Model or mould? A challenge for better traceability

S. J. Morris
Department of Computing
City University
London, UK
sjm@soi.city.ac.uk

O. C. Z. Gotel
Department of Computer Science
Pace University
New York, USA
ogotel@pace.edu

Abstract

This paper examines the notion of the model as it may be used in software engineering via the definition of a series of progressively more complex relationships between the modeller, the model and what is modelled. A gradual historical development is identified where the purpose of the model changes from its representation of a pre-existing subject to its action as a precedent and a definition for some subsequent artifact that is its object. The notion of a composite model type, or mould, combining both purposes is made explicit. Its implications for traceability in multi-model processes are being investigated.

1. Introduction

Models have certainly been in use for centuries, if not millennia [12], and very recent history by itself provides a wide variety of examples to which appeals can be made when attempting to explain or justify the models now used in software engineering. An alternative analysis, particularly of models that form part of object oriented development processes, suggests that such artifacts are fundamentally different from their antecedents. They differ in ways that make both apparently simple pedagogical tasks, such as answering the basic question "What is a model?" and more specific technical challenges, such as tracing requirements through a series of models, equally problematic. The objective of this paper is to outline a position regarding model definition that will bridge the ever widening gap between vernacular understanding and technical definitions. To do this it is necessary to define the qualities which belong to the relationships between a model and its modeller and between a model and its subject (or object), to the form of its representation, to the manner of its derivation and to its purpose. In trying to disentangle the host of different and sometimes contradictory definitions already available, we follow the example of contemporary history of ideas and seek to shift attention away from meaning and towards questions about intentionality, agency and usage [19]. A more systematic examination of existing usage, particular within requirements engineering, will benefit from these preliminary investigations and will lead to a well-founded discussion of the problems of tracing between specific models and of improving traceability qualities in general.

2. Primary relationship between model and modeller

Although it may be the case that everything is a model [1], or is to be one in software engineering, it is more generally true that anything can be a model. This is the consequence of models being representations, "representations to ourselves of what we do, of what we want, and what we hope for ... not simply a reflection of some state of affairs, but beyond this, a putative mode of action, a representation of prospective practice, or of acquired modes of action" [20]. Representations are, in all their forms, intentional, objects. "Nothing is a representation except in so far as we construct or construe it as one and it is precisely the representation we make it, or take it to be" [20]. Thus a UML class diagram printed on an A4 sheet of paper may be interpreted as a model, or may be folded to function and be seen as an airplane or represent a fictional animal. For each of these different models to be recognised separate qualities require recognition, including essentially the existence of a specific modeller. The basic modelling relationship becomes specifically and explicitly triadic

rather than diadic in the much looser sense which is common:

1) *M(S, x, y) where S takes x a*s a *model of y* [20].

Identity of modellers may often be self-evident during software development, but subsequent traceability may depend on explicit records of authorship and, in a more developed view of this primary relationship, on contribution structures [7].

The role of the modeller is also fundamental because he or she is the one who actually tackles the problem of complexity in all its aspects, a basic motivation behind model making. The explicit introduction of the modeller, as one who will sort out, from the plethora of the available characteristics of the object, those that are and those that are not to be modelled requires an extension of the definition of the modelling relationship:

2) *M(S, x, y) and R(x) < R(y), where S takes x as a model of y and R is the range or richness of relevant properties* [20].

The abstractive becomes defined as being less profuse in whatever is characteristic and apposite. The author of this definition, Wartofsky, notes negatively that a model richer in properties than its object fails as a model because a 'negative analogy' may be discovered (the term defined by Hesse [9] for properties which may be wholly inappropriate in all senses) and positively that a successful model reveals previously unnoticed properties of its object. The 'richness' test appears firmly grounded but its implications within software engineering are yet to be tested.

