technique, possibly with minor extensions, for which a formal definition exists (e.g. Entity-Relationship and Finite State Machines). In case of major extensions or techniques that are a collection of concepts of diverse origin, a *low* score is assigned

Most methods list the basic (graphical) symbols that can appear in a schema. How these symbols are actually combined into a schema, is generally illustrated by examples. Precise definitions appear only in OSA, O/B and Fusion. OSA defines OSA-schemas by means of a meta model, which itself is stated in terms of ORM-diagrams (Object-Relationship-Model diagrams, the OSA-equivalent of ER-diagrams). The meta model describes the syntax of ORM-diagrams, state-nets and interaction diagrams. Fusion gives a detailed, though informal, description of the graphical symbols in the Object Model and the ways of combining these symbols. The syntax of the interaction model and data dictionary are given in BNF-notation. A syntax definition for the petri-nets in O/B can be found in [13].

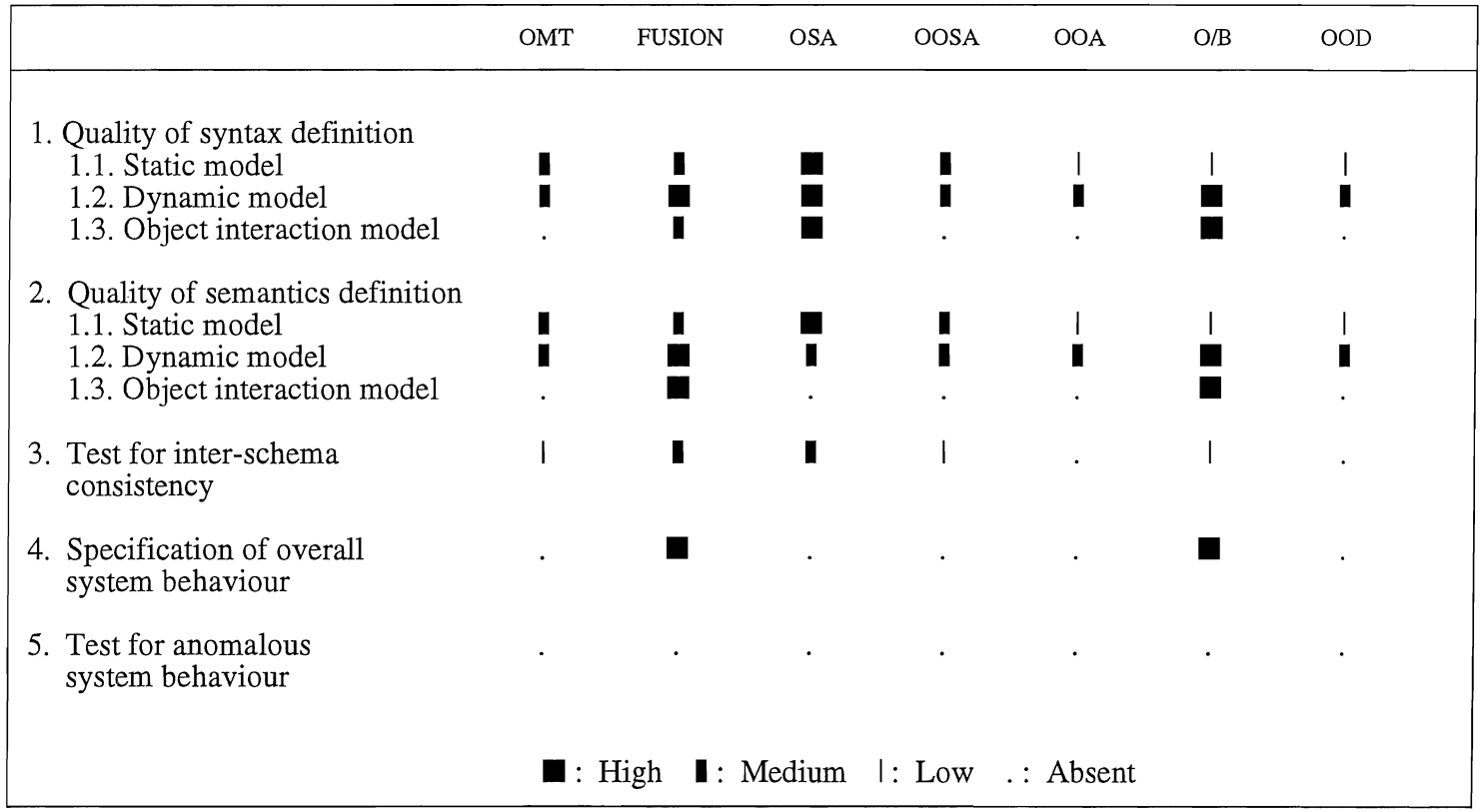

Fig. 5. Formal aspects of object-oriented analysis methods.

The meaning (semantics) of concepts are mostly explained by means of one or more examples. The meaning of ORM-diagrams in OSA is defined by a mapping from ORM-diagrams to first-order predicate calculus. The meaning of state-nets and interaction diagrams, on the other hand, is not defined. A recent book on Fusion [7] informally describes the semantics of the concepts in use. Nevertheless, in earlier work [6], the behaviour of single objects is described by deriving sets of traces from an object definition. Interestingly, this approach allows the definition of a global system behaviour.

Consistency between schemas. Not one of the methods under consideration provides a formal treatment of consistency between schemas. In OOA and OOD the question of consistency checking is not even mentioned, which explains an *absent* score. OMT, OOSA and O/B deal with this topic in a very vague and informal manner, for which they deserve a *low* score.

