KATHOLIEKE
UNIVERSITEIT
LEUVEN

DEPARTEMENT TOEGEPASTE ECONOMISCHE WETENSCHAPPEN

RESEARCH REPORT 0029

MEASURES FOR ASSESSING DYNAMIC COMPLEXITY ASPECTS OF OBJECT-ORIENTED CONCEPTUAL SCHEMES

by
G. POELS
G. DEDENE

D/2000/2376/29

Measures for Assessing Dynamic Complexity Aspects of Object-Oriented Conceptual Schemes

Geert Poels, Guido Dedene

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

Abstract. System developers are increasingly realising that the quality of a system must be ensured in the early stages of the development life cycle. It is in this context that a number of quality frameworks for conceptual schemes have been proposed. However, before the quality of a conceptual schema can be improved, it must be assessed. Accordingly, a number of measure suites have been proposed for measuring quality properties of conceptual schemes. In this paper we focus on one particular quality property, i.e. complexity. This property can be described as the mental burden of the persons that must understand, modify, extend, verify, implement, and reuse conceptual schemes. The proposed complexity measures for conceptual schemes have in common that they only capture the complexity of the static or structural aspects of a conceptual schema. We therefore present a complementary suite of measures that focuses on conceptual schema complexity as seen from a dynamic perspective.

1 Introduction

Conceptual modeling is an integral part of modern approaches towards system development, like Catalysis [1] and the Rational Unified Process [2]. The conceptual schema is not merely the basis for modeling the persistent system data. In object-oriented modeling, where data and process are closely linked, conceptual schemes provide the solid foundation for the design and implementation of information systems.

As an early available, key analysis artifact the quality of the conceptual schema is crucial to the success of system development. Generally, problems in the artifacts produced in the initial stages of system development propagate to the artifacts produced in later stages, where they are much more costly to identify and correct [3]. Therefore, the quality of conceptual schemes must be evaluated, and if needed improved.

This paper must be seen in the context of a measurement-based approach towards quality control for object-oriented conceptual schemes. Before quality properties can be evaluated, their values must be assessed, either by subjective expert ratings or by objective measurements. In this paper we present a formally defined measure suite to quantify various aspects related to one particular, but highly important quality property of conceptual schemes, i.e. their simplicity. We consider simplicity as a 'quality' because its inverse, complexity, has been shown, both theoretically and empirically (e.g. [4]), to be detrimental to the ability to understand, modify, extend, verify, implement, and reuse system development artifacts.

Our measure suite differs from other suites of conceptual schema measures in the sense that it takes dynamic views on the conceptual schema into account. It extends previous work on measuring entity relationship schemes and object relationship schemes. Whereas the ER-related work focused on complexity aspects of logic data schemes, which are static by nature, the OR-related research produced complexity measures for static object schemes that result from object-oriented (domain) analysis activities. Some of these measures capture the functionality that is encapsulated in the objects, e.g. in terms of the (public) operations defined and/or inherited. In general however, no measures have been proposed to assess complexity aspects related to the functional and dynamic behaviour dimensions of conceptual schemes.

The measure suite presented in this paper is based on a formal model of object functionality and behaviour that uses the notion of event. The modeling of events and the participation of objects in these events introduces a dynamic perspective on conceptual modeling. It allows expressing complexity measures in terms of object interaction (e.g. when objects participate in the same event) and in terms of object life cycle specifications (e.g. sequence constraints on the participation in

events). Our measure suite thus complements the previously proposed measures for 'static' complexity aspects of object-oriented conceptual schemes.

This paper is organised as follows. Section 2 reviews quality models for conceptual schemes and discusses the role of complexity as a quality property. Section 3 reviews previous work on conceptual schema measures. In section 4 we introduce the cornerstone of our formal model of object functionality and behaviour: the object type – event type association matrix. Next, in section 5 the measure suite is presented. Section 6 briefly discusses the complex issue of measure validation and touches upon the theoretical and empirical validity of the proposed measures. Finally, section 7 presents conclusions.

2 Quality and Complexity in Conceptual Modeling

From a systems theory point of view, a system is called complex if it is composed of many (different types of) elements, with many (different types of) (dynamically changing) relationships between them. In software engineering, it is well accepted by now that no single definition or measure can capture all possible aspects of complexity [5]. Nevertheless, the relationship between individual complexity aspects (e.g. information flow complexity [6]), also called 'internal' quality properties of a system development artifact, and 'external' quality properties like reliability, reusability and maintainability has been investigated with the purpose of building software quality prediction, evaluation and control models [7].

