PrimaVera Working Paper Series

UNIVERSITEIT VAN AMSTERDAM

PrimaVera Working Paper 2004-16

Metapattern as context orientation

meeting Odell's challenge of object orientation

Pieter Wisse

October 2004

Category: scientific

Universiteit van Amsterdam
Department of Business Studies
Roetersstraat 11
1018 WB Amsterdam
http://primavera.fee.uva.nl

Metapattern as context orientation

meeting Odell's challenge of object orientation

Pieter Wisse

Abstract

As object orientation has matured, its limitations also became more evident and accepted. J.J. Odell highlights such issues in a recent collection of essays. This paper takes up his challenge. Metapattern is an approach to information modeling especially designed for moving beyond traditional OO. What Odell demonstrates as remaining inherently problematic with OO, is given elegant solutions with metapattern.

Keywords

Metapattern, conceptual information modeling, ontological engineering, situational variety, multiple contexts, temporal variety, information dynamics, J.J. Odell.

author

Dr. ir. Pieter Wisse (*pieter@wisse.cc*; *www.wisse.cc*) is the founder and president of Information Dynamics, an independent company operating from the Netherlands and involved in research & development of infrastructure for complex information systems and services. He is also affiliated with the PrimaVera research program in information management of Amsterdam University.

INDEX

Introduction

Throughout his work, J.J. Odell emphasizes conceptual modeling. For *Advanced Object-Oriented Analysis & Design Using UML* (1998), Odell selected twenty-two essays which he previously published separately. Because he addresses advanced topics in object orientation by presenting theoretical guidelines, comments given here[1] on his treatment are also largely theoretical and conceptual. Therefore, the more general purpose with this paper is a fundamental comparison of metapattern (Wisse, 2001) to object orientation for which Odell provides the challenge. Such cues are taken mostly in the order as the essays appear in his collection.

Important points in Odell's thoughtful essays remain of course valid for context-oriented information modeling, too. The selection from his work is therefore admittedly one-sided. A deliberate emphasis is placed on issues provoking discussion. Of particular interest are (1) problems which help illuminate how metapattern differs from traditional object orientation and, (2) how it enables a modeler to accomplish better solutions.

1. Conceptual model and implementation considerations

In his essay 'Modeling Objects: Using Binary- and Entity-Relationship Approaches,' Odell's main argument concerns the decision to model information as an object class. He maintains that only what has been conceptually specified as an object type leads to implementation as an object class. He notes that "the class acts as an index for both structure and operation in the OO world." Clearly, his objection to traditional entity-attribute-relationship (EAR) models is that EAR attribute types are unclear as to whether or not an object type has been modeled. He suggests that, when applicable, an attribute type be enhanced to include an explicit description of an object type. Indeed, this is an adequate solution to the problem he has identified.

Odell does not pursue another issue: with binary-relationship (BR) models, anything indicated on each side of a relationship may be considered an object type. This could lead to over-specification (instead of the possibility of under-specification with EAR). The solution is the inverse of what Odell suggested for attribute types in EAR; a BR object, when applicable, is to be disabled as an OO object.

In his essay, day, location and color are the contested information objects. Must a corresponding type, and thereby class, be attributed to, say, day? In this context, Odell states somewhat remarkably that "implementation is not the issue in OO analysis [for t]he primary goal of analysis is to model the end-user's concepts and leave the concept implementation to the designers." This contrasts with the problem he raises about whether or not to specify a class for information of a particular type. He writes: "To properly assist the OO designer in making

correct implementation decisions, the OO analyst must clearly specify *all* object types." This is the balanced view. As the conceptual model controls many decisions about implementation, it also contains transformation concepts. That a type may become a class is just an example. The conceptual modeler must handle such transformation concepts with awareness and care.

Metapattern makes a distinction between primitive and pointer information objects (Wisse, 2001, 2004). A primitive information object corresponds to a normal entity attribute. Then, particular points in time are presented as primitive information objects. Apart from its context, this means that every point in time is completely independent from others.

As an alternative, the relationship for starting time could have been conceived as a pointer information object, making that object part of a time management system beyond its own immediate context. The information set may contain several identical pointer objects which effectively and even *systematically* pertain to the same time point.

The cohesive mechanism offered by pointers has a price. By definition a pointer never provides direct access to information; one or more relationships must be navigated to arrive at the information "itself." How this functions is shown in figure 1, taking elements from the case Odell explains in his essay. Alternative b. stresses that somewhere in the information set a particular date must be "primitively" available.[2]

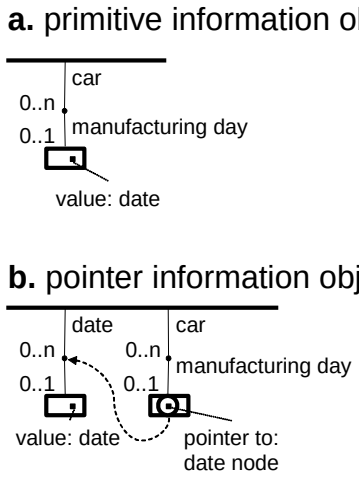

Figure 1: Choice between primitive and reference.

Actually, these alternatives do not reflect "the end-user's concepts," at all—certainly not at first. Users do not perceive how conceptual specifications influence implementation. Therefore, the conceptual modeler—whom Odell calls an analyst—must inform future users and enhance their decision-making quality by presenting alternatives. Playing the role of change agent, the

modeler helps users gain trust in what is essentially a *design process* toward an information model. That is, a model, when created, is designed, not analyzed.

But users will often lack the capacity for positive verification, explaining why modeling for complex information requirements is a separate profession. A user must have sufficient ground for trust in the professional. And the professional must be seen to make complex decisions about information *on behalf of* the user.

Figure 1 shows the cardinality of information objects beside their nodes. In some cases, the minimal value of zero for the number of instances might appear unorthodox. But information really could be unavailable.

On the other hand, in the course of time and/or through corrections, a multitude of information objects may exist. As time is universally managed in metapattern at the lowest level of aggregation, the additional cardinality of existence and validity entries is nowhere stated explicitly.

2. Navigational guidelines

Figure 2 presents an information model in terms of metapattern of Odell's complete case in his essay 'Modeling Objects: Using Binary- and Entity-Relationship Approaches.'

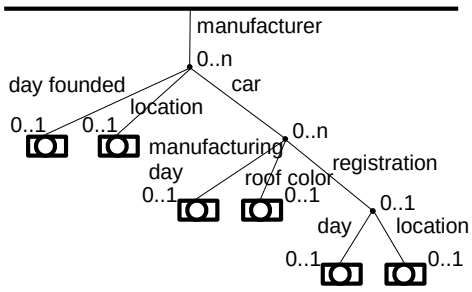

Figure 2: A manufacturer builds cars which are registered.

This is, however, only one of many alternatives. To maintain overview, details such as which information objects pointers are directed have been left out.