To the authors' knowledge, OSA has not concerned inter-schema consistency. Nevertheless, checking different schemas for inconsistencies seems theoretically possible. The authors of Fusion admit that their approach to consistency checking is intractable:

> "*In practice, proof [of consistency between models] is totally impractical. Thus we do not expect the analyst to prove consistency between models in the general case. Informal reasoning about judiciously selected examples has to be sufficient.*" [12, p. 181].

This explains the *medium* score for both methods. No method has a formal consistency checking procedure.

Overall system behaviour. As is to be expected, methods without formal syntax and semantics do not define overall system behaviour. This aspect is only covered by O/B and Fusion. Unfortunately, Fusion does not define the dynamic creation and deletion of objects at run time.

Anomalous system behaviour. None of the reviewed methods investigates properties of the overall system behaviour.

Concluding Remarks

OOA-methods can benefit in several ways from the availability of formal mathematical semantics (e.g. consistency checking). We found that most current object oriented specification techniques are informal in one way or another. Importantly, not one of the methods reviewed incorporates the concept of overall system behaviour.

From the point of view of the quality of the software development process, the formalisation of OOA-methods will allow for correctness checking at an earlier stage in the software development process, hereby reducing development costs. The precise definition of the syntax of a method is a prerequisite for the development of a supporting CASE-tool. The precise definition of semantics and the availability of a formal procedure to check consistency between schemas allow to add intelligence to such a CASE-tool. Without these features, CASE-tools can't offer much more support than diagram-editors.

References

[1] Aho, A.V., and Ullman, J.D. *The theory of Parsing, Translation and Compiling. Volume I: Parsing*. Prentice Hall, Englewood Cliffs, N.J., 1972.

[2] Booch, G. Object oriented development. *IEEE Transactions on Software Engineering 12*, 2 (Feb. 1986), 211-212.

[3] Booch, G. *Object Oriented Analysis and Design with Applications*. Second Edition, Benjamin/Cummings, Redwood City, CA, 1994.

[4] Chen P.P., *The Entity Relationship Approach to logical Database Design*, QED information sciences Wellesley (Mass.),1977

[5] Coad, P., and Yourdon, E. *Object-Oriented analysis*. Prentice Hall, Englewood Cliffs, N.J., 1991.

[6] Coleman, D., Hayes, F., and Bear, S. Introducing Objectcharts or How to Use Statecharts in Object Oriented Design. *IEEE Transactions on Software Engineering 18*, 1 (Jan. 1992), 9-18.

[7] Coleman, D., Arnold, P., Bodoff, S., Dollin, C., Gilchrist, H., Hayes, F. Jeremaes, P., *Object-Oriented Development, The FUSION Method*. Prentice Hall, Englewood Cliffs, N.J., 1994.

[8] Cooke, J. Formal methods—Mathematics, theory, recipes or what ? *The Computer Journal 35*, 5 (May 1992), 419-423.

[9] de Champeaux D., America P., Coleman D., Duke R., Lea D., Leavens G., Formal Techniques for OO Software Development (PANEL), *OOPSLA'91 conference proceedings*, Addison-Wesley Publishing Company, pp.166-170

[10] Dedene G., Snoeck M., Formal deadlock elimination in an object oriented conceptual schema, *Data and Knowledge Engineering*, Vol. 15 (1995) 1-30

[11] Embley, D.W., Kurtz, B.D., and Woodfield, S.N. *Object-Oriented Systems Analysis: A Model-Driven Approach.* Yourdon Press, Prentice Hall, Englewood Cliffs, N.J., 1992.

[12] Hayes, F., and Coleman, D. Coherent Models for Object Oriented Analysis. In *Proceedings of OOPSLA'91 Conference*, 8-10 Octobre, 1991, ACM Press (N.Y.), 1991, 171-183.

[13] Kappel, G., and Schrefl, M. Using an object-oriented diagram technique for the design of information systems. In Sol, H.G., and van Hee, K.M., Eds. *Dynamic Modeling of Information Systems*, Elsevier Science Publishers B.V., North-Holland, 1991, 121-164.

[14] Lang, N. Shlaer-Mellor Object-Oriented Analysis Rules. *ACM SIGSOFT Software Engineering Notes 18*, 1 (Jan. 1993), 54-58.

[15] Put, F *Introducing Dynamic and Temporal aspects in a Conceptual (Database) Schema.* Ph.D. Dissertation, K.U.Leuven, Department of Applied Economic Sciences, 1988.

[16] Rumbaugh, J., Blaha, M., Premerlani, W., Eddy, F., and Lorensen, W. *Object Oriented Modeling and Design.* Prentice Hall, Englewood Cliffs, N.J., 1991.

[17] Shlaer, S., and Mellor, S.J. *Object-Oriented Systems Analysis: Modeling the World in Data.* Yourdon Press, Englewood Cliffs, N.J., 1988.

[18] Shlaer, S., and Mellor, S.J. *Object Lifecycles: Modeling the World in States.* Yourdon Press, Englewood Cliffs, N.J., 1992.

[19] Snoeck M., A process Algebra Approach for the construction and analysis of M.E.R.O.DE.-based conceptual models, PhD. Dissertation, K.U.Leuven, Faculty of Science and Department of Computer Science, 1995

[20] Snoeck M., Dedene G., Generalization/Specialization and Role in Object Oriented Conceptual Modeling, accepted for publication in *Data and Knowledge Engineering*.