Compared to software engineering, the concept of quality in conceptual modeling is poorly understood [8]. Only a few comprehensive and structured quality evaluation frameworks have been proposed that provide more than a pure listing of desirable quality properties. These proposals include the frameworks of Lindland et al. [9] and Moody et al. [10]. The framework of Lindland et al. uses linguistic concepts to distinguish between three types of conceptual schema quality: syntactic quality (i.e. the degree to which the rules of the modeling technique are adhered to), semantic quality (i.e. the degree to which the schema corresponds to the domain it models), and pragmatic quality (i.e. the degree to which users understand the schema). Within the bounds set by the complexity of the rules of the modeling technique and the complexity of the domain that must be modeled, complexity, or better, its inverse simplicity, must be seen as a pragmatic quality aspect. The framework of Moody et al. considers in its revised form eight quality factors, one of them being simplicity.

According to Moody [11] a data schema is characterised by simplicity if it contains the minimum possible constructs in terms of entities, relationships and attributes. Schema size, in terms of object types and attributes, has also been considered as an aspect of pragmatic quality by Assenova and Johannesson [8]. However, they acknowledge that the lowest possible size is not necessarily a 'quality' of a conceptual schema. Also, Lindland et al. [9] argue that

structuredness might be more important for pragmatical quality than 'expressive economy'. These findings are consistent with the notions of complexity that have been used in software engineering research, where more emphasis is laid on structural aspects such as coupling, cohesion, depth and width of the inheritance lattice, etc [7].

Our model of conceptual schema complexity, its relationship with size and structure on the one hand, and 'external' quality properties on the other hand, is based upon similar models used in software engineering research [12], [4]. Fig. 1 shows for instance the complexity model of Briand et al. [12]. The structural properties of a software engineering artifact affect the 'cognitive' complexity of the artifact, i.e. the mental burden of the persons who have to deal with the artifact (e.g. developers, testers, maintainers). According to Briand et al. it is the, sometimes necessary, high complexity of an artifact which causes it to display undesirable external qualities.

The complexity model of Briand et al. is the basis for much empirical research in the area of software artifact complexity. In this paper we assume a similar model to hold for conceptual schemes, given that they are artifacts used in the initial stages of system development.

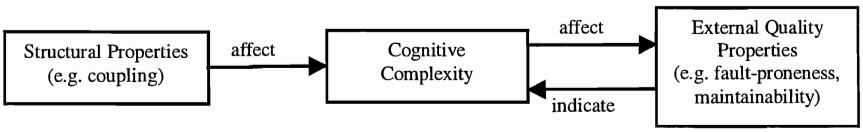

Fig. 1. The complexity model for system development artifacts of Briand et al. [12]

3 Previous Work on Conceptual Schema Measures

Lindland et al. [9] mention inspection, visualisation, animation, explanation, simulation and filtering as techniques for improving pragmatical quality. These techniques help to understand a conceptual schema without modifying it. Another technique is to transform a schema in order to improve its pragmatic quality properties (e.g. complexity) [8, 13]. Such schema transformations require pragmatic quality properties to be assessed, both before and after the schema is transformed, to evaluate the effectiveness of a transformation. It is in this context, and to assess conceptual schema quality in general, that measurement instruments in the form of rating scales and measures have been proposed.

A number of researchers have proposed measurement instruments for entity relationship schemes. Moody [11] presents twenty-five 'metrics' for assessing the quality factors of entity relationship schemes identified in [10]. These 'metrics' are a mix of rating scales, cost figures or estimates, and counts. The latter include

counts based on subjective expert knowledge (e.g. number of items in the data model that do not correspond to user requirements). However, the measures proposed for simplicity (i.e. number of entities, number of entities and relationships, number of entities, relationships and attributes) are objective and 'automatable' counts. A similar suite of twelve complexity measures for ER schemes has been presented by Genero et al. [14].

Moser and Misic [15] propose size, coupling and cohesion measures for object-oriented conceptual schemes (also called business models), which are based on a formal and generic object model described in [16]. Badri et al. [17] and Genero et al. [18] present complexity measures for the object-oriented analysis schemes of more specific development methods, respectively OOA [19] and OMT [20]. Most of the object-oriented software measures proposed in the literature (e.g. the MOOSE measures [21], the MOOD measures [22]) are however design or code measures that capture aspects not relevant for conceptual modeling. Briand et al. [23, 24] have shown that at least a few of these measures could also be used as specification or analysis measures.

The complexity measures for object relationship diagrams take size and structure aspects related to object operations into account. Compared to the measure suites for ER schemes they measure more than data schema quality. However, they are based on a static object model and do not capture the complexity of the dynamic perspective. The measure suite presented in this paper is meant to remedy this situation and complements the existing measures for (object-oriented) conceptual modeling.

4 Event-Driven Object-Oriented Conceptual Modeling

Objects in a domain are affected by the occurrence of events. As an example, consider an ORDER object that can be placed, changed, delivered, invoiced, paid, etc. In event-driven conceptual modeling a dynamic perspective on the domain is taken. Objects, relationships, rules and constraints are modelled starting from the things that happen, i.e. the events. In this section we present a formal model of objects and events based on an 'archetype' method, described in [25, 26] that is sufficiently abstract and generic to capture the main aspects of event-driven conceptual modeling.