Compared to Odell's original model, metapattern's interpretation already provides more opportunities. Location, for example, is explicitly modeled in two different contexts, allowing for corresponding differentiation between intexts. At the location level such flexibility is impossible with BR or EAR models; derived entity types would have to be introduced.

Especially illuminating: date appears in many different contexts.

Based on the concept of context, and in particular on the nil object, the (mainly) conceptual model can already take navigation into account. Of course, an OO class is already fitted for navigation, with mechanisms to control access to its instances. Metapattern invites development of conceptually-equivalent models; they are simultaneously highly-differentiated with respect to implementation as concrete information sets. As such, metapattern is also a tool for bridging the gap between conceptual and implementation models. As stated earlier, some important implementation issues can already be "prepared" in the conceptual model through the use of transformation concepts.

Metapattern tries to incorporate a positive legacy of hierarchical and network data modeling. On the negative side, those traditional modeling approaches are almost completely biased toward implementation. Though implementation issues may indeed be prepared, the focus remains firmly conceptual.

Figure 3 contains an alternative to the model sketched in figure 2. Car, not manufacturer, is taken as a starting point.

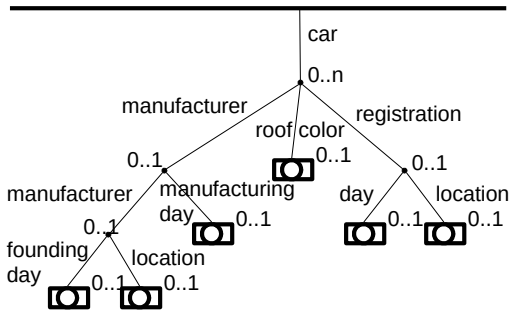

Figure 3: A car is manufactured and registered.

It is—why not?—also feasible to take day as a starting point for modeling (figure 4).

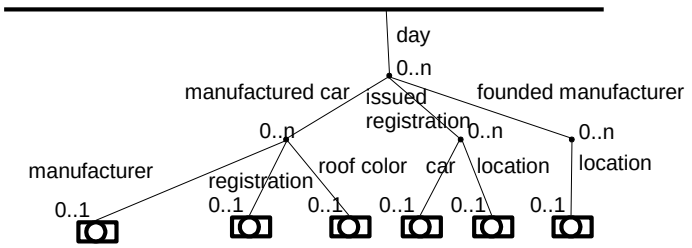

Figure 4: A day with different events.

This last alternative is particularly artificial. This should come as no surprise, as day is one of the attribute types Odell uses to support the case for extending EAR models. Following Odell, it seems logical to develop the model from three points of view, not just one. So, in figure 5, the "normal" entities—entity types, actually—of manufacturer, car and (car) registration are all given a minimal context.

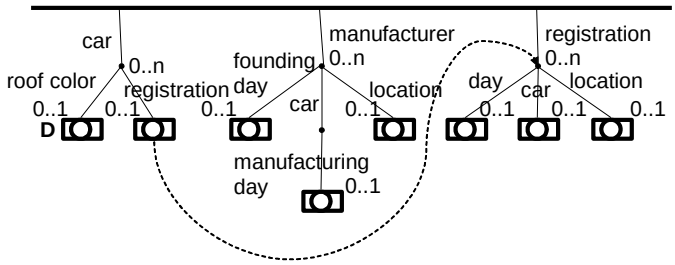

Figure 5: From a single, implicit context to multiple contexts.

As needs arise, information objects in such a figure may be connected with dotted lines. They help to make visible which navigation is supported by pointer information objects. In Figure 5, one such line has been added for "jumping" from the registration-of-car to the (car) registration itself.

Figure 5 reflects only one of many alternatives. Which information model is best can be judged, among other criteria, against the advantages or disadvantages of context designs (traditionally: types and classes, respectively) and navigational options. An EAR or BR model is unfamiliar with the concept of context, at least as a metaconcept, so cannot provide insight. In particular, something like the nil object is completely absent. EAR and BR, therefore, lack even the base from which to integrate suggestions for navigation into a conceptual model.

It is possible to make more elaborate preparations for navigation at the stage of conceptual modeling. When an information object is indicated by a capital letter D, as in Figure 5 (the roof color of a car**)**, access should be as direct as possible. The mechanism for doing so—through an index, for example—is and should remain a matter for implementation.

3. Type-on-context

What should be apparent with EAR and BR models is their redundancy. A relationship's name is often identical with an involved entity's name. This occurs with relationship in all directions.

An information model designed with metapattern is less troubled by such redundancy because intermediary and pointer information objects are not complete objects. Rather, they perform the

service of establishing a partial identity for what is constituted as an overall object. It is particularly difficult to come up with additional names for all partial identities; the preference is thus to leave out names for nodes altogether.

Not assigning names to nodes makes the fundamental nature of metapattern's differences stand out even more clearly, since relationships provide the most important elements in typing. Although specific information objects may contribute to establishment of a type, relationships usually suffice. EAR/BR modeling is radically different; the concept of entity has priority, leading to type-on-entity, and the axiom of traditional object orientation is type-on-object.

In metapattern, priority is squarely given to context, not object. In general, the type is "on" the (whole) context. In practice (usually), only type-on-relationship is actively used from this wide range.

But the possibility does exist for a concrete information object to participate in determining the type of an (other) information object. When used, this typing mode must be expressed in the model, explaining why instances and types are sometimes mixed in the same model. Rather than being a shortcoming of metapattern, it exemplifies its powerful variety.

Another interesting metapattern feature is that no inverse relationships—which traditionally point out the need for a navigational path—appear. They are indicated explicitly, with complete precision and in a positive way. This happens where (partial) identities of an overall object are specified in various contexts. The (car) registration in figure 5 is a good example, as it exists more or less in its own right (on the figure's right) and as a property of a particular car (on the figure's left).

4. Context-oriented, multi-layered typing

In 'Object Types as Objects and Vice Versa,' Odell discusses some fundamental aspects of object-orientation. He uses as an example a case concerning audio equipment. To a sales person, product types are relevant. A sale to a customer concerns, first, any turntable of type X or any compact disk player of type Y. Whichever instance of one product type or another changes hands is not yet important.

However, in the world view of the inventory clerk, individual machines figure prominently. Instances to the sales person are only types to the clerk, who must know that he is dealing with the instance x of type X or the instance y of type Y.

Odell illustrates that object-oriented tools often lack important qualities. A major limitation: the number of modeling levels is a priori fixed. Another is that the metamodel—Odell's term—can almost never be adapted. The only available possibility is defining different types which can be

instantiated (i.e., instances may be derived from them). But due to lack of flexibility problems may emerge on two sides, as Odell indicates. First, an instance of some type can be a type itself. Second, the "fixed" metamodel may block further aggregation and abstraction.

To overcome such obstacles, Odell favors a universal modeling framework, one with no a priori, strictly independent levels. In summarizing the advantages he writes: "An approach of this kind can both describe different models and provide a common framework for expressing and comparing models."