3. Concrete or abstract in representation or content

The 'richness' of traditional physical models could be defined exactly by scale. At the beginning of the last century the physicist Ludwig Boltzmann wrote an oft-quoted entry for 'model' in the famous eleventh edition of the Encyclopaedia Britannica. His definition: "A tangible representation, whether the size be equal or greater or smaller, of an object which is either in actual existence, or has to be constructed in fact or in thought" [3] remains satisfactory for a large class of artifacts called 'models'. While he recognised other extensions of the means by which "we comprehend objects in thought and represent them in language or writing", he set aside anything which did not involve "a concrete spatial analogy in three dimensions". Physical models conventionally incorporate a scale, normally expressed in terms of a numerical ratio, which acts as an exact and uncontentious means for defining the likeness between model and object modelled.

Such clarity evaporated with the widespread use of the 'model' in a scientific context as a "selective or abstractive duplication of some aspect of the world ... a construction in which we organize symbols of our experience and thought in such a way that we effect a systematic representation of this experience, or thought, as a means of understanding it or explaining it to others." [20] Such models are abstract, as opposed to concrete, because their forms of representation lack most spatiotemporal properties and are inaccessible to most sense perceptions. This issue of basic representational form or substance has importance in 'upstream' software engineering activities, particularly requirements engineering, where documents may employ a variety of physical and abstract media [8].

In other engineering disciplines there has been a clear transfer from concrete to abstract models, good examples coming from mechanical and structural engineering. The design and testing of structures such as electricity pylons [13] and dams was until recently dependant upon physical models, themselves sophisticated artifacts employing the special components and materials needed to simulate both size and behaviour to scale [10]. Replacement of these physical models by abstract 'finite element models' with computable behaviours is well documented [22], as are techniques for automated production from such models, for example via laser stereolithography.

The term abstract also has connotations of disassociation from fixed forms or arbitrariness in expression, so it is appropriate here to mention conceptual models. The notions of a 'theory', a set of axioms, or some well established or universally conceded principles are all what might be called general concepts. The term 'conceptual model' has now come to apply generally to any set of general concepts derived from analysis of a particular domain and intended for continued use in the same or an analogous domain. In this sense they are 'consequent models' that derive from their antecedent subjects and denote their domains. The obvious paradigmatic example is the entity-relationship model of data which has a very simple set of concepts in its original form. It adopts "the more natural view that the real world consists of entities and relationships" [4]. Similarly the object-oriented model of software systems developed from concepts of 'class' and 'object'. Emphasis in both cases has to be on the definite article

in order to maintain the denotational character of the definition. Conceptual models have come to play an important role in requirements engineering and will be studied further.

Models which are abstract in the vernacular sense of intangible or lacking any three dimensional form may also be abstract in a sense referring to their derivation. According to the terms and definitions in UML 2.0, a model element is "an element that is an abstraction drawn from the system being modeled" [16]. The process of abstraction may itself be alternatively extractive or generalising, its product being for example 'leg' rather than 'limb', or have some more specialised meanings particularly in an object-oriented environment. Models called abstract in this extractive or generalising sense are also important because they comprise an important class depending upon a form of analogy in their derivation.

4. Derivation via analogy

Analogical models all resemble their subjects in some aspect of their appearance, structure or relationships. They normally belong at a level of abstraction, in the same non-concrete sense of being withdrawn or separated from matter, between scale models which are iconic in the Piercian sense [18] and bear a close resemblance to some material subject, and mathematical models in which all components and relationships are mathematical entities represented in a mathematical language. It is common in software engineering to appeal to analogy as the basis for model derivation while at the same time characterising such models as 'analytic' on the basis of their purpose. Two activities may be involved, the derivation of likeness by abstracting or generalising particular characteristics (what one might call 'abstracted analogy') or alternatively the identification of analogy referring to some wholly independent entity or previously unrelated phenomenon, often as a result of what may appear to be an arbitrary choice (what one might call 'homomorphic analogy'). Abstracted analogy has been the basis for much modelling in software engineering whereas homomorphic analogy has been the basis of many scientific models, an often cited example being billiard balls in random motion as a model of gas.

Although analogy is often claimed as the basis for derivation of models in software engineering, its limitations go unnoticed. Analogy is not transitive (If a is analagous to b and b is analagous to c, then a is not analagous to c) [9] with implications for any chain of models based on an analogical relationship with what is being modelled. There is also no consistent level of 'metaphysical commitment' [20] or belief in the applicability of particular analogies, which may be as different as an 'ad hoc' similarity and an assertion of archetypal relationship based on first principles [2]. In addition homomorphic analogy is subject to the strong variability of cultural association and understanding.

