PrimaVera Working Paper Series

UNIVERSITEIT VAN AMSTERDAM

PrimaVera Working Paper 2004-01

The pattern of metapattern: ontological formalization of context and time for open interconnection

Pieter Wisse

January 2004

Category: scientific

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

The pattern of metapattern: ontological formalization of context and time for open interconnection

Pieter Wisse

Information Dynamics, Voorburg

Abstract: Metapattern is a technique for (meta)information analysis and modeling. Context and time are critically important, allowing for adjustment (or re-adjustment) of a model to time-induced and/or situational changes which it must account for in order to maintain its integrity. Context is included as a formal variable *within* information sets, instead of seeing context, often implicitly and therefore unrecognized, as an informal presupposition that is kept outside. An information object may appear in multiple contexts, with unambiguously corresponding variety of behavior. By paying consistent attention to the aspect of time, the approach is augmented even further.

Keywords

Metapattern, conceptual information modeling, ontological engineering, situational variety, multiple contexts, temporal variety, accountability.

Author

Dr. Pieter Wisse (*www.informationdynamics.nl/pwisse*) 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

1. Contexts **5**

2. Intexts **16**

3. Types **24**

4. Time **33**

Metapattern for ontological engineering

Ontologies have entered the scene of information modeling. Ontological engineering should be recognized alongside software engineering. As a tool for ontological engineers, metapattern[1] demonstrates three essential characteristics:

1. Context is a structural concept: Through its partial identities, an overall object may be contained by a variety of contexts. Different contexts facilitate a corresponding, unambiguous variety of information object behavior. Multiple contexts within the information model most elegantly fulfill the increasingly relevant requirement of multiple classification.
2. Time is factored at every meaningful node. As a matter of routine, the period(s) of existence of information objects (meaningful nodes) at each level of granularity versus aggregation is kept track of. As metainformation is modeled just like any information, the increasingly relevant requirement of dynamic classification is fulfilled by default.
3. Validity is factored at every meaningful node. Errors, intentional or not, are a fact of life. Information discovered to be mistaken is not removed but maintained with a non-valid status. The new information entered as a correction takes on the status of valid. Combined with time-factoring, a comprehensive audit trail results by default.

These characteristics, taken together, make metapattern a powerful approach to conceptual information modeling. Metapattern supports multiple and dynamic classification, as well as a high level of accountability. It not only solves many tough modeling problems elegantly, it also creates additional opportunities for information systems.
How the world is believed to exist on a fundamental level is known as ontology. The word, which is derived from ancient Greek, means “knowledge of being.” Other knowledge may then be grounded on such essentials. Another way of putting it: an ontology covers axioms—or, as some prefer to call them, first principles.
A common ontology still takes that absolute objects, or entities, make up the world. Such an assumption may work well enough in most of daily life. With the need for integrating growing diversity, uniform concepts no longer hold sufficient variety. By super-imposing the concept of context, and especially by doing so combined with the concept of time, earlier uniform concepts are given enough room to shift their meanings to pluriform usage. A contextualized object is different from an absolute object as assumed by, for example, traditional object orientation.
The metapattern way of explaining the world denies the absolute existence of objects. Please note that objects are still believed to exist—very much so, in fact. It is their *absolute* existence which is removed as the one and only starting point for explaining the world. This is done for a practical purpose, for behavior is often, and increasingly, diverse.

How can different behaviors be attributed to a single object? Metapattern assumes that specific behavior is always determined by an object *being in a situation*. The same object in different situations will show different behaviors. Take yourself as an example. Overall, you surely feel you are one and the same person regardless of the situation you find yourself in. Yet, your behavior at home might be different from how you behave at work, in your car, and so on. Variety is a characteristic of behavior. Metapattern is all about balancing variety of behaviors while maintaining an object's basic unity.

For its closer connotation with information, the term *context* is preferred to situation. As explained above, while metapattern leaves absolute existence of objects behind, the existence of contexts is assumed to be even more essential. It means that any object may exist in several contexts. By definition, the object's behavior in one context is different from its behavior in all other contexts. So, by placing the concept of context before that of object, the difference of metapattern is of an ontological nature.