Fig. 2 shows an extract of our meta-model (in UML notation [27]) for event-driven object-oriented conceptual modeling. Modeling starts by identifying the different types of event that are relevant to the Universe of Discourse (i.e. the domain). Hereafter, a capital $\boldsymbol{A}$ is used to denote the universe of event types relevant for the UoD. It must be noted that some methods (e.g. Catalysis [1]) model events only indirectly, via the action concept. Actions are different from events in the sense that they have a duration. However, actions can easily be transformed into events: both the beginning and ending of an action qualify as events.

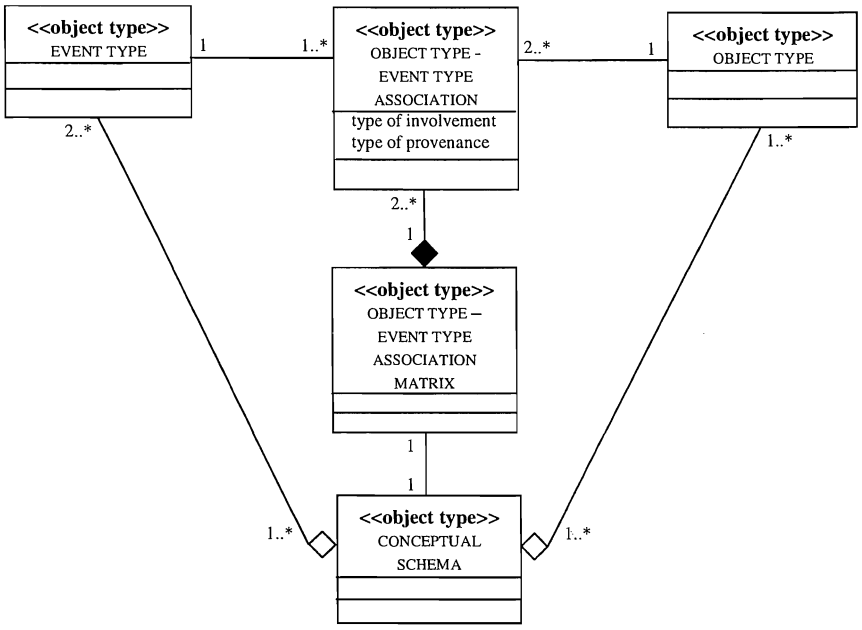

Fig. 2. A partial view on the meta-model for event-driven object-oriented conceptual modeling

The persons, things, etc. in the UoD that are involved in the occurrences of the event types in $\boldsymbol{A}$ are modelled as objects. Objects are described by a number of properties, which are specified in an object type. We assume that all objects identified during conceptual modeling are persistent, i.e. they have a state, represented at any moment by the values of their attributes, and they exist for a certain period of time. Objects therefore always participate in at least two 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. The set of object types relevant to the universe of event types $\boldsymbol{A}$ is denoted by a capital $\boldsymbol{T}$.

In event-driven conceptual modeling we specify an operation in the object type for each type of events that the instances of the object type can participate in. When an object participates in an event then the corresponding operation is triggered, which (possibly) changes the state of the object. It must be noted that in an object-oriented implementation of the conceptual schema, not all operations must effectively be implemented as methods in the class definition of the object type because of mechanisms like inheritance and delegation.

This dynamic perspective on conceptual modeling is captured in the so-called object type – event type association matrix [25]. Each conceptual schema has one

object type – event type association matrix composed of object type - event type associations. Such an association relates one event type in $\boldsymbol{A}$ with one object type in $\boldsymbol{T}$ and means that instances of the object type participate in occurrences of the event type. It has two attributes. The type of involvement specifies whether an event participation creates an object (value: 'C'), ends the life of an object (value: 'E'), or just modifies the life cycle state of the object (value: 'M'). A 'modifying' event type can, but does not necessarily change the visible object state (i.e. the attribute values). The type of provenance specifies whether the object type - event type association has been inherited (value: 'I') from an ancestor, has been acquired through propagation (value: 'A'), or is a newly defined (or 'own') association (value: 'O'). The formal definition of the object type - event type association matrix is taken from [25]:

> For some UoD, let $\boldsymbol{A}$ be the universe of event types and $\boldsymbol{T}$ be the set of object types.
> The object type - event type association matrix is a map
> $\tau: \boldsymbol{A} \times \boldsymbol{T} \rightarrow \{O, A, I\} \times \{C, M, E\} \cup \{(' ', ' ')\}$.
> When $\tau(e,P) = (R,J)$ with $R \in \{O, A, I, ' '\}$ 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 $\tau_p: \boldsymbol{A} \times \boldsymbol{T} \rightarrow \{O, A, I, ' '\}$ and $\tau_i: \boldsymbol{A} \times \boldsymbol{T} \rightarrow \{C, M, E, ' '\}$.