Metapattern supports Odell's important ideas, even allowing elaboration of his problem statement. He starts by distinguishing between two appearances of a particular object: one constitutes an instance, the other a type giving rise to instances.

From a context-oriented viewpoint, something like sales and inventory contexts seem to exist. An important question might be: "Is the type of audio equipment relevant to the inventory clerk who deals with specific machines?" Or, perhaps, "Is the clerk adequately informed when the machines are uniquely identified?"

While it will always be necessary to await the answer for a particular context, machine types are fundamentally different enough to be of interest. The conclusion is that (product) type should be part of the inventory context. How that context is exactly modeled—by specifying relationships and/or nodes, for example—is irrelevant at this stage. The issue here is that machine type (product type) determines the behavior of its machine instances.

But this does not make machine type the sole determinant of the type of the machine instances. Within the complete context, several relationships and/or information objects may exist, all contributing to the type for underlying node instances. Perhaps audio equipment is only one of many product categories in the whole information set. Such an additional differentiation of context annex type is shown in figure 6.

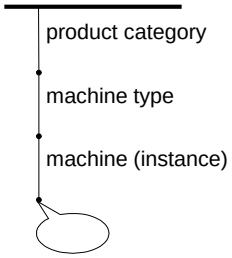

Figure 6: Necessary and sufficient detail in typing.

The above suggestion opens the possibility of entering an expression about specialization versus generalization into the relationship between an information object and its context. It agrees with Odell's intention of the (more) general information object constituting a type for the (more) specific object. But an information object may be like a part, with the context acting as its whole.

Along these lines, Odell's case can be generalized, beginning with product elements. From them the concept of product as a homogeneous classification hierarchy is formed, with each node representing a separate product. All nodes likely share the same type, leaving out generalization/specialization. But products in the context of another product could be considered its parts, and when different behavioral meanings are required in sales and inventory contexts, corresponding partial identities or contexts should be modeled (see figure 7).

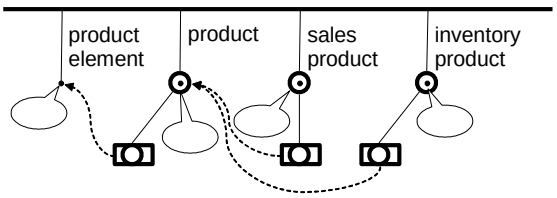

Figure 7: Differentiation of product behaviors.

The route following generalization/specialization in the context of the information object circumvents Odell's problem. Why? Because now hierarchical types are given a correspondingly hierarchical place on what he calls the data & process level (figure 6). Another route eliminates a hierarchy of types altogether, thereby making the whole problem disappear (figure 7).

Typing is, generally speaking, a matter of hierarchical classification, combining strategies of generalization/specialization and whole/part. A particular combination, considering relevant hierarchical levels, amounts to a particular type. Odell does not start from such a general view. His framework appears to be a special case of two levels; the relationship between instances at those levels is determined by generalization/ specialization. However, it is almost impossible to recognize the general view from such a special case. The other way around is straightforward.

Odell raises two problems. The first is the double appearance of an object, i.e., as an instance and as a type of (other) instances. His second problem concerns the opposite end of the range—not differentiation but aggregation of types. His fully-justified objection against limiting the number of metalevels is practically removed by metapattern because the expression of meta-information (or type) equally follows metapattern. Meta-information may also contain a

reference to a particular type. In this manner, types may be connected along a chain of increasing abstraction.

In ‘Object Types as Objects and Vice Versa,’ Odell suggests that the end of such a chain is controlled by modeling approach types. Again, this constraint is not shared by metapattern, because types do not own a priori status, with instances being the consequence. Rather, the inverse applies. Primary status is accorded to context, with instances-in-context coming second. Types, including the number of abstraction levels used for their determination, are by all accounts tertiary. This makes metapattern’s concepts relating to types constitute a different order. They are not, as with traditional object orientation, of the highest order within the information set.

5. From power type to type-on-context

The problem Odell presents in ‘Power Types’ concerns multiple levels in the types of properties. His case deals with trees; with tree species like sugar maple, American elm and apricot; with properties of tree species; and with properties of individual trees.

An individual tree—an instance—has a particular location in space. All trees have that kind of property. Another kind of property trees share is their species—sugar maple, for example. To avoid duplication, the “typical” leaf pattern should be a property of tree species, not of tree. The problem narrows down to the requirement that, apart from a general property such as location, one or more properties of tree instances depend on the tree species involved. Thus, every tree species determines, at least partly, a corresponding tree type.

Odell proceeds by defining all tree species as subtypes of tree. At the same time, each tree species is an instance of the type tree species; in its capacity of such an instance, a tree species has, for example, a particular leaf pattern as a property (value).

The double capacity in which tree species act requires unambiguous control—probably the reason why Odell chose to identify tree species as a power type. His definition of a power type is “an object type whose instances are subtypes of another object type.”

Odell states that “a particularly complex expression of categorization called power types is not addressed by traditional object structure approaches.” Metapattern, however, offers a transparent solution. The differentiation is required for properties of trees. Precision in placing properties succeeds through applying corresponding contexts. As a full context may determine a type, the resulting information model remains compact.

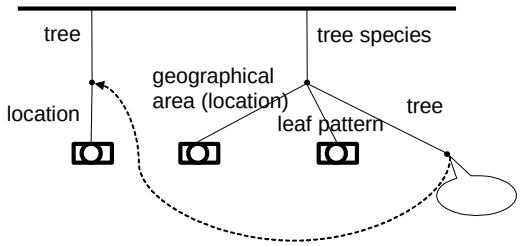

Figure 8: Distributing behavior to appropriate contexts.

The intext of a tree *in the context of* a tree species may be modeled with all its details elsewhere in the information model. Figure 8 shows the relationships of location and geographical area (expressed identically as location). Both lead toward partial identities of the same overall object. What metapattern recognizes as an overall object, working through various contexts, Odell can only use as a single symbol. Metapattern opens the possibility of pointing, from within different contexts, to the same (partial) identity of an overall object. Whenever necessary, from one of its partial identities all its other partial identities may be reached. This is guaranteed by pointers to the same shared nil identity. In addition, for direct navigation (a conceptual direction), an explicit pointer object may be included.

The information model of Odell's case can easily be upgraded by considering tree species as a homogeneous classification hierarchy (instead of it being modeled as a singular information object). Again, concept meanings should be changed accordingly. To construct a hierarchy, its constituting classification elements must first be defined (figure 9).

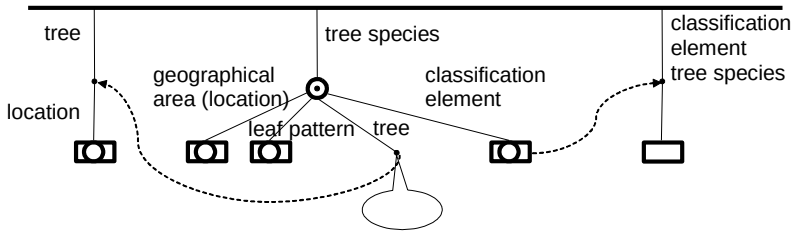

Figure 9: Upgrading possibilities through compositions.