Metapattern's innovation is the *radical* importance attributed to the concept of context. Also essential is this concept's implementation—context as an open-ended recursion of object-relationship pairs. How this is achieved is explained fully later. With multiple contexts for any object, differentiating behavior is a straightforward matter.

Considering *context*—not object—as the first principle is the very paradigm shift of metapattern. As a consequence, an object can only exist *within* one or more contexts. The type which determines behavior is no longer for an object-as-such but for an object-in-context. Actually, context *is* type. Objects keep their relationships. Based on the new principle, however, a relationship between context and object is added. This procedure also has a compensating quality, as it defines even context as object. Advantages for compact information modeling follow.

Actually, given the current state and projected development of the world, it is logical to declare *context* a fundamental concept for information modeling. And *time*, too. Most people will increasingly experience the world as multi-faceted, highly pluriform. And accelerating changes make the world ever more dynamic. In fact, modern information and communication technology itself is now a prime driver of this postmodern human condition.

Social psychologists have known for a long time that the overall behavior of a person is never completely consistent. Rather, consistency is limited to what is called—in metapattern terms—a context. But this does not mean that modelers should apply the opposite idea and postulate as many different persons as there are relevant contexts. Overall, it is also still one indivisible person. Metapattern provides the balance between identity and difference, a balance which should be managed in open interconnection.

1. Contexts

Metapattern identifies several concepts in a characteristic semantic structure. This paragraph develops a basic concept—that of context—and elaborates upon the connections between context, object and relationship (resulting in an even wider definition of object, i.e. as seen from the dual metapattern perspective).

Contexts with object

An object behaves according to a particular situation; that is, its behavior can change from one situation to another. For instance, if you only know Bill as your next-door neighbor, you might be surprised to see him in action at work or visiting with his mother. It is this behavioral variety, and the need to model it conceptually, which has resulted in the assignment of primacy to the situation rather than the object. Another word for situation is *context*.[2] Thus, primary attention shifts to what is *around* an object.

The term context is favored here because most people seem familiar with its use in the sense of appreciating why something is not always completely the same thing. Metapattern presumes that an object's behaviors are completely different from one context to another. Odd as it may look, the absence of *any* shared properties by an object among its contexts exemplifies metapattern. The minimal mechanism holding the object's contextual identities together is explained at the end of this paragraph.

Suppose a particular context **A** exists. And suppose that the equally specific object **x** is contained by **A**. An example would be Bill (**x**) at work (**A**). This can be straightforwardly modeled as in Figure 1.

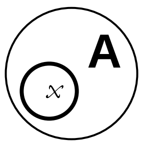

Figure 1: Object in context.

Another context, say **B**, could also contain the object **x**. This is shown in Figure 2. Bill (**x**) is either at work (**A**) or at home (**B**).

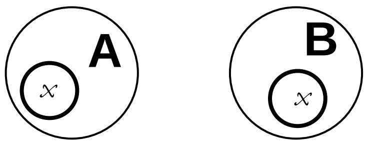

Figure 2: Object in multiple contexts.

Contexts **A** and **B** may overlap, as in Figure 3. This happens, for example, when Bill works at home.

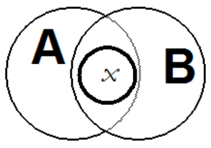

Figure 3: Object in overlapping contexts.

However, any overlap can be reduced (in this case, to three disjunct contexts). One context applies strictly to work behavior, the second to home behavior, and the third to behavior in which work and home are mixed. These derived contexts are defined using the accepted notation of set theory. In general, the set containing all elements outside any set **A** is denoted by **Ā**. As such reductive conversion is always possible, contexts are assumed to be disjunct a priori for simplicity's sake (Figure 4).