Table 1 shows an example object type - event type association matrix for a (simplified) library. Consistent with the rules proposed in [25], an object type - event type association is propagated from an object type P to an object type Q if objects of P are existence dependent on objects of Q.[1]

The object type - event type association matrix is the cornerstone of our formal model of object functionality and behaviour. It allows expressing complexity aspects of conceptual schemes in terms of (common) event participation. The basic underlying conjecture is that, all other things being equal, the more types of event that an object participates in, and the more objects of different types that participate in a same event, the more complexity is added to the conceptual schema. One of the motivations for this conjecture is that when objects participate in an event, interesting things happen in both the domain and the information system: business rules and constraints are checked (i.e. is the event participation allowed?), operations are triggered, object states are changed, etc. This is especially relevant when two or more objects jointly participate in an event (e.g. when a member of the library makes a reservation for a copy, an instance of MEMBER and an instance of COPY participate in a *reserve* event). Such a joint participation synchronises the lives of the participating objects, might lead to the creation or ending of instances of other object types (e.g. *reserve* creates an instance of RESERVATION), might necessitate checking additional rules (e.g. a reservation is refused if the copy is on shelf), etc.

The type of involvement values of the object type - event type associations help to derive the dynamic behaviour of objects. They specify a default life cycle.

[1] For a formal and elaborate definition of 'existence dependency' we refer to [26].

First, a choice is made between the creating event types to create an object instance. Next, the state of the object may be modified zero, one or more times, using any type of modifying event. Finally, a choice is made between the ending event types to end the life of the object. For instance, if sequence, selection and iteration are denoted using the ".", "+", and "*" symbols respectively, then a default life cycle for a RENEWABLE_LOAN object is specified by (*borrow* + *fetch*) . *renew** . (*return* + *lose*).

Table 1. Object type - event type association matrix for a simplified library

	ITEM	VOLUME	COPY	RESERVATION	MEMBER	LOAN	NOT_RENEWABLE_LOAN	RENEWABLE_LOAN
acquire	O/C	I/C	I/C					
catalogue	O/M	I/M	I/M					
sell	O/E	I/E	I/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	A/M		A/M	O/C	I/C	I/C
return		A/M	A/M		A/M	O/E	I/E	I/E
lose		A/E	A/E		A/M	O/E	I/E	I/E
renew			A/M		A/M			O/M

The domain might impose additional constraints on the life cycle of objects. We might for instance require that a copy can only be borrowed if it has been catalogued first. The diagrams (e.g. Finite State Machines) or mathematical expressions used to specify life cycle schemes, other than the default ones, are not shown in the meta-model of Fig. 2. We use them to complement the object type - event type association matrix when measuring complexity aspects related to the dynamic behaviour of objects.

5 Complexity Measures for Event-Driven Object-Oriented Conceptual Modeling

The suite of measures presented here assesses various complexity aspects related to the size and structure of a conceptual schema from the dynamic perspective described in the previous section. Most of these measures simply require querying the object type - event type association matrix. A single one requires additional information that is contained in the object life cycle specifications. As the measure definitions are formulated in terms of object type - event type associations (i.e. event participations), instead of attributes, relationships, or operations, they complement, but do not necessarily substitute, the previously published measure suites for conceptual modeling that were reviewed in section 3.

We do not claim that the measure suite is complete. In fact, due to space limitations, we had to limit the number of measure definitions. For some extra measures, related to polymorphic behaviour aspects, we refer to [28].

For the measure definitions, assume a universally qualified conceptual schema S with universe of event types A, set of object types T, and an object type - event type association matrix τ. We use the symbol # for the cardinality of a set.

5.1 A Size Measure

The size of a schema has been defined as the number of object types and attributes [8] or the number of constructs (i.e. entities, relationships and attributes) [11]. Analogously, we define it here as the number of object type - event type associations specified in the object type - event type association matrix.

A size measure for conceptual schemes is the Level of Event Participation (LEP). It returns the count of non-empty cells in the object type - event type association matrix. The LEP measure is given by the following equation:

$$LEP(S) = \sum_{P \in T} \#\{e \in A \mid \tau(e,P) \neq ' ' \text{ '/' } '\} \; . \qquad (1)$$

The value returned for the library example is 42. Note that the size of a conceptual schema is related to the size of the information system (though not necessarily in terms of data volume). The more types of event an object is involved in, the more operations must possibly (but not necessarily) be implemented in the class definition of the object type. Hence, LEP can be used to derive an early, albeit rough, estimate of the size of the information system (e.g. in terms of lines of code). 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.

