On the Measurement of Event-Based Object-Oriented Conceptual Models

Geert Poels

Management Information Systems Group
Department of Applied Economic Sciences
Katholieke Universiteit Leuven
Naamsestraat 69, B-3000 Leuven, Belgium
geert.poels@econ.kuleuven.ac.be

Abstract

A suite of measures is presented that addresses two problem areas within contemporary object-oriented software measurement, i.e. the lack of measures for the early stages of system development, like conceptual modeling, and the lack of measures for dynamic models of an object-oriented system. Our suite of measures is based on a formally defined object-interaction model, called the object-event table. Generally, the objects in a domain are affected by the occurrence of real-world events. A framework for measurement is presented that expresses and measures aspects of the data, function and dynamic behaviour dimensions of a domain, in terms of (common) event participations.

1. Introduction

After so many years of research into object-oriented software measurement, a huge amount of measures for object-oriented software models and artifacts has been produced. Zuse (1998) provides a list of no less than 137 of such measures and estimates the actual number of measures at more than threehunderd. In spite of these research efforts there are still areas that remain largely untouched by object-oriented software measurement research. In this paper we present an approach that addresses two of these research 'niches'.

A first problem area, identified during the previous WQAOOSE (Brito e Abreu *et al*. 1999), concerns the lack of measures for models that capture the dynamic aspects of an object-oriented software system. A typical measure suite for object-oriented software, like for instance MOOSE (Chidamber and Kemerer 1994), focuses on the data and function dimensions of software, but ignores the dynamic behaviour dimension as captured by behavioural and object-interaction models like state-transition diagrams and activity diagrams.

A second problem area concerns the lack of measures for object-oriented software specifications (Briand *et al*. 1999). Although industry begs for measurement instruments that can be applied in the early phases of the development process (mainly for early quality control and project budgeting decisions), nearly all published object-oriented software measures can only be used after (high-level) system design. Some exceptions known to us are measures for object-oriented analysis models (e.g. task points (Graham 1995) for Graham's SOMA, the QOOD measure suite (Badri *et al*. 1995) for Coad and Yourdon's OOA, and the complexity measures presented in (Genero *et al*. 1999) for Rumbaugh's OMT).

We believe these two problem areas to be somewhat related. Modern, UML-compliant approaches towards domain analysis, object-oriented analysis and design, and component-based software engineering, like for instance Catalysis (D'Souza and Wills 1999), put emphasis on both the static and dynamic aspects of a domain or software system. However, in an object-oriented implementation the dynamic aspects become somewhat subordinate to the static aspects. Many of the 'rules' that were explicitly captured during behavioural and object-interaction modeling, are translated into class invariants or into preconditions and postconditions that are attached to

specific methods within the class definitions. Strangly enough, these types of assertions haven't received the attention of research into object-oriented design and code measurement either.

In this paper, we present part of a framework for measuring object-oriented conceptual models. Conceptual modeling is used to model, structure and analyse a (part of a) domain,[1] irrespective of the software system that must be built. Defining a domain model is part of the requirements engineering step in the development of a software system. All the rules described in the domain model have to be supported by the system. Object-oriented analysis in general also aims at modeling and analysing the specific system requirements (user interface, data storage, workflow aspects, quality requirements, etc.). The conceptual model can be considered as an early object-oriented analysis artifact. As a consequence, our framework is suited for early measurement.

Conceptual models are combinations of different sub-models. The part of the framework presented here is based on one of these sub-models: the object-interaction model. To model the interaction (and communication) between objects the framework assumes the event broadcasting mechanism, which from a conceptual modeling point of view is to be preferred above the message passing mechanism.[2] The cornerstone of the measurement framework is a formally defined object-interaction model based on event broadcasting, called the object-event table (OET). The OET provides a formal basis for a suite of measures that is defined in terms of (common) event participations.

In section 2 the object-event table is presented. A compact suite of measures, including size, coupling, inheritance, specialisation, propagation and polymorphism measures is presented in section 3. Finally, section 4 contains conclusions and topics for further research.

2. The object-event table

In conceptual modeling, the events that are modeled are real-world events, sometimes also referred to as business events. Real-world events are characterised by the following properties:

- A real-world event corresponds to something that happens in the real world. This 'real world' is the universe of discourse, i.e. the domain or relevant part of the domain that must be modelled.
- A real-world event has no duration, i.e. it occurs or is recognised at one point in time.
- The real-world events that are identified during conceptual modeling are not further decomposable. They are defined at the lowest level of granularity and cannot be meaningfully split into other, more fine-grained, events.