The logical knot, with Odell leading up to a power type, is disentangled by explicitly specifying the upper boundary of the tree classification as constituting the *general* type. This makes all tree instances conform to this type. By including boundary values into the structural model, the model is rendered even more compact. In the figure below, the "general" classification value is implicitly present.

For conceptual modeling, this serves as another example that a preemptive (a priori) boundary between information structure and information values does not exist. At this point it's interesting to discuss why mathematicians choose to call some values "boundary values." What boundary values really do is "fix" applicability of a more general structure. A higher-level structure can contain, as values, information which would otherwise be incorporated into the lower-level structure. Information is more definite in a structural than a value form. As it's easier to change values than structure, a higher-level structure is often an important improvement in flexibility.

Without an appropriate boundary value, tree type classification would not suffice as a structure. Additionally, a typeless type (or, as it's called above, a general type) would be necessary. Essentially, this procedure is identical to inclusion of zero into the number system. Zero is the quintessential example of a boundary value; it is the numberless number.

A conceptual information model should reflect the optimal level of abstraction. Its measure is relative—it is dependent—on actual, predicted information requirements. The professional modeler strives to *invent* more and more abstract concepts. However, when the structure (in regard to the life problem which the tool must solve) can no longer be fixed by assuming one or more boundary values, the abstraction has obviously been pursued too far or in the wrong direction. The modeler must then decrease abstraction or change direction. A conceptual information model without boundary value definitions indicates that concepts at a still-higher level of abstraction may be discovered.

The heuristics of finding an optimal conceptual model/structure are practically limited by the modeling time available, as well as by the knowledge and imagination the modeler brings to the task and can inspire in other stakeholders. Most principals, alas, still don't fully understand that time (i.e., money) spent on conceptual modeling is the single best guarantee of success. A well-orchestrated effort of conceptual modeling serves to involve stakeholders and gain their support.

Finding a balance between structure and values must be taken seriously with every modeling approach. Metapattern softens such design decisions somewhat, because structures, too, may be changed (Wisse, 2001). This should not detract the modeler from introducing abstraction into models but, rather, inspire their continuous improvement.

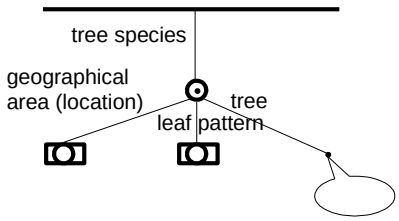

Figure 10: Abstraction through object instance with boundary value.

Take the model shown in figure 10. Due to type-on-context, each tree node placed under the node for tree species corresponding to the value general is, at that point, supplied with the location property. Other nodes in the hierarchy may appear to serve as (partial) identities of the same overall tree object instance. Intext for each tree instance node will also be determined or differentiated by the (other) specific tree species nodes which are part of its context.

Metapattern has no need for power types. The context orientation is already a sufficient mechanism for precision in differentiation and coordination. The only objective of classification is to create *necessary and sufficient* differences. As its context is the complete classification of any information object, its type is immediately and fully defined by that very context.

The elimination of power types must be considered a major advantage; they can easily lead to confusion, as the typing of types is at stake. It takes a highly-disciplined approach to consistently make proper distinctions between one type of type and another.

From a logical point of view, tree species are not subtypes of tree, because tree is associated with instances. One tree species can only be a subtype of another tree species. Where Odell introduces super types, he does not make the double role (or interpretations) of tree apparent. Figure 10 does acknowledge the distinction between types of types. Indeed, a more logical overview is guaranteed, as a tree species is always placed "above" a tree instance. An important consideration in modeling is to keep decomposition as straightforward as possible. Thus the decomposed tree species belongs to the context of a tree instance, not its intext.

Among the cases Odell presents in his essay 'Power Types,' the insurance policy deserves to be mentioned, too. His point is that the properties of a policy should be differentiated according to several perspectives. He again defines as many power types as there are relevant perspectives—some are further subdivided.

Basically, metapattern offers two alternative solutions. The first corresponds to figure 9, in which different perspectives are translated into juxtaposed contexts for different (partial) identities of an overall insurance policy.

The second alternative is sketched in figure 10, in which all perspectives are represented in a single homogeneous classification hierarchy. Based on such a hierarchy, an overall policy is registered with as many nodes annex (partial) identities as are required to accommodate relevant intexts.

6. Structural operations

Metapattern offers many modeling options. It may sound paradoxical, but certain limitations are necessary—by *realistically* limiting behavior, an unambiguous orientation is offered.

In 'Specifying Structural Constraints,' Odell supplies a clear introduction to various limitations relevant to the relationships between a particular object and other objects. He includes reflexive relationships in which an object relates to itself. He also discusses a range of relationship options, from simple conditions for the number of instances (cardinality) to complex rules of governance.

Structurally, metapattern does not depart much from traditional object orientation. It also needs operations to establish and maintain structure in an information set. A type's description must allow for the proper elaboration of necessary and sufficient structural operations.

Only the essay's opening sentence invites a discussion with metapattern (all but that sentence are strongly recommended). It reads: "We know that objects exist in our world, that objects relate to one another, and that these objects and their associations can be changed."

Metapattern has left behind this principle of treating each object as an absolute unit, and that is not a matter of software engineering. The paradigm shift is grounded in the epistemological attitude of the modeler, in how she/he knows the world.

Metapattern replaces the traditional object principle with the context principle, giving context a privileged ontological status. Only within a particular context is an object supposed to exist. As the same object may exist in a multitude of contexts, its appearances are limited to contextual, i.e., partial, identities. Only within those identities and their context-bound intexts do they make *sense* from an information point of view.

Odell seems to express an opinion about the whole world. Metapattern makes no such claims about *all* of reality. Its structural definitions are limited to how people perceive their world, so what is defined "only" refers to a model of information objects and their relationships. All metapattern assumes about objects and how they relate to the world is that it's *as if* the information set is the world's representation—or at least, that it represents a relevant part of the world.

7. Contextual principle

A formal (i.e., strict or mathematical) notation supports precise communication and understanding. That is why Odell, in his essay 'Toward a Formalization of OO Analysis' (written with G. Ramackers), provides "an initial attempt to produce such a formalism for those notions used to represent the results of OO analysis." The authors favor the language of set theory. As an information model will always specify objects-in-structure, the formalization of object relationships deserves special attention.

Entering into detailed comments on the text of Odell and Ramackers would lead astray. Their principle is already fundamentally different from metapattern's; their tenet is that a concept serves "to classify those things around us." As far as its formal aspects are concerned, a concept is known by its intension and extension. The intension "is its meaning, or its complete definition." This leads to the question what the term "or" stands for. And is "meaning" something different from "complete definition"? Or is "complete definition" a detailed explanation of "meaning"? It is one thing to introduce a concept, especially one acting as a metaconcept, but it's impossible to positively define it. This realistic attitude toward metaconcepts is characteristic for a professional modeler (Wisse, 2001).