5.2 Structure Measures

An aspect of structure that has received a lot of attention in software engineering is coupling. Coupling has been described as the degree of interdependence between system development artifacts (e.g. object classes) [7]. The main arguments in favour of low coupling are that the stronger the coupling between 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.

In object-oriented software, coupling has mostly been measured in terms of message passing [24]. 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.

A coupling measure for conceptual schemes is the Level of Object Type Coupling (LOTC). It counts for each object type P the number of other types of object that participate in a same type of event as the instances of P, and then adds these counts. The equation for the LOTC measure is:

$$\text{LOTC}(S) = \sum_{P \in \boldsymbol{T}} \#\{Q \in \boldsymbol{T} - \{P\} \mid \exists\, e \in \boldsymbol{A}: \tau(e,P) \neq \text{' '/' '} \wedge \tau(e,Q) \neq \text{' '/' '}\} \,. \quad (2)$$

The value returned for the library example is 40. A normalised version of this measure is the Degree of Object Type Coupling (DOTC). It relates the actual LOTC value to the theoretical maximum LOTC value given the number of object types in the schema. The DOTC measure is given by the following equation:

$$\text{DOTC}(S) = \text{LOTC}(S) \,/\, (\#\boldsymbol{T}.(\#\boldsymbol{T} - 1)) \,. \quad (3)$$

The DOTC value for the library is 40 / 56 = 0.71 . In object-oriented analysis and design, coupling between object types has also been defined in terms of the number of associations and generalisation/specialisation relationships with other object types in the schema. This type of coupling is called association-based or static coupling [29]. In the context of conceptual modeling, association-based coupling has not been measured in terms of its effect on dynamic aspects, like the inheritance and propagation of event participations. To assess the extent of inheritance and propagation in a conceptual schema we propose the following measures.

The Level of Inheritance of Event Participation (LIEP) returns the count of inherited object type - event type associations. The Level of Propagation of Event Participation (LPEP) returns the count of object type - event type associations that have been acquired through propagation. The equations for LIEP and LPEP are:

$$\mathrm{LIEP}(S) = \sum_{P \in T} \#\{e \in A \mid \tau_P(e,P) = I\} \quad (4)$$

$$\mathrm{LPEP}(S) = \sum_{P \in T} \#\{e \in A \mid \tau_P(e,P) = A\} . \quad (5)$$

The respective values for library are 12 and 17. Normalised versions of these measures are the Degree of Inheritance of Event Participation (DIEP) and the Degree of Propagation of Event Participation (DPEP). The equations are:

$$\mathrm{DIEP}(S) = \mathrm{LIEP}(S) / \mathrm{LEP}(S) \quad (6)$$

$$\mathrm{DPEP}(S) = \mathrm{LPEP}(S) / \mathrm{LEP}(S) . \quad (7)$$

The respective values for library are 12 / 42 = 0.29 and 17 / 42 = 0.40 . As opposed to DOTC, the values of DIEP and DPEP can never be equal to one. There must always be some object type - event type associations that are neither inherited, nor acquired through propagation.

5.3 Measures for Dynamic Behaviour Complexity

Objects synchronise their lives when they jointly participate in an event. Some of these synchronising events are special in the sense that they both create and end objects. The object types involved in such event types are in a way coupled, but this coupling is not captured by measures for static coupling, nor by the inheritance and propagation measures defined in the previous subsection. We therefore define here a measure of synchronisation-based coupling. We say that an object type P is synchronisation-based coupled with an object type Q if there is an event that ends the life of an instance of P and creates an instance of Q, or vice versa.

The Level of Synchronisation-based Coupling (LSC) measures the extent of synchronisation-based coupling in a conceptual schema. It counts for each object type P the number of other object types it is synchronisation-based coupled with, and then adds these counts. The degree of Synchronisation-based Coupling (DSC) normalises the value of LSC by relating it to the Level of Object Type Coupling. The LSC and DSC measures are given by the following equations:

$$\mathrm{LSC}(S) = \sum_{P \in T} \#\{Q \in T - \{P\} \mid \exists\, e \in A: (\tau_I(e,P) = C \wedge \tau_I(e,Q) = E) \vee (\tau_I(e,P) = E \wedge \tau_I(e,Q) = C)\} \quad (8)$$

$$\mathrm{DSC}(S) = \mathrm{LSC}(S) / \mathrm{LOTC}(S) . \quad (9)$$

The values for LSC and DSC in the library example are 2 and 0.05 . Only RESERVATION and RENEWABLE_LOAN are synchronisation-based coupled through *fetch* events.

The final measure we present here is somewhat different from the rest. It compares the object life cycle specifications (e.g. Finite State Machines) with the default life cycles as specified in the object type - event type association matrix. The greater the difference between the two, the more sequence constraints apply to the participation of objects in events, and thus the higher the complexity of the dynamic behaviour of objects.

