# Reuse of generic concepts in information modeling

Haim Kilov
Bellcore, MRE-1F 216
435 South Street
Morristown, NJ 07960

haim@bcr.cc.bellcore.com

## 1 Position Statement

Two major issues have existed for some time in system development: (1) the information to be processed should be understood, and (2) the complexity of existing systems precludes understanding. As early as in 1972, E.W.Dijkstra [Dijk 72] explicitly acknowledged this and urged programmers to deal only with intellectually manageable problems. Although important lessons have been learned in programming, all too often other levels of system development, in particular, planning and analysis, still remain close to black magic. This need not be the case: ideas from programming methodology – most importantly, abstraction – can and should be reused at all levels, and not just in coding.

The sequence understand – specify – reuse may provide a reasonable framework for the job to be done. Implementation should start only after a clear, precise, and formal contract specification [Meye 88, ODM 91] exists. The specification should be abstract enough: irrelevant details like implementation considerations should be suppressed, and the declarative approach (i.e., formulating pre- and postconditions for an operation), well-known in programming methodology [Dijk 76], is appropriate (i.e., should be reused) at the analysis level as well. Intellectual economy (reuse rather than reinvention) is possible only if the construct to be reused is understood, i.e., if its specification exists and is precise and explicit. However, this need not mean top-down waterfall development: a higher-level primitive may be built from lower-level ones that already exist.

The most important difference between a typical contract considered from a programming language viewpoint [Meye 88] and a contract considered from an information management viewpoint is the existence of explicit inter-class relationships. Namely, the pre- and postconditions refer not just to properties of their class, but also to properties of other related classes visible to the client of the class. An operation for which the contract is specified may be spanned across several classes [Kilo 91]. An invariant for a class may include visible properties of objects belonging to other classes, and in this case it may be more proper to consider an inter-class invariant. This approach to contracts may be used both at the generic level (example: "create a dependent entity" and at any application domain-specific level (example: "hire an employee").

Certain generic modeling concepts (entities, relationships, dependencies, etc.) have been used by analysts, often inconsistently. These inconsistencies were due to the absence of formal and

implementation-independent definitions of the concepts. As a result, concepts have not been properly understood (a typical remark by a subject matter expert: "we don't know whether this is a dependent entity or a subtype") and therefore not properly (re)used. Moreover, structural properties ("data") and behavioral properties ("processes") have been artificially separated, leading to unnecessary complexities.

We have provided precise and formal definitions of generic information modeling concepts, i.e., defined a reusable class library of entity meta-types (aka object classes), based on pre- and postconditions and invariants [Kilo 91]. Pre- and postconditions define operations that can be applied to instances of these classes, and invariants are operation-independent conditions associated with collections of classes that must be true at all times outside of any operation on those classes [ODM 91]. The invariant, pre- and postconditions usually refer to the properties of more than one of the objects. Whereas in considering isolated objects, it has been possible – to a certain extent – to underestimate the importance of precise and formal specifications of behavior, this is not possible anymore for inter-object relationships. The reason is simple: the relationships must be intellectually manageable. As they are substantially more complex than isolated objects, their understanding is possible only by means of encapsulating their implementations and providing explicit and precise, i.e., formal, specifications of their behavior. This approach leads to a clear understanding of concepts, to conceptual simplicity, and also, as an important side-effect, to non-proliferation of different and often shallow definitions for commonly encountered terms. Therefore these concepts can easily be understood and therefore reused both by the customers of the information model (including subject matter experts) and by its implementors. For instance, the definition of a dependent entity includes an invariant: "the existence of a dependent entity instance implies the existence of an appropriate instance of its parent entity". Evidently, without understanding of the information model it cannot be correctly implemented and used; programmers will have to introduce their own understanding because a program has to be precise (and in this manner a programmer will have to become a modeler, usually without the benefit of reusing the class libraries of information model components [only exceptional programmers can do that; however, they work within a certain application domain, and the problem of redundant and inconsistent data across different application areas can not be solved in this manner]).