The last interpretation of "meaning" seems reasonable, and shows the fundamental difference between the authors' object orientation and metapattern's context orientation. We cannot over-emphasize that metapattern does not provide a positive, complete definition of object. At the most, a definition exists for what an object is *within* a context. Further, the number of possible contexts is infinite. By implication, an overall object may appear in any number of identities. Indeed, for the information set a complete, overall (information) object is postulated; but this is very different from a complete definition of a concept (for a set of objects) in the real world outside the information set. That complete object may well be considered an approximation of the concept definition. As the complete object changes, the definition also changes. A characteristic of such a "definition" is that it is always a relative conclusion, not a firm principle.

Despite their fundamentally different principle, Odell and Ramacker offer valuable suggestions; they remain valid when considering metapattern. For a detailed account, read Odell's excellent collection. Metapattern's theoretical foundation is formally presented elsewhere (Wisse, 2001, 2004).

8. Contextual state at specific time

Change and state are aspects of the same phenomenon. According to Odell, what changes in an object is its state.

But what *is* object state? In the essay 'What is Object State?' Odell's remarks are equally valid for metapattern. Strangely, he doesn't appear to reckon distinctly with time.

Metapattern may be contrasted against this where time is dealt with *as a principle* (Wisse, 2001, 2004): a particular state is only valid *in an explicit relationship* to a certain point in time.

Metapattern views the concept of state contextually. A particular context is always what determines the relevant part of an overall object; the particular part has a corresponding contextual identity. The state of any such object part covers the existence of its identifying node as well as its context and intext. The state of an overall object—with an overall object's skeleton consisting of all nodes pointing to the same nil identity—is the collection of all object-part states. The question: is there any practical relevance in considering the overall object's state? A good reason must exist to differentiate behavior. This leads to contextual states which (precisely because contexts are supposed to be disjunct) are largely independent. It is thus contextual rather than overall state that puts time in proper perspective.

9. Dynamic, multiple typing

Contrary to his opening sentence in 'Specifying Structural Constraints' (see § 6, above), Odell begins the essay 'Dynamic and Multiple Classification' as follows: "Object-oriented analysis should *not* model reality - rather it should model the way reality is understood by *people*."

Metapattern's ontological paradigm agrees with this statement about the relationship between "the" reality and information about reality. However, in the same essay Odell advocates that "OO analysis should not be based on any implementation technology." He also mentions that "[o]ne of the reasons why the OO approach has been so successful is because the shift from concept to implementation is smaller than with conventional approaches."

Indeed, by applying metapattern instead of conventional object orientation, the gap between conceptual information model and practical information systems may be further shortened. But a fundamental difference between the two extremes remains (Wisse, 2001). The gap between tools for life and living for tools can never be completely closed. Rather than continue with ill-fated attempts to close it on the basis of a single focus, the duality of focus should be recognized through at least two metamodels. On the conceptual side, metapattern is a powerful metamodel. Its conceptual results need to be translated into construction/implementation models required to switch focus to tool technology.

Odell defines dynamic classification as the system's property—an object may be declared a member of a certain class at a particular time, declassified as such at another time, and so on. The fact that no components are available for object-oriented implementation of such dynamic classification is something he views as a problem. He believes that too wide a gap exists between information model and implementation.

Metapattern suggests that implementation components can go a long way to behaving as the conceptual information model requires for dynamic classification/typing. First, it is not "the" object that changes its type—the change is always limited to a particular context, and within any context are existence entries to (re)construct the state of such a partial identity at any point in time.

In this way metapattern offers more than an elegant solution to dynamic typing requirements. Precisely such information requirements have led to metapattern's design; the need for dynamics of information objects, between contexts and in time, has been its primary source of inspiration.

In addition to dynamic classification, "an object can have multiple object types that apply to it at any moment. When an object is an instance of more than one type, this is called multiple classification." With traditional object orientation, then, a problem emerges whenever the particular object is an instance of "multiple classes that are not implied from a superclass hierarchy." As a first implementation solution, Odell says, every combination of (conceptual) types leads to a separate (implementation) class (he adds that this approach has several disadvantages). The second solution: object slicing, in which the object is divided into as many parts as there are different types.

At this point Odell is close to concepts central to metapattern, but, although he seems to favor the second solution, he remains hesitant. The idea of object division apparently runs too much against the established principles of object orientation. Coming as he does from a world of objects and the indivisibility of any object, it's understandable that the second option is only awarded exception status.

From the philosophical foundation of metapattern, the division of an overall object is no exception but the absolute rule. This paradigm shift in conceptual information modeling supplies many awkward problems in traditional object orientation with highly compact, elegant solutions which directly follow from principles of context and time.

Metapattern aims at multiple and dynamic typing. Each context completely or partly supplies a corresponding identity—one of possibly many of an overall object—with a relevant type. Again, this greatly shortens the gap between conceptual and implementation models. It's a precondition

for making an alliance between appropriate metamodels successful (Wisse, 2001; there, see the second essay in the Appendix).

Odell ends the essay 'Dynamic and Multiple Classification' by declaring his support for "enhancing OO programming languages so that these notions are directly supported." He is correct, but greater advantages are achieved by entertaining a more powerful approach to conceptual information modeling—metapattern, for example. Metapattern's context orientation is not only richer for conceptual modeling, it's also a rigorous frame of reference for better implementation tools/components for the precision of behavioral differentiation.

10. "Add" as a single basic operation

A student of conceptual modeling may be confused by Odell's essay, 'Events and their Specification.' To a conceptual modeler the title suggests a treatise on events in the real world and how they may be modeled for optimized information services. However, the essay sums up operations that should be available for digital (not conceptual) information objects and/or relationships between those objects. He calls such operations "events," indicating that his primary perspective (at least when he wrote this essay) concerns implementation of object orientation—not conceptual information modeling. This explains why the solutions he offers remain within the scope of traditional (technologically-focused) object orientation—notwithstanding how close he comes in other essays to metapattern principles (above, see § 9 on multiple typing).

According to Odell, all operations may be reduced to add and delete. The only *basic operation* required by metapattern is add**,** following from the existence entry included in any registration. In principle, all information remains in the information set (Wisse, 2001, 2004).

Of course, the claim of reduction to a single basic operation is made here only from a conceptual perspective. It would be too hasty to conclude that a single conceptual operation could eventually be supported by a corresponding single digital tool operation. What metapattern does is suggest a construction metamodel, and what follows should be read in this predominantly conceptual light.

The equivalent of delete is also an existence entry (an add operation). It specifies the value non-existence together with a time value which indicates to starting time of that existence mode. (This is another example of maintaining a more abstract structure of the conceptual information model by choosing an appropriate boundary value. See § 5, above, for some guidelines on balancing modeling abstraction with boundary values.)