This particular aspect of complexity is called object life cycle complexity. The Object Life Cycle Complexity (OLCC) measure takes the form of a distance measure. The greater the distance between the actual life cycle specifications and the default life cycle specifications, the higher the value of OLCC. The elaboration of the definition of the OLCC measure is outside the scope of this paper and has been published previously [30]. We therefore only present an informal definition here, based on the library example.

To keep things simple, assume that in the library conceptual schema only ITEM and its specialisations have a non-default life cycle specification. The default life cycle specification for ITEM, based on the type of involvement indications in Table 1, is *acquire . catalogue* . sell*, implying that between acquiring and selling, the item can be catalogued zero, one or more times. However, the actual life cycle specification is more restricted: there must be exactly one participation in a *catalogue* event, i.e. the iteration on *catalogue* events must be dropped. For the specialisations of ITEM, i.e. VOLUME and COPY, it is required that they are catalogued before being borrowed for the first time. Hence, the respective life cycle specifications are *acquire . catalogue . (borrow + return)* . (lose + sell)* and *acquire . catalogue . (borrow + renew + return + reserve + cancel + fetch)* . (lose + sell)*.

In [30] we have proposed a set of elementary life cycle specification transformations for which it is proven that they can be used to express the distance between two life cycle specifications. Basically, these elementary life cycle specification transformations involve adding an event type in sequence or selection, removing an event type from a sequence or selection, and adding or removing iteration operators. Object life cycle complexity is then defined as the minimum number of such transformations needed to transform the actual life cycle specification in the default life cycle specification (or vice versa). The OLCC measure returns the sum of this minimum number of transformations, over all object types. In the example, one transformation is needed for ITEM (i.e. adding an iteration operator on *catalogue*) and two transformations are needed for VOLUME and COPY (i.e. removing *catalogue* from the sequence and adding it to the selection of 'modifying' event types). Hence, the value of OLCC for library is 5.

6 Some Observations on Measure Validity

In order to be credible and useful, proposed measures for system development artifacts must be validated, both theoretically and empirically [31]. A measure is theoretically valid if it measures what it is purported to measure. Zuse [32] advocates the use of measurement theory [33] as a reference framework for the theoretical validation of software measures. All measures presented in this paper have been validated using a specific measurement theoretic structure, i.e. the segmentally additive proximity structure [34]. Basically, all measures have initially been developed as distance measures that measure the difference with respect to the property of interest (e.g. size) between the artifact (e.g. a conceptual schema) and an hypothetical 'reference' artifact showing the theoretical lowest value for the property (e.g. an empty schema). The difference (or distance, dissimilarity) between these two artifacts is then measured by counting the minimum number of elementary transformations that are needed to transform one artifact into the other (cf. our discussion of the OLCC measure in the previous section). The details of this validation process are beyond the scope of this paper, but can be found in [35].

Equally important is the empirical validation of the measures. Basically this means that we must gather empirical evidence on the relationship between the various complexity aspects (i.e. size, structure, dynamic behaviour) that are measured and 'external' quality properties (cf. Fig. 1). As far as we know, no comprehensive empirical validation study in the area of conceptual schema quality has been published yet. Moody [11] proposes action research to refine the measures he has proposed. Genero et al. [14] use a case study to claim that some of their ER schema complexity measures correlate well with the maintainance time of the application programs that manage the data conceptually represented in the ER schemes. Another validation strategy is to use schema transformations, like the ones proposed in [13], to validate measures. The basic hypothesis underlying this type of study is that schema transformations improve the quality of the conceptual schema, and thus the complexity values returned by the measures should be lower after the transformations than they were before. It must be noted however that quality criteria, including objective measures, have also been used to show that schema transformations improve the quality of the schemes [8].

Currently, we have only gathered limited evidence of the empirical validity of our measures. Some of the complexity and distance measures have been applied in the context of a reference framework for conceptual schemes of an organisation's front-office to investigate their potential as indicators of perceived complexity and reengineering impact [36]. However, we were not able yet to draw definite conclusions regarding their empirical validity. We must note however that for many software engineering artifacts, the impact of size and structural properties on external quality properties has been demonstrated [37]. Examples include the relationship between object class size and defects found

[38], the negative effect of coupling on fault-proneness and reusability in object-oriented software [12, 39, 40] and object-based software [41], the impact of the morphology of the inheritance structure on the quality of software [42], and the relationship between the extent of polymorphism in a system and its probability of containing faults [43]. Although the external validity of these empirical studies in the context of conceptual schema quality must still be properly investigated, they do provide an indication of the potential importance and relevancy of many of the complexity aspects for which measures were proposed in this paper.