Given such concept definitions, information modelers and their customers reuse common concepts independently of methodologies, CASE tools, implementations, etc., both at domain-independent and domain-specific levels. The generic class library described in [Kilo 91] is extensible: a sufficient number of application domains sharing a common concept leads to the inclusion of an appropriate generic concept into this library. Examples of currently existing – and reusable – generic concepts are: "regular entities", "dependents",/ "composites", "reference entities", etc. Concepts currently considered for inclusion into the library are exemplified by "derived entity", "version", etc.

Generic concept definitions are based only on primitive Create-Read-Update-Delete (CRUD) operations. Naturally, the signatures, pre- and postconditions of these operations may refer not only to the entity itself, but also to its associated entities (e.g., to create an instance of a dependent entity, references to its parent entity type and instance are needed). Note that an application domain-specific model consists of interrelated objects that may be considered as subclasses of the generic object classes. In this manner, generic properties of an object belonging to a particular domain-specific class (e.g., "account transaction") should not be reinvented: they are reused from

the definition of its generic object (super)class (e.g., "dependent" with respect to "account").

Our experience with information modeling in Bellcore suggests that the reusable component library of generic meta-types leads to drastically improved understanding of information models. The components of these models become clearly defined and therefore reusable. On the other hand, the granularity of these components is appropriate: the size of the models does not preclude their understanding, especially taking into account that one "high-level" entity meta-type can be decomposed into a cluster of interrelated "lower-level" entity meta-types. (For instance, a "document", being a subtype of a "composite entity", may belong, together with its associations, to the high-level model. Components of a document, e.g., pieces of text, tables, pictures, etc., being subtypes of a "component entity", may be of no interest – and therefore invisible – to the high level model, but will belong, together with their associations, to the lower-level model. In this manner, the high-level model is an abstraction ("suppression of irrelevant detail to establish a simplified model" [ODPQ 91]) of the lower-level one.) Naturally, model clustering provides a way of browsing through the model, again, both by its users and implementors.

# References


[Dijk 72] E.W.Dijkstra. The humble programmer. Communications of the ACM, Vol. 15 (1972), No. 10, pp. 859-886.

[Meye 88] B.Meyer. Object-oriented software construction. Prentice-Hall, 1988.

[Kilo 91] H.Kilov. Generic information modeling concepts: a reusable component library. In: TOOLS '91 (Proceedings of the Fourth International Conference on Technology of Object-Oriented Languages and Systems, Paris, 1991, pp. 187-201). Prentice-Hall, 1991.

[ODM 91] A Reference Model for Object Data Management. Final Revision. (ANSI Accredited Standards Committee. X3, Information Processing Systems.) Document Number OODB 89-01R8. August 10, 1991.

[Dijk 76] E.W.Dijkstra. A discipline of programming. Prentice-Hall, 1976.

[ODPQ 91] Basic Reference Model for Open Distributed Processing Q Part 2: Descriptive Model. Committee Draft ISO/IEC CD 10746-2. ISO/IEC JTC1/SC 21 N 6079. 1991-07-24.


# 2 About the Author

Haim Kilov has been through all stages of various software (compilers, preprocessors, post-relational DBMS, etc.) design and development – from initial conception to actual implementation and release, and also has been engaged in research, development, and consulting in advanced information modeling. He is currently involved in information modeling as a Member of Technical Staff at Bellcore (Morristown, NJ). His approach to creating a reusable library of generic object classes for this purpose has been widely used in actual modeling activities within Bellcore. He is also a member of the ANSI X3 Database System Study Group and its subgroups (Object Database Task Group

and OSI/Database Task Group); he is one of the Editors of the reports of the Object Database Task Group and one of the active contributors to the Object Data Management Reference Model. He is a member of the Editorial Board of "Computer Standards and Interfaces" and has been a Program Committee member of several domestic and international conferences and workshops on information management. He has a significant number of published papers and reviews in the database and information modeling areas. His current interests and experience are in the areas of information modeling and programming methodology.