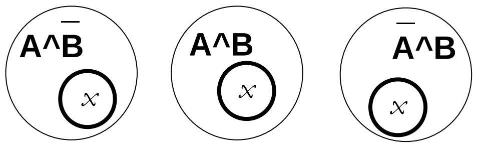

Figure 4: Conversion to disjunct contexts.

On instances and types

It's important to distinguish between instances and types when necessary. To call an object a person, for example, usually means that a particular *object instance* is referred to which obeys the *object type* of person. Actually, listing relevant elements is the extensional approach to defining a set. A type specifies a set intentionally—that is, through conditions which must hold valid for elements to qualify as set members.

The context in which the term object appears often makes it immediately clear whether that object's instance or type is meant. To avoid cluttering text, we'll use this implicit procedure throughout this article.

Object with contexts

Figure 1 suggests that object **x** exists *in* situation/context **A**. Simultaneously, however, the idea of context leads to independent consideration of the object. This postulates an explicit relationship; that is, that between a context and an object. Let's call the relationship between context **A** and object **x**, **a.** Thus, Bill (**x**) is an employee (**a**) at Case Study Corp. (**A**). To make for an easier overview, put **x** below **A**, formally retaining **a** within the context (as shown in Figure 5). As part of **A**, something of a subcontext **A'** originates, where **A** = {**a**, **A'**}.

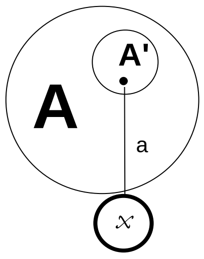

Figure 5; An explicit relationship between object and context.

Because relationship **a** remains part of context **A**, every context of **x** is unique. A relationship **b** that could connect **x** to **A'** would lead to a different context. Bill (**x**) could also be retired (**b**) from Case Study

Corp. (**A**). This also holds for relationship **a** between object **x** and another context **B**. Indeed, this reflects Bill (**x**) also being an employee (**a**) of Trial & Error Inc. (**B**).

The formal distinction between context **A** and subcontext **A'** only continues to be specified in the text if the argument is otherwise confused. Where an explicit relationship appears, it is deemed simpler to use only a single capital letter to indicate a subcontext.

Another schematic convention: the line representing the relationship runs into the same set the object seems to have left *because of* that relationship.

For context **B**, everything said about **A** holds true. Therefore, object **x** can be involved in relationships with various contexts. Figure 6 shows the abstraction with two contexts.

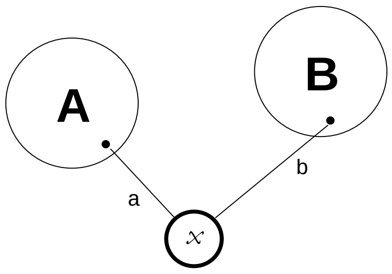

Figure 6: An object's relationships with multiple contexts.

Figures 1 and 5 are equivalent, as are 3 and 6. In both 5 and 6 the context has been objectified. Context and object are placed apart. An explicit relationship between them is required to maintain a view on their original synthesis.

With the relationship subsequently defined as part of its context, an object is optimally positioned for unique behaviors. Note the plurality of behaviors. An object's behavior may be differentially specified on the basis of relevant contexts.

On reality and information objects

The question as to the degree in which reality and information concur defies a comprehensive theoretical answer. For most practical purposes it is highly "realistic" to interpret information *as if* it represents reality. The information objects should then be considered representatives of real-world objects. But what about contexts? And relationships?

According to metapattern, it's also *as if* they exist. Are they actually objects? With metapattern they also have information objects as their "representatives." This is what was meant earlier when stating that contexts are objectified for the purpose of information modeling. And relationships are, too. However, the formalization of relationship is kept straightforward throughout this metapattern exposition. It is established from the perspective of objects (including, at this stage, whole contexts). Thus, a particular relationship marks a difference between objects while maintaining them in an overall structure. For this purpose, the information object acting as a relationship is everywhere represented by a named link. Together with objects, relationships help to form contexts. At the same time, a particular relationship *appears* in a certain context, too. The meaning of a named relationship is therefore always unequivocally clear.