7 Conclusions

This paper presents a suite of measures to assess size, structure, and dynamic behaviour aspects of the complexity of object-oriented conceptual schemes as seen from a dynamic perspective. This measure suite is based on a formal model of object functionality and behaviour that is obtained using an event-driven approach to conceptual modeling. We related the new complexity measures to existing quality frameworks for conceptual modeling and to existing measure suites for entity relationship schemes and object-oriented (domain) analysis schemes and we showed the complementary nature of our measures.

We also noted the lack of a comprehensive empirical validation study in the area of conceptual schema quality. The work presented in this paper is part of a project investigating the effect of complexity aspects of early system development artifacts on the 'external' quality properties of information systems (e.g., maintainability, reusability). The ultimate goal of this project is to build early quality prediction, evaluation and control models. Our further research therefore includes a number of empirical investigations that will use the measures presented in this paper as part of its measurement instrumentation.

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 F.W.O.

References

[1] D'Souza, D.F., Wills, A.C.: Objects, Components, and Frameworks with UML: the Catalysis Approach. Addison-Wesley (1999)

[2] Rational Software: Object Oriented Analysis and Design, Student Manual (1998) http://www.rational.com/

[3] Boehm, B.W.: Software Engineering Economics. Prentice-Hall (1981)

[4] Tegarden, D.P., Sheetz, S.D., Monarchi, D.E.: A Software Complexity Model of Object-Oriented Systems. Decision Support Systems: An Int'l J. 13 (1995) 241-262

[5] Fenton, N.: Software Measurement: A Necessary Scientific Base. IEEE Trans. Software Eng. 20 (1994) 199-206
[6] Shepperd, M., Ince, D.: Algebraic Validation of Software Metrics. In: Proc. 3rd European Software Eng. Conf. (ESEC'91). Milan (1991) 343-363
[7] Fenton, N.E., Pfleeger, S.L.: Software Metrics: A Rigorous & Practical Approach. PWS Publishing Company, London (1997)
[8] Assenova, P., Johannesson, P.: Improving Quality in Conceptual Modelling by the Use of Schema Transformations. In: Proc. 15th Int'l Conf. Conceptual Modeling (ER'96). Cottbus, Germany (1996) 277-291
[9] Lindland, O.I., Sindre, G., Solvberg, A.: Understanding Quality in Conceptual Modeling. IEEE Software 11 (1994) 42-49
[10] Moody, D.L., Shanks, G.G., Darke, P.: Improving the Quality of Entity Relationship Models - Experience in Research and Practice. In: Proc. 17th Int'l Conf. Conceptual Modeling (ER'98). Singapore (1998) 255-276
[11] Moody, D.L.: Metrics for Evaluating the Quality of Entity Relationship Models. In: Proc. 17th Int'l Conf. Conceptual Modeling (ER'98). Singapore (1998) 211-225
[12] Briand, L.C., Wüst, J., Ikonomovski, S., Lounis, H.: A Comprehensive Investigation of Quality Factors in Object-Oriented Designs: an Industrial Case Study. In: Proc. 21st Int'l Conf. Software Eng. (ICSE'99). Los Angeles (1999) 345-354
[13] McBrien, P., Poulovassilis, A.: A Formal Framework for ER Schema Transformation. In: Proc. 16th Int'l Conf. Conceptual Modeling (ER'97). Los Angeles (1997) 408-421
[14] Genero, M., Piattini, M., Calero, C.: An Approach to Evaluate the Complexity of Conceptual Database Models. In: Proc. 3rd European Software Measurement Conf. Madrid (2000)
[15] Moser, S., Misic, V.B.: Measuring Class Coupling and Cohesion: A Formal Metamodel Approach. In: Proc. Asia Pacific Software Eng. Conf. (APSEC'97). Hong Kong (1997) 31-40
[16] Misic, V.B., Moser, S.: Formal Approach to Metamodeling: A Generic Object-Oriented Perspective. In: Proc. 16th Int'l Conf. Conceptual Modeling (ER'97). Los Angeles (1997) 243-256
[17] Badri, L., Badri, M., Ferdenache, S.: Towards Quality Control Metrics for Object-Oriented Systems Analysis. In: Proc. 16th Int'l Conf. Technology of Object-Oriented Languages (TOOLS-16). Versailles, France (1995) 193-206
[18] Genero, M., Manso, M.E., Piattini, M., Garcia, F.J.: Assessing the Quality and the Complexity of OMT Models. In: Proc. 2nd European Software Measurement Conf. Amsterdam (1999) 99-109
[19] Coad, P., Yourdon, E.: Object-Oriented Analysis. Prentice-Hall (1990)
[20] Rumbaugh, J., Blaha, M., Premerlani, W., Eddy, F., Lorensen, W.: Object Oriented Modeling and Design. Prentice-Hall (1991)
[21] Chidamber, S.R., Kemerer, C.F.: A Metrics Suite for Object Oriented Design. IEEE Trans. Software Eng. 20 (1994) 476-493
[22] Brito e Abreu, F., Carapuça, R.: Object-Oriented Software Engineering: Measuring and Controlling the Development Process. In: Proc. 4th Int'l Conf. Software Quality (ICSQ'94). McLean, VA (1994)
[23] Briand, L.C., Daly, J.W., Wüst, J.K.: A Unified Framework for Cohesion Measurement in Object-Oriented Systems. Empirical Software Eng., An Int'l J. 3 (1998) 65-117
[24] Briand, L.C., Daly, J.W., Wüst, J.K.: A Unified Framework for Coupling Measurement in Object-Oriented Systems. IEEE Trans. Software Eng. 25 (1999) 91-121
[25] Snoeck, M.: On a process algebra approach for the construction and analysis of M.E.R.O.DE.-based conceptual models. Ph.D. dissertation. Katholieke Universiteit Leuven (1995)