The radical application of the add operation for all information changes makes the information model more realistic; thus, the actual information system will perform more realistically: along an active time dimension, phenomena do not permanently disappear. Because they *existed* at some time, a reconstruction at that same time must make them appear to *exist* again. For this reason, metapattern views time as a fundamental category; a specific time value is present for every information object and relationship.

Merely adding information could lead to an unmanageable volume, so it should be possible to remove information from the operational set/system before that happens. A specialized housekeeping activity is defined for that purpose, amounting to—how else?— application of the delete operation. From the perspective of metapattern, such delete is *not* a basic operation. The information set is not changed based on the equivalent of a real event—a change outside and independent of the set/system. In other words, a delete operation in the information set does not correspond with specification of a real event. This correspondence, at the heart of conceptual information modeling, does exist for the add operation.

When a cleaning operation would leave an overall object with just its nil identity, such an object has evidently lost its right to continue to exist in the information set. That particular nil identity, too, will be removed. In practice, this seems rare. All information, including past and future, is considered relevant, so it may never happen that an overall object is completely "cleaned up."

11. Characteristic modeling paradigm

Every state change—in a machine, for example—can be modeled to occur instantaneously. It follows directly that such a machine always exists unambiguously in a specific state.

The transition from one state to another (or, more accurately, the commencement of a different state) may have external effects. For example: an effect on another "machine" and a resultant change to its particular state.

The smallest volume of information content, when applying digital information and communication technology, is a zero or a one. Abstracting from this smallest unit to any information object, and assuming instantaneous transitions, information systems (especially digital) exist in a specific state at any point in time. Between digits and the complete set, any subset has this finite quality. This characteristic makes digital information sets eminently suitable for simulation and/or control of other "machines" with variable states which are/should be as unambiguous as possible at any time.

In 'Approaches to Finite-State Machine Modeling,' Odell takes the inverse approach, showing the advantages/disadvantages of describing object behavior *as* a finite-state machine. When such

conceptualization is achieved, subsequent implementation is straightforward because each machine corresponds to an object class.

Odell does not further explore the idea that a conceptual model assumes that "something" in reality works according to a system of finite-state machines. The behavior of those "original" machines is modeled *onto* or translated *into* object behavior, with objects acting as derived machines. Odell doesn't distinguish—as required for viable, life-focused conceptual models—between a machine-in-reality and an information-object-as-machine-in-the-information-set. He starts with the implementation and then seeks a model to accommodate it. He is right to take that conceptual model as the relevant part of reality for implementation, but he implicitly places "the" reality underlying the conceptual model in the same perspective as that model. For purposes of implementation there is no cause for confusion. However, when the modeler's activity is conceptualization rather than implementation, the absence of properly-distinguished perspectives is confusing and can obstruct quality. Odell does apply his single perspective consistently; for example, he sees events primarily from the implementation point of view (see § 10).

Odell's approach confirms how strong the influence of technological professionals is to extend their focus, something which generally happens with the best of intentions. As the term paradigm suggests, it's nearly impossible to see beyond a particular paradigm; it is only too natural to see the rest of the world from the same perspective, too. Though understandable, it's not necessarily right. An independent, professional conceptual information modeler should ensure that such implementation bias is removed as much as possible when modeling information from the point of view of its *original requirement*. By the same token, a conceptual modeler should refrain from construction of complex information systems. The underlying paradigms are just too different to be reconciled by a single professional.

It will always be difficult to distinguish between reality and a conceptual information model when the ultimate perspective is governed by implementation. Seen from an implementation viewpoint, it's impossible to discern the conceptual information object and the assumed object in the real world. At that stage, only a single kind of object is visible—the information or (perhaps) digital object.

Again, this inevitable bias of implementation suggests the need for a fundamental change. Conceptual modelers must consider the possibilities and limitations of implementation. Above all, to succeed in conceptualization, a characteristic paradigm for conceptualization must be applied. The right modeling paradigm keeps the modeler concentrating on the relationship between reality and her/his (conceptual) representative information model. The implementation

paradigm has a different theme, reflecting essential properties of the tools used to actually construct the information system/set structure.

The contents proper of Odell's essay on finite-state machines give little cause for discussion. What might be explained is that metapattern allows reality to be modeled as finite-state machines to an even larger extent. Odell gives several reasons on when to abstain from that approach, one being "[w]hen an object is viewed as being in many states at any given moment." This can be defined in more detail with metapattern, as it not the overall object but its contextual parts that must meet the criterion of practical "finite-stateness." Inversely, it should be possible to define a specific context—and at the same time the type for the object part—so that the condition of finite states holds.

With "many states," Odell does not refer to the complete state of a particular object at a specific time, as this would disqualify the object a priori as a finite-state machine. He is no doubt speaking about the set of properties, each property having its own state.

An information model of a particular finite-state machine must describe each state transition. In terms of metapattern, a transition boils down to a composition consisting of a begin state and an end state. A transition's intext will include, among other properties, the relevant conditions. A particular condition may rule that one or more states should/should not have existed previously. Such a memory is exactly what metapattern offers as a matter of principle.

For the proper conceptual model, an important consideration is whether the finite-state machine is involved in a single, continuous process or discrete (trans)actions. With different transactions, details must be accommodated and the model must consider the possibility of several machines.

Odell presents specific examples in some detail. Figure 11, while simply an overview, adds the distinction between machine types. Metapattern invites the modeler to generate (more) flexible models.

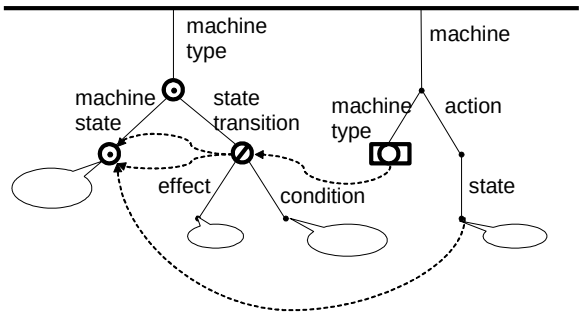

Figure 11: Information pattern of finite-state machines.

12. Intext with static and dynamic properties

Every state transition of a finite-state machine may be described with rules, but Odell doesn't refer to such machines in 'Business Rules' and 'Using Rules with Diagrams.' The first of these essays contains a general categorization; in the second he offers several suggestions for integrating rule presentations in formal schemata. He emphasizes that "rules can also be executable specifications for an automated system."

To bring rules into the domain of finite-state machines, a boundary case must be defined as an assurance. This concerns preservation of state. Preservation is a special case of transition—the absence of any other state. Preservation may also be governed by explicit rules.

Metapattern is first concerned with conceptual information modeling. From that perspective, rules should be considered a property. Every rule will thus appear as part of an intext, attached to a particular node. For example, in figure 11, the relationship effect leads to a node that "contains," among other parts of its intext, specifications of the required rules; those specifications will consist of intermediary, pointer and/or primitive information objects.