5. Models as interpretations

Given its disassociation from any purely analytical approaches, it is not surprising that the important alternative definition of model from logic has been rejected as unsuitable for software engineering and particularly for requirements engineering [11]. The duality of logicians that associates 'model' with 'theory' is significant because the modelling relationship which it embodies has widespread, if sometimes vague, use. Inverting the ordinary view of models as abstractive representations, logicians speak of 'embodiments' or 'interpretations' of some axiomatized formal system, designated a 'theory', where there is an isomorphic relationship between its structure and that of its interpretation, designated a 'model'. The usefulness of this model, in the context of scientific enquiry, lies in the fact that "postulates of theory make existential claims, but the model serves merely to channel these to some confrontation with experimentally testable consequences" [20].

In the world of software engineering this definition of a model, embodying rather than abstracting, is that taken, at least in part, by all object oriented approaches, which take as their premises the existence of classes and objects and other principles. Likewise definition of the UML may be viewed as a complex set of axiomatic statements to be used as the basis for the interpretation of any particular domain and thus the creation of a model of it. Any appeal to an analogical basis relationship between a UML model and its subject (or object) is further undermined by definition of a model as 'an instance of a meta-model' [15] concomitant upon the model conforming to the principles set out in its meta-language. A full analysis of multi-layered approaches to programming going back to Dykstra [5] is outside the scope of current work, but it does require some discussion.

6. Relationships of denotation or exemplification

In order to clarify the effect of integrating models into a layered hierarchy, such as that to which the UML belongs, it is necessary to distinguish between denotation and exemplification. One basic type of model denotes, or has as an instance, what it models, for example the 'ship model' traditionally constructed in the greatest detail to show the finished vessel, or the 'mathematical model' expressed as a formula in a mathematical language that applies to a state or object modelled. In the same manner an 'architectural model' denotes a building or buildings. Its purpose is essentially communicative, to provide a preliminary view of what is to be built once a detailed design has reached or is nearing completion. After actual construction the model provides a record of the design, perhaps not as finally realised, but during building it has no primary function in determining what actually rises from the ground. The model derives directly from architectural drawings, which themselves follow conventions sufficiently strict [6] for them to act in the same capacity as a musical 'score' and allowing multiple 'performances' of the same 'work'. The drawings act as the essential antecedent to both building and model, although only the drawings also act as the essential precedent that dictates the consequent form and function of what is subsequently constructed.

(3) D(x,y) Model x denotes y or has an instance y.

Models in software engineering serve, or attempt to serve, the same denotational function as architectural drawings. Some however serve the function of exemplification. In the alternative basic type the roles are reversed; the model is an example, or an instance, of what it models [6], for example a 'fashion model' exemplifies the physical characteristics that the clothes designer hopes all wearers will have, or a 'compliant universe' is the 'model' of a set of axioms.

(4) E(x,v) Model x is an instance v; x is an exemplar of v and x complies with Rr(v) where Rr(v) are the rules embodying v.

There is a common, but not binding, association between models that denote and models that are analytic and also between models that exemplify and models that are telic.

7. Analytic or telic purpose

Although all models may be analytic in some sense related to purpose or derivation, and all models are created with some end use in view, a distinction between specifically analytic and telic purposes is significant. The role of analytic models, those made as an aid to practical or theoretical understanding of a particular domain or artifact, have been widely studied and they have played a fundamental role in upstream software engineering processes. The role of telic models, those created specifically with an expectation of action and with some particular end in view, has been little studied because alternative and long developed professional conventions have been the means of representation.

Most 'concrete' engineering models have been quasi-telic because of their subsidiary but specific role in testing. What are the characteristics of the airflow over a wing? At what wind speed will an electricity pylon fall? What strength of earthquake will a dam withstand? These are questions all once answered by physical models and now replaced by representation as flow algorithms or matrices of forces in 'finite element models'. Architectural models (other than ones that are essentially initial 'sketches') are likewise a by-product of the design process, which play an important communicative role, rather than an essential intermediate product in building design and construction.