Recursiveness in context

The explicit relationship between context and object raises the question of how the connection inside the context is established. For that purpose, as a matter of design, another object is assumed. In other words, not only is a context generally objectified but its structure also consists of objects and relationships. Metapattern's flexibility is conditioned by the overall application of object and relationship as building blocks in information modeling, including modeling of contexts.

Let **x** be the original object and let **y** be the object added inside the context to serve as a hook for the relationship established between object **x** and context **A** (Figure 7). Bill (**x**) is an employee (**a**) in the Purchasing Department (**y**) inside Case Study Corp. (**A**).

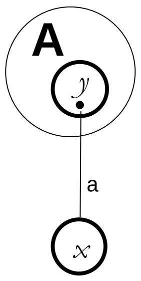

Figure 7: An object's partner in a contextual relationship is another object.

The assumption of object **y** is a powerful procedure. As a principle, context is given primacy (existence before object). In information models, however, a context is treated *as if* it consists of objects (and their relationships). This results in objects also being elements of contexts—in objects recursively forming a particular context.

Previously, through relationship **a**, object **x** was placed below "its" context **A'.** To keep things simple, context is almost everywhere called **A** again (using the same name in no way detracts from the principle).

The same procedure can be applied to object **y**. Drawing on the original context **A**, the overall metamodel is as presented in Figure 8.

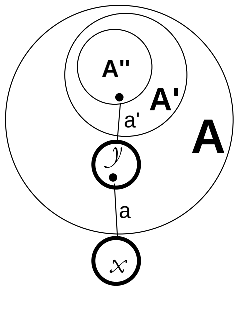

Figure 8: Every object is taken outside its context through a relationship.

Parsing the context of **x** in the manner described can be propagated by, in **A’’**, postulating an object **z**. Bill (**x**) is an employee (**a’**) in Purchasing (**y**), which is a department (**a’’**) of Facilities (**z**), which is a division (**a’’’**) of...Case Study Corp. (**A**). Thus, the original **A** is gradually decomposed, resulting in **A’** first, then in **A’’**, and so on.

The structure of the context—an ordered collection of relationships and objects—is equivalent to a hierarchical classification schema. As such, a context *classifies* the objects it contains. The context as a so-called whole, and the object as a so-called part, together represent just one classification type.

It should now be clear that a positive definition of context (as in positivist science) is impossible. There does not exist something-out-there-in-the-world which only needs to be labeled properly. The metapattern view, in fact, is the other way around. Context is a conceptual construct or an instance of a fiction. This fiction is used to (further) conceptualize reality. For this reason, context is not just any kind of concept, but a metaconcept.

No metaconcept can be positively defined. In *fiction* several metaconcepts are required to establish a metamodel. Metapattern *is* a metamodel. Its only three metaconcepts are context, object and relationship. Context is then defined as a particular structure of object and relationship—there can never be positive proof that such a structure really exists. To look for such proof misses the point. Context, object and relationship have axiomatic status in metapattern. They are fictions in the service of understanding the rest of reality.

Remotely, the procedure of expressing context in terms of object and relationship might be compared to locating the middle point between two points on a straight line. Both original points are unambiguously located, and may be expressed by their distances to a fixed point on the same line. As measures of their location, this would yield $\mathbf{d_1}$ and $\mathbf{d_2}$ respectively. How can the halfway point (i.e., a particular context) be determined? The procedure assumes that such a middle point exists. It is given a variable name—**m**, for example. The vital property of **m** would be that its distance to $\mathbf{d_1}$ is equal to the distance to $\mathbf{d_2}$. Assuming that $\mathbf{d_1} < \mathbf{d_2}$, it follows that $(\mathbf{m} - \mathbf{d_1}) = (\mathbf{d_2} - \mathbf{m})$ must hold. The location of the middle point is thus found as a function of both outside points. The general result is: $\mathbf{m} = (\mathbf{d_1} + \mathbf{d_2}) / 2$.