[26] Snoeck, M., Dedene, G.: Existence Dependency: The Key to Semantic Integrity Between Structural and Behavioural Aspects of Object Types. IEEE Trans. Software Eng. 24 (1998) 233-251
[27] Booch, G., Rumbaugh, J., Jacobson, I.: The Unified Modeling Language User Guide. Addison-Wesley (1999)
[28] Poels, G., Dedene, G.: Measures for Object-Event Interactions. In: Proc. 33rd Int'l Conf. Technology of Object-Oriented Languages (TOOLS-33). Mont St. Michel, France (2000) 70-81
[29] Brito e Abreu, F., Esteves, R., Goulao, M.: The Design of Eiffel Programs: Quantitative Evaluation Using the MOOD Metrics. In: Proc. 20th Int'l Conf. Technology of Object-Oriented Languages (TOOLS-20). Santa Barbara, Calif. (1996)
[30] Poels, G.: On the use of a Segmentally Additive Proximity Structure to Measure Object Class Life Cycle Complexity. In: Dumke, R., Abran, A.: Software Measurement: Current Trends in Research and Practice. Deutscher Universitäts Verlag, Wiesbaden, Germany (1999) 61-79
[31] Kitchenham, B., Pfleeger, S.L., Fenton, N.: Towards a Framework for Software Measurement Validation. IEEE Trans. Software Eng. 21 (1995) 929-944
[32] Zuse, H.: A Framework for Software Measurement. Walter de Gruyter, Berlin (1998)
[33] Roberts, F.S.: Measurement Theory with Applications to Decisionmaking, Utility and the Social Sciences. Addison-Wesley (1979)
[34] Suppes, P., Krantz, D.M., Luce, R.D., Tversky, A.: Foundations of Measurement: Geometrical, Threshold, and Probabilistic Representations. Academic Press, San Diego, Calif. (1989)
[35] Poels, G., Dedene, G.: Distance-based software measurement: necessary and sufficient properties for software measures. Information and Software Technology 42 (2000) 35-46
[36] Poels, G., Viaene, S., Dedene, G.: Distance Measures for Information System Reengineering. In: Proc. 12th Int'l Conf. Advanced Information Systems Eng. (CAiSE*00), Stockholm (2000) 387-400
[37] Briand, L., Arisholm, E., Counsell, S., Houdek, F., Thévenod-Fosse, P.: Empirical Studies of Object-Oriented Artifacts, Methods, and Processes: State of The Art and Future Directions. Tech. Rep. IESE 037.99/E, Fraunhofer IESE (1999)
[38] Benlarbi, S., El Emam, K., Goel, N.: Issues in Validating Object-Oriented Metrics for Early Risk Prediction. In: Proc. 10th Int'l Symposium Software Reliability Eng. (ISSRE'99). Boca Raton, Florida (1999)
[39] Basili, V.R., Briand, L., Melo, W.L.: A Validation of Object-Oriented Design Metrics as Quality Indicators. IEEE Trans. Software Eng. 22 (1996) 751-761
[40] Briand, L., Daly, J.W., Porter, V., Wüst, J.: A Comprehensive Empirical Validation of Product Measures for Object-Oriented Systems. Tech. Rep. ISERN-98-07, Fraunhofer IESE (1998)
[41] Briand, L.C., Morasca, S., Basili, V.R.: Defining and Validating Measures for Object-Based High-Level Design. IEEE Trans. Software Eng. 25 (1999) 722-743
[42] Brito e Abreu, F., Melo, W.: Evaluating the Impact of Object-Oriented Design on Quality. In: Proc. 3rd Int'l Software Metrics Symposium (METRICS'96). Berlin (1996)
[43] Benlarbi, S., Melo, W.L.: Polymorphism Measures for Early Risk Prediction. In: Proc. 21st Int'l Conf. Software Eng. (ICSE'99). Los Angeles (1999) 334-344