It is common to model events as event types, rather than referring to specific event occurrences. Since real-world events are the focal point in event-based conceptual modeling, a notation is needed to designate the set of event types that is relevant for a particular universe of

[1] Strictly spoken, conceptual modeling is different from domain analysis or engineering. Domain analysis methods like FODA (Kang *et al.* 1990) are used to derive models that are common to a collection of individual organisations. An analysis of the similarities and differences between the individual enterprise models (also called business models) plays a crucial role in such methods. When we use the term 'domain' or 'domain model' in this paper, we mean a 'domain' in a more general sense. It can refer to a real domain model (e.g. stock/inventory management, front-office, manufacturing), but as well to an enterprise model that is specific to a particular organisation.

[2] In reality, objects do not pass messages to each other. For instance, if a person rents a car, then the person does not send a 'rent' message to the car, nor does the car send a 'rent' message to the person. However, both objects (person and car) are involved in the same real-world event, i.e. the renting of the car by the person. The event broadcasting mechanism simultaneously notifies all participating objects of the event occurrence, without deciding on an order yet as with message passing (e.g. the person notifies the car or the car notifies the person). Compared to message passing, the event broadcasting mechanism leads to more maintainable and reusable conceptual models (Snoeck and Poels 2000).

discourse. A capital ***A*** is used to denote the universe of event types associated with some universe of discourse. All event types relevant to the universe of discourse are elements of ***A***.

An example is presented of a simplified loan circulation process in the context of a library. Assume that the scope of the **LIBRARY** conceptual model is initially delimited such that the universe of event types is

A = {*start_membership, end_membership, acquire, catalogue, borrow, renew, return, sell, reserve, cancel, fetch, lose*}

A conceptual model also identifies the entities (persons, things, etc.) in the universe of discourse that participate in real-world events. Such entities are said to be 'relevant to' the universe of event types ***A***. In object-oriented conceptual modeling these entities are represented as objects. Objects are characterised as follows:

- Each object in the conceptual model corresponds to a real-world concept.
- Objects are described by a number of properties. The properties of an object are specified in an object type (e.g. a UML classifier with an <<object type>> stereotype).
- Objects exist for a certain period of time.
- An object always participates in at least two real-world events: a creating event and an ending event. The participation in the ending event does not imply that the object is physically destroyed. It means that the object can no longer participate in real-world events.

Objects have a state and a set of operations. Although the specific form of communication (e.g. message passing) is not relevant for conceptual modeling, we assume that for each type of event that an object participates in, there is an operation specified in the object type. The state of an object is represented by its values for the attributes that have been specified in the object type. These attributes must be seen as abstract attributes, i.e. they must not necessarily be stored attributes in the class definition of the object. The effect of an event participation is modeled by specifying how the operation that is triggered by the event, affects the state of the participating object. The set of operations / triggering event types for an object type is called its *alphabet*. It is a subset of the universe of event types.

Event participations are modeled using an object-event table (OET). The *type of involvement* of an event participation is create (C), modify (M), or end (E). A modifying event type for an object type does not create object instances of the type, nor does it end their lives. A modifying event may however change the state of an object. Table 1 contains the OET for **LIBRARY**. Apart from a type of involvement indication, we also indicate the *type of provenance* of an event participation. An operation / event type in the alphabet of an object type is either acquired through propagation (A) (cf. infra), inherited (I), or specialised, i.e. inherited in a specialised version (S) (cf. infra). The class of 'own' event types (O) completes the partitioning.

To formally define the measures in the next section, the notion of object-event table is formalised (Snoeck *et al*. 1999). The set of object types relevant to the universe of event types ***A*** is denoted by a capital ***T***.

Let ***A*** be the universe of event types and ***T*** be the set of object types.
The object-event table is a map τ: $\boldsymbol{A} \times \boldsymbol{T} \rightarrow \{O, A, S, I\} \times \{C, M, E\} \cup \{(' ',' ')\}$
When $\tau(e,P) = (R,J)$ with $R \in \{O, A, S, I, ' '\}$ and $J \in \{C, M, E, ' '\}$, we write that $\tau(e,P) = R/J$.
We define the partial maps τ_P and τ_I that return the type of provenance and the type of involvement as
τ_P: $\boldsymbol{A} \times \boldsymbol{T} \rightarrow \{O, A, S, I, ' '\}$ and τ_I: $\boldsymbol{A} \times \boldsymbol{T} \rightarrow \{C, M, E, ' '\}$

Event-based conceptual modeling is also concerned with modeling the static structure of the universe of discourse. This means that associations between object types, with their optionalities and cardinalities, are identified. A key feature of event-based conceptual modeling is that the effect of associations on event participation is explicitly modeled.