Likewise, metapattern offers a general procedure to express any context in terms of instances of its other metaconcept objects and relationships. This pervasive *functional* nature of context—its (meta)concept—makes metapattern fundamentally different from other known metamodels for conceptual information modeling. Elsewhere, the fiction of context is not a function, but a completely independent (type of) object/entity.

Metapattern’s functional approach to context is the key to simplicity in conceptual modeling. This holds true even for information requirements of extremely complex variety. Powerful tool construction technology may be derived from this functional concept of context, too.

Boundary

With its structuralist (as opposed to positivist) foundation, metapattern offers great precision for conceptual modeling. Again, a distinction between reality and information is useful, this time to establish the boundary between context parsing—a boundary defined by the complete collection (set) of information made up by information objects.

Introducing levels of inspection, the whole set is called the nil object. This name might seem paradoxical at first, but we'll see that starting from zero helps formalize a principally open-ended progression into ever-more-detailed information objects. As a corollary, metapattern does not set any limit to establishing parts of a whole. No information object may claim atomic status.

Given the primary status of context in metapattern, a specific object **x** minimally has a relationship with the nil object. This is depicted in Figure 9.

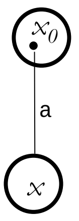

Figure 9: Ultimate context is the nil object.

Suppose $\mathbf{x_n}$ is an object at inspection level **n**. It follows that its context consists of the chain of objects from $\mathbf{x_{n-1}}$ to and including $\mathbf{x_0}$. The recursiveness can also be formally expressed, the context of $\mathbf{x_n}$ being an ordered collection. Context($\mathbf{x_n}$) = {$\mathbf{a_n}$, $\mathbf{x_{n-1}}$, Context($\mathbf{x_{n-1}}$)}, which holds for $\mathbf{n} > \mathbf{0}$. The exception is $\mathbf{x_0}$, which object does not have a context. Another way of stipulating the same notion is that the nil object *is* its own context.

Relative determination of context and object

Figure 10 clearly shows that point of view determines what counts as context and that one context can contain another.

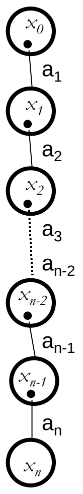

Figure 10: Context decomposition yields a chain of objects.

As the collection determining context grows, the context itself shrinks. When the whole information set (or information system) determines the widest possible context, for example {$\mathbf{a_1}$, $\mathbf{x_0}$}, it is more specific. That is the context of $\mathbf{x_1}$. Consequently, the context of $\mathbf{x_2}$ is the ordered collection {$\mathbf{a_2}$, $\mathbf{x_1}$, {$\mathbf{a_1}$, $\mathbf{x_0}$}}.
Again, what counts as a specific context depends on object choice. Such a point of view is variable. The same object can be either the point of view itself or an element of another object's context (the other object then shaping point of view).
A third possibility exists. Properties of some other objects are also, in their turn, relationships and objects. Metapattern calls properties intext (intext structure is discussed in the next paragraph).
The objects constituting a context are fundamentally identical to the object that happens to be relevant for inspection. With a variable point of view, what deserves to be labeled as context or object appears to be relative (dependent on the point of view chosen). This property of metapattern is powerful, offering great flexibility and compactness. A single approach for conceptual information modeling can now cover a wide variety of problems and opportunities.

Characteristic difference between approaches

What distinguishes metapattern from so many other approaches to modeling is the ontological principle—the context having the privileged status in knowledge. Modelers unfamiliar with leaving the object's existence as a first principle in favor of context will be uncomfortable with this approach and with the lack of positive definitions. But it is precisely the assumption of contexts, objects and relationships *defining each other* that is metapattern's deciding quality. The absence of absolute definitions is not a problem; it's a powerful solution. The relative nature of metaconcepts is a precondition to arriving at the fiction of positive definitions of all other concepts.
In context orientation, the modeler should give priority to parsing contexts. This contrasts with traditional decomposition, which takes an independent object as its starting point. Precisely put, the assumption of an object's independence does not allow one or more contexts to enter the (conceptual) information model.
What happens is always decomposition of an object *within* a specific context. Following this procedure, object properties are detailed corresponding to contexts. This explains how metapattern adds a full degree of freedom to conceptual information modeling.