Odell considers conditions as belonging to rules. But conditions, and possibly processing rules, may be modeled separately, with everything connected by a node (figure 11).

As with all information, rule duplication must be avoided as much as possible. Rule specifications are thus modeled at the type level. A rule is always executed, however, for an object instance. Every type is present in the information set as an instance in its own right; to accomplish state transitions of a type, its metatype supplies the rules, etc. When a particular state transition should also affect another machine (information object as node), the rule-based action is limited to a trigger; then the rule as specified for that other machine's type takes over to establish the actual effect.

Intext holds the description of the behavior of an object's partial identity in its corresponding context (Wisse, 2001, 2004). So far, only static properties have been discussed, whereas behavior also implies dynamics. Through pointers to rules—type-based or not—each object's part with its contextual identity has the capacity to change. That changes are almost always externally triggered does not deflect from this capacity. After such an impulse has been literally taken in, the partial identity itself controls its dynamics. That is to say, with access to rules in its intext, that same intext does indeed describe behavior. Static properties (state) may provide information for the dynamic properties (rules) to determine the next state, including possible triggers to effect state transitions elsewhere.

When the actual rule is supplied for the type rather than the instance, it is for a static property of the type. The type's dynamic properties, when specified, refer to metarules implemented as static properties of the metatype involved. In this way, an elaborate hierarchy for abstraction may be modeled and, subsequently, constructed.

13. Degree of freedom and purity

Most of Odell's more specific conceptual problems with object orientation are presented in his collection's first eleven essays. They provide an ideal background for explaining several aspects of metapattern in detail and for sketching some equally specific metapattern-based alternative models. The second half of Odell's book contains eleven philosophical essays. Those are used to place additional emphasis on general philosophical aspects of metapattern. Conceptual information modeling is, after all, applied philosophy.

Classification establishes a relationship between an instance and a type. For example, if we call Smith a man he is classified or typed *as an instance of* the male type. If we call a man a human being, we must deal with generalization: the relationship is concerned with two types, where man is supposed to be seen *as a subtype of* human being.

In 'Managing Object Complexity,' Odell uses a similar approach to explain the fundamental difference between classification and generalization/specialization. The difference is indeed fundamental, but Odell seems to view it as absolute.

As an alternative, metapattern allows the concepts to be interpreted in a relative and more flexible manner. Every information object is an instance of its own complete context. A context *classifies* the information objects it contains (Wisse, 2001, 2004). The particular node determines two things: what counts as instance (being the particular node itself); and which relationships and other nodes constitute its type. Of course, results differ between all nodes. Different nodes may share their complete context or—as their type—identical context subsets (see figure 12 for a schematic summary).

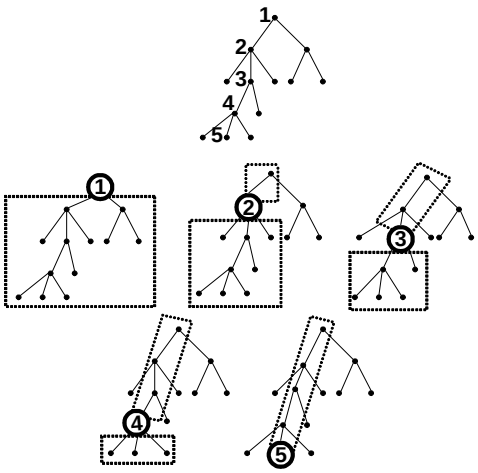

Figure 12: Relative nature of context and intext.

Subtypes, as indicated by Odell, are often more productively modeled as disjunct contexts for a corresponding variety of partial identities of the overall object. In these instances, a more general type does not necessarily serve to act *only* as a node connecting more general information; it can also be the node supplying each instance with specific information as intext (but with that intext's structure being generally valid). All this can be modeled as a single hierarchy. Hence the reappearance, in different locations, of the same relationship type.
However, when a single hierarchy may be replaced by a series of hierarchies without loss of options for classification, a question arises: Is the original generalization/specialization pure (enough)? Figure 13 shows the range of such alternatives.

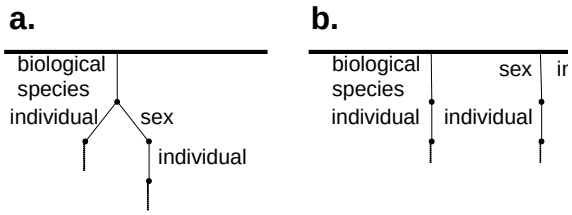

Figure 13: Sorting out generalizations.

Traditional object orientation is biased in the sense that it is ideally based on a single typological hierarchy. The relationship between information objects may thus be interpreted wrongly as a case of generalization/ specialization.
Metapattern avoids this problem altogether because context orientation does not presuppose a single hierarchy. In fact, there is no preference for one or more hierarchies. And the principle of

multiple contexts seems to allow a special freedom for conceptual modeling. It turns out that, through abstraction, a homogeneous classification hierarchy results, offering a purer model of generalization/ specialization. The paradox: larger freedom at implementation subsequently results in greater simplicity. A precondition is that implementation tools must incorporate metapattern.

In the second part of the same essay, Odell explains the aggregate (a whole). For metapattern, every intermediary information object is the identification of a whole, with intext for its parts. On its own terms, any part at the next level of specification may act as a whole itself (also having an intext for its parts). And so on. With metapattern, no information object (node) is a priori atomic. Decomposition into properties (next-level intext) may continue indefinitely. The modeler set limits through interpretation of relevant information requirements.

Odell considers a composition to be a special case of an aggregate. A necessary condition reads that one or more parts must be present. The boundary case is the composition with just a single constituting part. The lower limit could be similarly defined for the Cartesian product in its capacity of a composition (Wisse, 2001; see especially chapter 5).

14. Structural set for specification of aggregates

Odell wants to prevent developers from wasting energy by reinventing software constructions. For this reason he devoted an essay, 'Six Different Kinds of Aggregation,' to the similarities and differences by which parts can relate to their whole and each other. Each type (why does Odell avoid to mention type in this context?) requires characteristic operations. After development, such operations are available for every corresponding aggregate instance.

What Odell did not elaborate upon is the conceptual modeling of the "functional or structural relationship to one another - as well as to the object they constitute." However, another essay, 'Toward a Formalization of OO Analysis' (see § 7, above, for comments), contains an important clue in this regard: it defines the power set of a set S as the set of all subsets of S. This definition makes every element of the power set a candidate for modeling the relationship with the whole. Such an element may also be considered the basis for modeling relationships between the constituting parts. It makes sense, of course, to limit attention to those particular members of the power set whose description contributes to structural understanding—that is, how parts relate to each other and to the whole.

The (sub)set of structurally relevant elements/members of the power set of (aggregate) S is here defined as the *structural set* of S. Members of such a structural set are called structural elements. The intext of every structural element, therefore, contains the model of "its" corresponding

substructure of the complete aggregate. The concepts of whole, part and structural set are modeled in figure 14.