Table 1. OET for LIBRARY

	ITEM	VOLUME	COPY	RESERVATION	MEMBER	LOAN	NOT_RENEWABLE_LOAN	RENEWABLE_LOAN
acquire	O/C							
acquire_volume		S/C						
acquire_copy			S/C					
catalogue	O/M	I/M	I/M					
sell	O/E							
sell_volume		S/E						
sell_copy			S/E					
reserve			A/M	O/C	A/M			
cancel			A/M	O/E	A/M			
fetch			A/M	O/E	A/M			O/C
start_membership					O/C			
end_membership					O/E			
borrow					A/M	O/C		
create_not_renewable_loan		A/M			A/M		S/C	
create_renewable_loan			A/M		A/M			S/C
return		A/M	A/M		A/M	O/E	I/E	I/E
lose					A/M	O/E		
lose_volume		A/E			A/M		S/E	
lose_copy			A/E		A/M			S/E
renew			A/M		A/M			O/M

One type of association is specialisation. One object type can specialise another object type. A subtype inherits the alphabet of its supertype.[3] It may also extend this alphabet, by participating in additional types of event. We also allow for event type specialisation, i.e. objects of a subtype may participate in events of a type that specialises an event type in the alphabet of the supertype (cf. Table 1).[4]

For the other types of association we follow a modeling strategy that factors all associations into binary existence dependency associations (Snoeck *et al*. 1999). Such associations put special restrictions on optionalities and cardinalities.[5] Moreover, they allow a formal definition of 'propagation' of operations. It is required that an object (hereafter called the master object) participates in all events in which its existence dependent objects participate. All these event participations are propagated from the existence dependent object to the master object (cf. Table 1). The operations in the existence dependent object type are also propagated into the master object type.[6]

[3] By convention, the inherited operations are explicitly specified in the subtype. This does not mean that they must be implemented in the class definition of the subtype. Conceptual modeling is not concerned with issues regarding implementation inheritance.

[4] The operations of the supertype that are specialised are not included in the subtype. Note that some object-oriented analysis methods (e.g. Catalysis) do not support event type / operation specialisation.

[5] The association is mandatory and has a cardinality of one for the existence dependent object type. Moreover, an existence dependent object is during its life always associated to the same object.

[6] This does not mean that all propagated operations must also be implemented in the class definition of the master object type.

Figure 1 shows the structural model of **LIBRARY**. Note that the object type ITEM has two subtypes: VOLUME and COPY. If we assume that volumes can be borrowed, but their loans cannot be renewed, then the object type LOAN must also be specialised.