Telic models which only exemplify (or 'are instances of') open the way for models that are fabrications using solely predefined concepts. They form the basis for multilayer language definitions and the foundation for the idea of a 'model' is 'an instance of a meta-model' and consequently for model driven architectures. The MDA specification advocates, at least in its glossary, the most extreme version of a telic definition of a model as something built with a specific purpose or end in view : "a formal specification of the function, structure and/or behavior of an application or system" [14]. Without any of the important connotations of exemplification the telic model is the prescriptive as opposed to descriptive model [21], or 'TO-BE' rather than 'AS-IS' model [17], these pairings both being examples from studies outside the object-oriented context.

8. Consequent of antecedent or precedent for consequent

There is also a common, but not binding, association between analytic models and *consequent models* of some *antecedent subject* and also between telic models and *precedent models* for some *consequent object* which comes subsequently.

Extending the formulae above and using *cM* for consequent model and *pM* for precedent model:

(5) cM(S, x, y) and D(x,y) and Rp(x) < Rp(y)
where:
- Rp is the range or richness of relevant properties
- S takes x as an 'consequent model of' y
e.g. x/y: sketched data flow diagram / observed office activity, or architectural model / the Tower of London.

(6) pM(S, x, y) and D(x,y) and Rp(x) < Rp(y)
where:
- Rp is the range or richness of relevant properties
- S takes x as an 'precedent model of' y
e.g. x/v: sculptor's maquette / a sculpture, or 'keep fit' video / someone's exercise routine.

Precedent models need not in general comply with any standard set of rules or axioms so compliance with *Rr(v)* is not demanded.

9. Model, cast or mould

With the advent of multi-model processes of development it is important to distinguish clearly between the traditional model, one which follows some antecedent and pre-existing artifact, design or domain (its subject), and its partner, the more recent model which itself defines and acts as the predecessor to and precedent for some consequent artifact (its object). To reuse a term from physical processes, this second type acts as a mould (Am. mold), any shape containing an internal void which, when filled, forms a 'cast'.

This distinction between these two types was implicit in the earlier definitions which separated (as mentioned above) descriptive and prescriptive models. However multi-layered model hierarchies require that the two types should be newly made explicit in a composite form. Recalling the physical 'cast', this can itself, in its own turn, act as the 'model' for another 'mould', as it does in 'lost wax', the traditional multi-stage process for metal casting.

Any form of software engineering model with both object and subject requires an explicit definition of a complete and composite 'mould' (mM) relationship. Much confusion may arise because such a 'mould' in its making is an instance, but in its use has an instance. This is particularly relevant for notations that are conventionally employed both for consequent and precedent models. Explicit transformations between representations for the consequent and the precedent remain a fundamental issue in requirements engineering. Unless the model relationships are explicit the essential transformations will remain uncertain.

(7) mM(S, v, x, y)
and D(x,y) and E(x,v) and Rp(x) < Rp(y) and x complies with Rr(v)
where:
- Rp is the range or richness of relevant properties
- Rr(v) are the rules embodying v
which may be wholly or partially defined
- S takes x either as a 'model of' both v and of y
or as a 'mould of' v (or 'from v) and y (of 'for y')
e.g. v/x/y: UML syntax and semantics / a UML class diagram / segment of object code.

The problem of switching from consequent to precedent model, from model to mould is implicit in all definitions which attempt to embrace both a system and the domain of which it will form part, for example the alternative definition of model provided in the body of the text of the OMG's MDA guide: "a description or specification of that system and its environment for some certain purpose. A model is often presented as a combination of drawings and text. The text may be in a modeling language or in a natural language" [14]. The particular point at which the switch takes place and the transformations involved will be a crucial traceability determinant.

10. Conclusions

The sense in which abstract concepts or axioms dictate the form and content of models marked a major shift away from traditional and vernacular ideas of a physical model. Our investigations suggest that recent approaches to modelling in software engineering have been characterized by a further extension of basic concepts in three ways: a reorientation towards the telic and intention to construct, a contraction of the possible domain for analogy, and a merger of previously separate analogic and metamathematical approaches via the conventionalised form of abstraction used in metamodels.