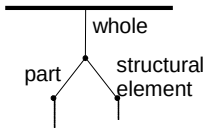

Figure 14: A structural set is a subset of a whole's parts.

15. Rule, no exception

'A Foundation for Aggregation' and 'A User-Level Model of Aggregation' were written by Odell with C. Bock as co-author. Both essays give special attention to parts which appear in various aggregates/wholes where they subsequently play different roles. An engine in a car, for example, is different in its behavior from a submarine's engine. That is why, as they suggest, engine needs to be subtyped: car engine and submarine engine, respectively.

According to the authors, what is involved are "the parts as necessary within the context" of a car or a submarine. They further state that "current methodologies, even though [these features] are needed in common applications," do not support that "[p]arts may be connected in certain ways unique to the composite…[that] each part has an identifiable role in the composite…[and that] parts may be assigned properties unique to the composite."

But now metapattern does provide such support, even as a matter of principle. As Bock and Odell write, "[a] qua-type is a subtype created solely to support a role"— a statement which shows their perception that such a role is exceptional. Apparently, they consider as a normal case a type with *general* application.

By contrast, metapattern assumes multiple contexts from the start. That is, *specialized* types are the rule, not the exception. An overall object, by definition, does not lead a general existence; it only works through specialized roles within corresponding contexts. A consequence is that subtyping, as meant by Bock and Odell, is unnecessary because the various contexts (of engine, for example) constitute just as many types (see figure 15). Thus, every context *is* a type, not a subtype of a more general type. Metapattern does not require "context-based subtypes." Type-on-context follows from the principle of context orientation. Figure 15 shows that cardinality constraints may differ from context to context (a submarine will often have more than one engine).

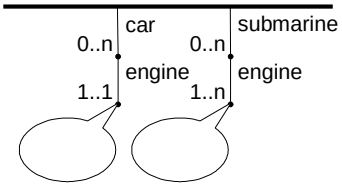

Figure 15: Straightforward modeling per context.

An important advantage of this: a smaller set of (basic) building blocks is sufficient to model a richer variety of information. Schemata, too, remain compact, as the contextual differentiation directly draws attention to what each *role* implies as specialization.

The authors are fully justified when stating that "[a]ggregation means that a class can describe the part structure of its instances." Metapattern limits such description to the relationships, and thereby related information objects occur at only the first level of the intext (Wisse, 2001, 2004). Through the principle of type-on-context, "all of the type-based services are available" for every information object.

16. Limitation of patterns by metapattern

In 'From Analysis to Design Using Templates,' Odell co-authored three essays with M. Fowler. The authors review a series of templates, or patterns, for basic and composite operations in information processing. Such patterns help to increase standardization which, in its turn, should contribute to improvements in the quality of the development process (means) and the resulting information system (end). Emphasis is placed on having specific development rules and their derived design patterns reflect both the approach to conceptual modeling and the infrastructure of information and communication technology. Whenever the modeling approach and/or the technical infrastructure changes, the design patterns must be adjusted accordingly. They also offer a general explanation as to why metapattern, as an approach for conceptual modeling, might lead to characteristic design patterns.

Earlier, in § 13, a paradox is presented: initial distance to implementation eventually increases its simplicity. This is particularly true for metapattern. As context and time are dealt with structurally, uniform rules apply. Odell and Fowler see the need to list, somewhere between rule and exception, several design patterns for dynamic and/or multiple classification. They deserve full credit for already recognizing a major part of the structural nature of such patterns. Now, with metapattern, an even more compact set of rules is modeled because the structural nature

has become a matter of principle. A prominent example is the one remaining basic operation left in metapattern: add (above, see § 10).

Because this paper emphasizes conceptual information modeling, no specific patterns for processing operations are listed for metapattern (as Odell and Fowler did for traditional object orientation in 'From Analysis to Design Using Templates'*)*. The manner in which they position the conceptual information model in the overall development process is relevant. The "model performs two roles: as a conceptual picture…and as a specification of the software components." That is, the conceptual model occupies the critical position for successful translation: it is a model, both to(ward) and before reality. The first reality lies outside the information set. For it is *as if* information will be shaped as representation of that reality (and in an operational information system the proposition is that information is indeed registered *as if* it represents reality). The second reality *becomes* and, through implementation, *is* the information set.

17. Context: background at foreground

In 'Method Engineering,' Odell writes about context, but in a different "context." What he refers to appears to lie outside the information set. For metapattern, however, context also exists within the information set. It has become the critical concept for unambiguously modeling the fundamental pluriformity of object behavior.

The essay entitled 'User Workshop Techniques' is interesting but does not specifically deal with object orientation. There is thus no reason to discuss it here.

In 'Object-Oriented Methodologies,' Odell recommends object orientation for "systems in general." That is, indeed, a valuable suggestion. But is OO actually any different from the already well-known general systems approach? By the way, Odell traces the development of object orientation to its origin as "a particular kind of programming language." From that, there grew "a broader interpretation [which] means that OO is a way of organizing our thoughts about our world." This interpretation should be elaborated upon. In the human perspective (that is, in reality) objects never appear fully self-contained but always, as Gestalt psychology has shown, as a foreground against a background. Metapattern starts with the recognition of this double movement in the single act of human understanding.

The complexity of conceptual information models might be suspected to increase by differentiating between foreground and background. Rather, the reverse occurs. Models become simpler and compact. Contemplation reveals that foreground (information object) and background (context) are not absolutes. Instead, they determine each other. Their dynamics

further explain that every context may also be expressed meaningfully by one or more related (other) information objects. This preserves (information) objects and their relationships as the basic building blocks of information models. However, the general approach to modeling has changed fundamentally from object- to context-oriented. By including time as a fundamental dimension of information, the set of basic concepts is still very limited. Applying those concepts can yield compact models featuring great variety. The actual information system is correspondingly flexible and adaptable. Metapattern does, indeed, also lead to an adequate solution for all the conceptual problems reported by Odell. Metapattern solutions are superior where dynamic and/or multiple classification are required. With such problems, it's *as if* the model is more realistic—that is, a closer model of reality. This is now easily explained by context and time as fundamental categories. From this it is reasonable to conclude that metapattern constitutes a richer approach for conceptual information modeling than purely object-oriented approaches.

notes

1. This paper is largely derived from part II of *Metapattern: context and time in information models* (Wisse, 2001).

2. Metapattern's visual language is fully explained in Wisse (2001 and 2004). See also *www.informationdynamics.nl*.

literature

Odell, J.J., *Advanced Object-Oriented Analysis & Design Using UML*, Cambridge University Press/SIGS, 1998.

Wisse, P.E., *Metapattern: context and time in information models*, Addison-Wesley, 2001.

————, *The pattern of metapattern: ontological formalization of context and time for open interconnection*, in: *PrimaVera*, working paper series in information management, nr 2004-01, Amsterdam University, 2004.