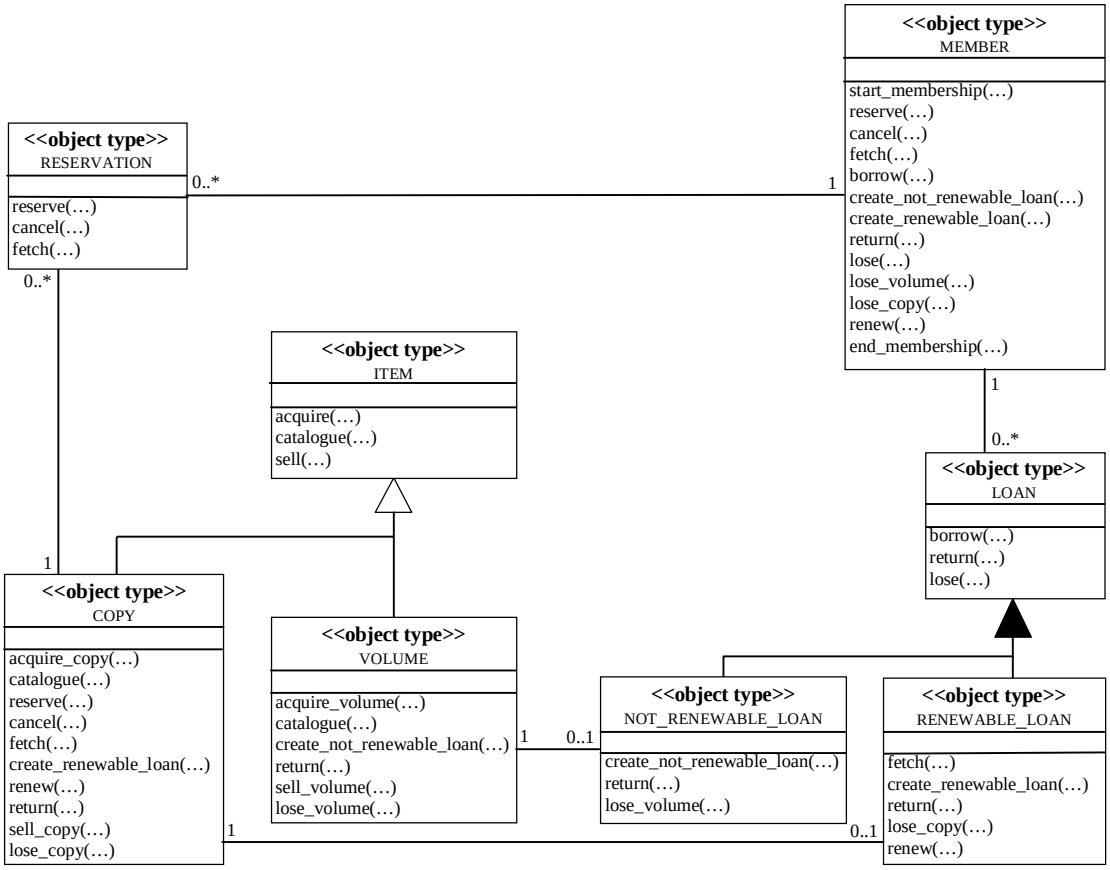

Figure 1. Structural model for LIBRARY

3. An OET-based measure suite

For the measure definitions, assume a universally qualified conceptual model S with universe of event types ***A***, set of object types ***T*** relevant to ***A***, and an object-event table τ. We use the symbol # for the cardinality of a set.

3.1 Size measures: Informally, the size of a software artifact is a function of the number of finer-grained elements that are used to define, specify, build or compose it. Size can be expressed and measured in terms of event participations. There are good reasons to do so. The more event types an object type is involved in, the more operations must possibly (but not necessarily) be implemented in the class definition. Hence, the count of event participations provides an early size estimate for classes. Early size estimates are useful (and essential) for project budgeting purposes. They are the basis for effort and cost estimates, and for pricing, outsourcing and scheduling decisions.

Table 2. Size measures based on the OET

MEASUREMENT OBJECT	MEASURE DESCRIPTION	MEASURE DEFINITION
Object type: $P \in \boldsymbol{T}$	Count of Event Participations	$CEP(P) = \#\{e \in \boldsymbol{A} \mid \tau(e,P) \neq \text{' '/' '}\}$
Conceptual model: S	Level of Object-Event Interaction	$LOEI(S) = \sum_{P \in \boldsymbol{T}} CEP(P)$

3.2 Coupling measures: Coupling can be described as the degree of interdependence between software artifacts (e.g. modules, classes, components, etc.). The main arguments in favour of low coupling are that the stronger the coupling between software artifacts, (i) the more difficult it is to understand individual artifacts, and hence to maintain them; (ii) the larger the extent of (unexpected) change and defect propagation effects across artifacts, and consequently the more testing required to achieve satisfactory reliability levels; (iii) the lower the reusability of individual artifacts. We therefore need to assess, and if needed reduce, the level of coupling in a software system. The earlier this is done the better.

Traditionally, coupling in object-oriented software has been measured in terms of message passing. In conceptual modeling we do not wish to decide yet whether object communication will be based on message passing. In our opinion, it might thus be useful to express coupling in terms of common event participations. Object types are then coupled if their instances participate in the same types of event.