Multiple contexts

Emphasis is maintained on the object in its context. Let a single object, **x**, be involved in several contexts. The parsing of contexts needn't result in an equal number of levels. Take **A** and **B** as two contexts. With **x** being Bill, context **A** might refer to a particular organization in which he works, while context **B** represents his favorite sports team (whose games he attends when they're played on home ground). Suppose that **x**, relative to its first context, is positioned at level **m;** relative to its second context at level **n**. This is shown in Figure 11.

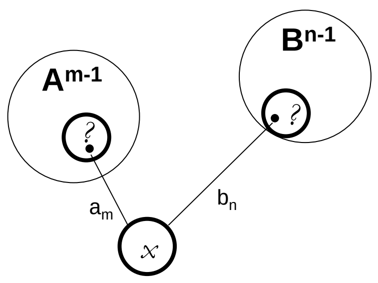

Figure 11: Variable quantity of context decomposition levels.

A question now arises: how can the so-called connecting objects in the contexts **A** and **B** be unambiguously identified? In a multiple setting, $\mathbf{x_{m-1}}$ and $\mathbf{x_{n-1}}$ are no longer sufficient. A suffix must be added to specify the relevant context. For example, $\mathbf{x_{A,\ m-1}}$. A problem remains, however, because of the relative nature of context determination: $\mathbf{x_{m-1}}$, not **A,** is the proper context, but what was indicated was **A'**. A logical mechanism consists of indexing contexts additionally on the basis of inspection levels. In Figure 11, as a boundary condition, $\mathbf{A_0}$ and $\mathbf{B_0}$ are defined equal to $\mathbf{x_0}$.

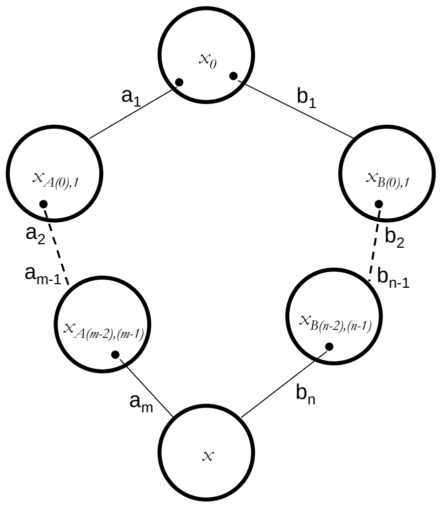

Figure 12: Inspection level added for unambiguous identification of nodes.

Unique nodes

In Figure 12, only the object **x** is left undetermined with respect to both of its contexts. The solution is easily explained by starting with the nil object.

The object appearing on the first level in the context $\mathbf{A_0}$ receives its unique identifier based on the specific relationship $\mathbf{a_1}$. That means that this single node is unique. The node identification process proceeds to higher numbered levels. The implication is important for an understanding of metapattern. In fact, a certain object does not have a relationship with another object in general, but with that other object *in its capacity as a unique node*.

For this reason, Figure 12 still does not reflect the full variety of metapattern. It already shows that a single object can be involved in multiple contexts. Unambiguous navigation, however, demands that the overall object be represented by a unique node *in every relevant context*. This is what Figure 13 captures.

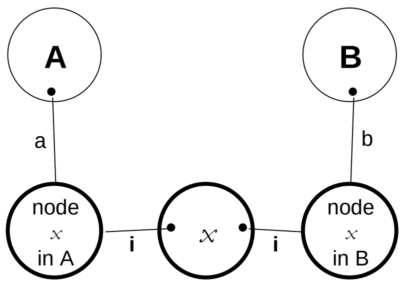

Figure 13: Every node is unique.