We conclude that a new type of composite model, or mould, requires recognition, both to assist understanding and to facilitate the tracing of elements through successions of models. Examination of the explicit relationships between models, modellers and what is modelled will be the basis for further work in the context of current requirements engineering research and practice. We hope that this will lead to a

proper understanding of how traceability properties may be identified within, or integrated into, multi-model processes.

11. Acknowledgements

The authors are grateful to the reviewers for their helpful comments.

12. References

[1] Bézivin, J. On the unification power of models. *Software and System Modeling.* Vol.4, 2005, p.171.

[2] Black, M. *Models and Metaphors. Studies in Language and Philosophy*. Cornell: Cornell UP, 1962, p.24.

[3] Boltzmann, L. Model. *Encyclopaedia Britannica* (11th Ed.). Cambridge: CUP, 1910-11, pp.638-640.

[4] Chen, P.P.-S. The entity relationship model - Toward a unified view of data. *ACM Transactions on Database Systems*, Vol.1, No.1, March 1976, p.9.

[5] Dahl, O.-J., Dykstra, E.W. and Hoare, C.A.R. *Structured Programming*. A.P.I.C. Studies in Data Processing No.8. London: Academic Press, 1972.

[6] Goodman, N. *Languages of Art. An Approach to a Theory of Symbols.* Indianapolis: Hackett, 1976, pp.218-219 and p.171.

[7] Gotel, O.C.Z. and Finkelstein A.C.W. Contribution Structures, In *Proceedings of the Second IEEE International Symposium on Requirements Engineering (RE'95).* IEEE Computer Society Press, York, UK, March 27-29, 1995, pp.100-107.

[8] Gotel, O.C.Z. and Morris, S.J. Crafting the Requirements Record with the Informed Use of Media. In *Proceedings of the First International Workshop on Multimedia Requirements Engineering (MeRE'06).* IEEE, September 2006.

[9] Hesse, M.B. *Models and Analogues in Science.* Notre Dame, IN: Univ. of Notre Dame Press, 1966, p.8 and p.67.

[10] Hossdorf, H. *Model Analysis of Structures*. New York: Von Nostrand Reihold, 1974.

[11] Jackson, M. *Software Requirements and Specification: A Lexicon of Practice, Principles and Prejudices*. Wokingham: Wiley and ACM, 1995, p.121.

[12] Jones, D. *Model boats from the tomb of Tut'ankhamun*. Oxford: Griffith Institute, 1990.

[13] Martinez, D.R. and Miller, A.K. (Eds) *Combined Experimental Analytical Modeling of Dynamic Structural Systems*, American Society of Mechanical Engineers, 1985, pp.50-51.

[14] OMG, *MDA Guide* Version 1.0.1, June 2003, p.A-2 and p.2-2.

[15] OMG *Unified Modeling Language* v1.5. OMG, 2003 p.2-5.

[16] OMG *Unified Modeling Language (UML) Specification: Infrastructure Version 2.0.* December 2003, p.11.

[17] Patel, N.V. Healthcare modelling through role activity diagrams for process-based information systems development. *Requirements Engineering*, Vol.5, No.2, 2000, p.85.

[18] Pierce, C.S. *Collected Papers of Charles Sanders Pierce*, Vol.II. Cambridge,MA: Harvard University Press, 1934.

[19] Skinner, Q. *Visions of Politics. Volume 1: Regarding Method*. Cambridge, CUP, 2002, p.2.

[20] Wartofsky, M.W. *Models, Representations and the Scientific Understanding.* Boston Studies in the Philosophy of Science Vol XLVIII. Dordrecht: Reidel, 1979, p.xv, p.xxi, p.6, p.8, p.xv, p.28 and p.31.

[21] Wieringa, R.J. *Requirements Engineering Frameworks for Understanding*. New York: Wiley, 1996, p.75.

[22] Zienkiewicz, O.C., Taylor, R.L. and Zhu, J.Z. *The Finite Element Method: Its Basis and Fundamentals.* 6th ed. Amsterdam; London: Elsevier Butterworth-Heinemann, 2005.