Table 3. Coupling measures based on the OET

MEASUREMENT OBJECT	MEASURE DESCRIPTION	MEASURE DEFINITION
Object type: $P \in \boldsymbol{T}$	Count of Coupled Object types	$CCO(P) = \#\{Q \in \boldsymbol{T} - \{P\} \mid \exists\, e \in \boldsymbol{A}: \tau(e,P) \neq \text{' '/' '} \wedge \tau(e,Q) \neq \text{' '/' '}\}$
Conceptual model: S	Level of Object type Coupling	$LOC(S) = \sum_{P \in \boldsymbol{T}} CCO(P)$

The OET provides also the basis to measure specific types of coupling, related to the dynamic behaviour of objects. An example is synchronisation-based coupling. In **LIBRARY**, RESERVATION and RENEWABLE_LOAN are not (directly) related through associations (cf. Figure 1). However, their alphabets contain the common event type *fetch*. A RESERVATION object and a RENEWABLE_LOAN object synchronise their lives when they participate in the same *fetch* event (i.e. the *fetch* ends the life of a RESERVATION object and creates a new RENEWABLE_LOAN object).

Table 4. Measures for synchronisation-based coupling

MEASUREMENT OBJECT	MEASURE DESCRIPTION	MEASURE DEFINITION
Object type: $P \in \boldsymbol{T}$	Count of Synchronisation-based Coupled Object types	$CSCO(P) = \#\{Q \in \boldsymbol{T} - \{P\} \mid \exists\, e \in \boldsymbol{A}: (\tau_l(e,P) = C \wedge \tau_l(e,Q) = E) \vee (\tau_l(e,P) = E \wedge \tau_l(e,Q) = C)\}$
Conceptual model: S	Level of Synchronisation-based Object type Coupling	$LSOC(S) = \sum_{P \in \boldsymbol{T}} CSCO(P)$

3.3 Inheritance, specialisation and propagation measures: Several measures for quantifying the absolute or relative amount of inherited properties in an object-oriented system have been proposed. The measures proposed in the literature are generally design or code measures that consider the inheritance of class methods. During conceptual modeling, models are built using type definitions, rather than class definitions, and consequently it has not been decided yet which

operations must be implemented, or inherited, overridden, etc. Nevertheless, early estimates of the degree of inheritance can be obtained by considering the type of provenance of event participations. Generally, operations that correspond to 'own' or specialised event participations are implemented as class methods, whereas inherited operations are not (unless there is a need to override the method body in the subclass).

As far as we know, there are no measures to assess the degree of propagation of operations. Nevertheless, propagation of operations is not an exclusive characteristic of existence dependency associations. For instance, in the context of the IS-PART-OF relation operations might also propagate from the aggregate to the parts (e.g. cascading deletes). Having an idea of the (relative) amount of propagated operations in an object type is useful, as these operations must not necessarily be implemented as methods in the own class definition.

Table 5. Inheritance, specialisation and propagation measures based on the OET

MEASUREMENT OBJECT	MEASURE DESCRIPTION	MEASURE DEFINITION
Object type: $P \in \boldsymbol{T}$	Degree Of Inheritance Degree Of Specialisation Degree Of Propagation	$DOI(P) = \#\{e \in \boldsymbol{A} \mid \tau_P(e,P) = I\}/CEP(P)$ $DOS(P) = \#\{e \in \boldsymbol{A} \mid \tau_P(e,P) = S\}/CEP(P)$ $DOP(P) = \#\{e \in \boldsymbol{A} \mid \tau_P(e,P) = A\}/CEP(P)$
Conceptual model: S	Degree Of Inheritance Degree Of Specialisation Degree Of Propagation	$DOI(S) = \sum_{P \in \boldsymbol{T}} \#\{e \in A \mid \tau_P(e,P) = I\}/LOEI(S)$ (analogue definitions for DOS and DOP)

3.4 Polymorphism measures: Literally, polymorphism refers to the ability to take different forms. The general idea of polymorphism is that different classes define a method with the same name and signature, but with a different implementation. Mostly, polymorphism is then considered in the context of inheritance, overriding and dynamic binding. However, methods with the same name and signature may also appear in classes not related through inheritance.

A potential polymorphic situation exists when more than one object type is involved in the same event type. So, the number of potential polymorphic situations is easily measured from the perspective of the event types. The type of provenance indications further allow to distinguish specific types of polymorphism, like non-inheritance-related polymorphism.

The following table presents the degree of polymorphism measures. They are relative measures, i.e. they relate the actual number of potential polymorphic situations of the type considered to the maximum number of such situations. Note that polymorphism is only defined for conceptual models.