Object identity

Through multiple nodes, the radical context orientation changes the meaning of object identity. Actually, nodes provide an overall object with separate identities; that is, with one identity for every context. In terms of his diverse behaviors, therefore, the overall object of Bill is modeled as consisting of partial identities, exactly equaling in number his behavioral variety.

Each overall object has one node with a special status. A mechanism is needed to bridge the separate identities of what is and should remain an overall object. All other object nodes point to the nil node, expressing the object's nil identity. A standardized identity relationship connects a "normal" node to the nil node; to keep metapattern as structured as possible even the nil nodes do not go without a context (Figure 14). Each nil node maintains a nil relationship with the nil object. The acronym for the nil object is: No.

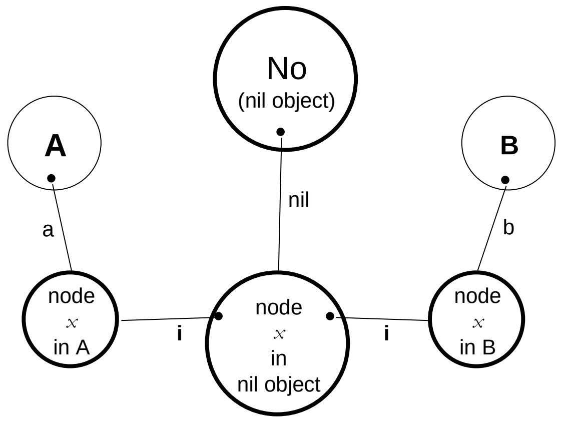

Figure 14: Consistent application of concepts.

Another notion which might appear counter-intuitive is that the node representing the nil identity has hardly any need for properties. Some reflection will make this clear as an overall object only "works" through its identities in (other) contexts.

Different behavioral Bills exist, each represented by a separate normal node reflecting a partial identity. That there is also one overall Bill is exemplified by a single nil identity to which all behavioral Bills refer.

A specific object is, by definition, registered within a context. When registration is not based on a reference to an object in another context, it must be a completely new object and a nil node must be created. Reverse reasoning indicates that an object only represented by its nil node has lost any reason to exist in the information system.

All information objects (except nil nodes themselves) refer to a nil node. This is shown below in Figure 15, an elaboration of Figure 14.

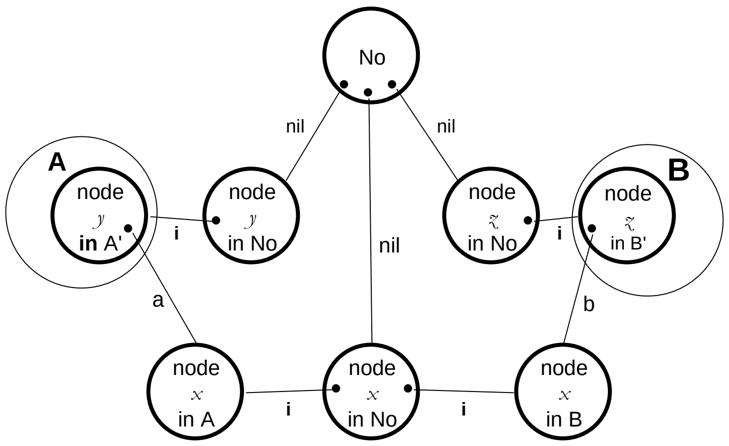

Figure 15: The nil node at the root of the information set.

Even the nil object should have its nil node, clearly the case for a boundary condition which reads that the nil object *is* its own nil node. It follows that they have an implicit nil relationship.

Identity as a network of nodes

Traditional object orientation assigns identity at the level of overall objects. Context orientation replaces this view of singular objects with that of plurality within every object; it always needs a context to uniquely identify a relevant part of an overall object, which is what identifying nodes regulate. When behaviors are identical, no distinction between contexts is called for. However, the Bill-watching-favorite-sports-team will most probably behave differently from the Bill-at-regular-job. With such diverse behaviors, the overall Bill is differentiated into partial identities according to behavior-determining contexts.