Table 6. Polymorphism measures based on the OET

MEASUREMENT OBJECT	MEASURE DESCRIPTION	MEASURE DEFINITION
Conceptual model: S	Degree of POlymorphism	$DPO(S) = (\sum_{e \in \boldsymbol{A}} \#\{P \in \boldsymbol{T} \mid \tau(e,P) \neq \text{' '/' '}\} - \#\boldsymbol{A}) / LOEI(S)$
	Degree of Non-inheritance-related POlymorphism	$DNPO(S) = (\sum_{e \in \boldsymbol{A}} \#\{P \in \boldsymbol{T} \mid \tau_P(e,P) \in \{O, A, S\}\} - \#\boldsymbol{A}) /$ $(LOEI(S) - \sum_{e \in \boldsymbol{A}} \#\{P \in \boldsymbol{T} \mid \tau_P(e,P) = I\})$

4. Conclusions and topics for further research

This paper presented part of a framework for the measurement of object-oriented conceptual models. Conceptual modeling is part of the requirements engineering process and as such our work addresses the need for measurement support in the early stages of system development.

The cornerstone of the measurement framework is the object-event table. In fact, to measure characteristics related to the data, function, as well as dynamic behaviour dimensions of object-oriented conceptual models, the framework assumes that the object-event interactions are somehow modeled. Information regarding such interactions is normally available when modeling and analysing a domain. But even if a method does not prescribe the use of an object-event table or an equivalent object-event interaction model, this information can easily be derived, as long as the method does not ignore behavioural aspects altogether.

This paper did not address the issue of measure validity. Regarding the theoretical validity of the measures we must note that all direct measures have been developed using a Measurement Theory-based approach described in (Poels and Dedene 2000). Regarding the usefulness of the measures we must note that a series of empirical investigations has been planned. It would be useful to examine whether the measures can indeed be used as early effort and quality predictors, and whether dynamic aspects of software are related to these variables. Preliminary results of a first experiment will be presented at the workshop.

Acknowledgements

Geert Poels is a Postdoctoral Fellow of the Fund for Scientific Research - Flanders (Belgium)(F.W.O.) and wishes to acknowledge the financial support of the Fund for Scientific Research.

References

L. Badri, M. Badri, and S. Ferdenache, "Towards Quality Control Metrics for Object-Oriented Systems Analysis", *Proceedings of TOOLS Europe'95*, Versailles, France, March 1995, pp. 193-206.

L. Briand, E. Arisholm, S. Counsell, F. Houdek, and P. Thévenod-Fosse, "Empirical Studies of Object-Oriented Artifacts, Methods, and Processes: State of The Art and Future Directions", Technical Report IESE 037.99/E, Fraunhofer Institute for Experimental Software Engineering, Kaiserslautern, Germany, 1999.

F. Brito e Abreu, H. Zuse, H. Sahraoui, and W. Melo, "Quantitative Approaches in OO Software Engineering", ECOOP'99 Workshop Reader, Lecture Notes in Computer Science, Springer Verlag, 1999.

S.R. Chidamber and C.F. Kemerer, "A Metrics Suite for Object Oriented Design", *IEEE Transactions on Software Engineering*, Vol. 20, No. 6, June 1994, pp. 476-493.

D.F. D'Souza and A.C. Wills, *Objects, Components, and Frameworks with UML: the Catalysis Approach*, Addison-Wesley, 1999.

M. Genero, M.E. Manso, M. Piattini, and F.J. Garcia, "Assessing the Quality and the Complexity of OMT Models", *Proceedings of the 2nd European Software Measurement Conference*, Amsterdam, October 1999, pp. 99-109.

I. Graham, *Migrating to Object Technology*, Addison-Wesley, 1995.

K.C. Kang, S.G. Cohen, J.A. Hess, W.E. Novak, and A.S. Peterson, "Feature-Oriented Domain Analysis (FODA) Feasibility Study", Technical Report CMU/SEI-90-TR-21 (ESD-90-TR-222), Software Engineering Institute, Carnegie-Mellon University, Pittsburgh, Pennsylvania, USA, November 1990.

G. Poels and G. Dedene, "Distance-based software measurement: necessary and sufficient properties for software measures", *Information and Software Technology*, Vol. 42, No. 1, January 2000, pp. 35-46.

M. Snoeck, G. Dedene, M. Verhelst, and A. Depuydt, *Object Oriented Enterprise Modelling with MERODE*, Academic Press Leuven, Belgium, 1999.

M. Snoeck and G. Poels, "Improving the Reuse Possibilities of the Behavioural Aspects of Object-Oriented Domain Models", 2000, submitted for publication.

H. Zuse, *A Framework for Software Measurement*, Walter de Gruyter, Berlin, 1998.