As a consequence, the identity is no longer (primarily) of a singular nature. According to the metapattern paradigm, an object's overall identity *is* the collection of context-determined identities. By including the nil identity as a coordination mechanism, the overall identity can be expressed as an ordered collection of identity nodes. Identity(**x**) = {$\mathbf{k}_{\mathbf{x,0}}$, $\mathbf{k}_{\mathbf{x,1}}$, $\mathbf{k}_{\mathbf{x,2}}$, ……, $\mathbf{k}_{\mathbf{x,p-2}}$, $\mathbf{k}_{\mathbf{x,p-1}}$, $\mathbf{k}_{\mathbf{x,p}}$}, where $\mathbf{p} > \mathbf{1}$.

In the expression above, $\mathbf{k}_{\mathbf{x,0}}$ stands for the object's nil identity and nil node, respectively. The second suffix is meant to count contexts. As shown in Figure 16, the only order in the collection consists of the nil node being the center of a star-shaped network.

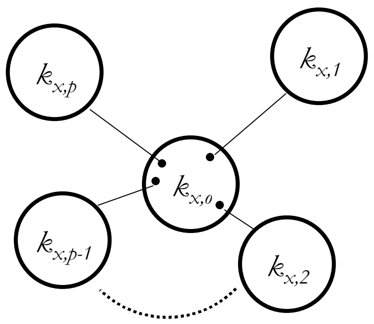

Figure 16: An object's partial identities coordinated through its nil identity.

The (unique) nodes do not need their own inspection level explicitly indicated. Every node's context can now be recursively expressed. The formalization established above serves as a model. Context($\mathbf{k_{x,p}}$) = {**a**, $\mathbf{k_{y,q}}$, Context($\mathbf{k_{y,q}}$)}.
Separate nodes allow the overall object to be modeled according to a corresponding number of contexts. Through networked nodes, the overall object is simultaneously kept together, continuing its existence as a whole.

2. Intexts

The emphasis so far was on context; an object's properties were never explicitly mentioned. Implicitly, however, context decomposition into a recursive alternation of object/relationship instances developed the structure for dealing with properties (attributes). By now it may be evident that an overall object's set of context-oriented identifiers constitutes just as many nodes for attaching relevant properties. By definition, properties are valid within a particular context; a set of properties which corresponds to a specific context is called *intext*.

Objects in context

A minor change of perspective illustrates that metapattern properties follow the rules outlined for contexts. Whereas in the previous paragraph the explanation started with one object having different contexts, the different objects here share the same context (Figure 17).

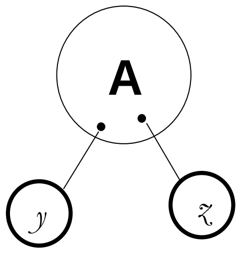

Figure 17: Objects sharing context.

In more detail, the structure resembles Figure 18.

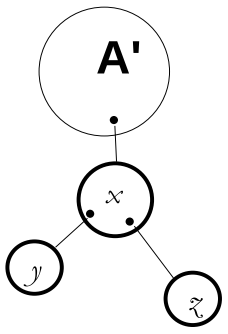

Figure 18: First decomposition of context.

From this perspective, objects **y** and **z** are clearly shown as properties of object **x**. Again, the point of view gives specific meaning to roles of such information objects. Starting with object **x**, **A'** is its context, and **y** and **z** its intext. When object **y** reflects the point of view chosen, its context is constituted by **A** (including object **x)** and there is no intext; i.e., the particular intext set is empty. For example, Bill (**x**), as the employee of Case Study Corp. (**A**), has been attributed a date of employment entry (**y**). In this particular context no definition has been given for any downward decomposition of employment date into next-level properties.

Intext in context

Figure 18 is still misleading, however, because **x** indicates an overall object. This should be replaced by the node serving as context-oriented identifier. Figure 19 is a major improvement.

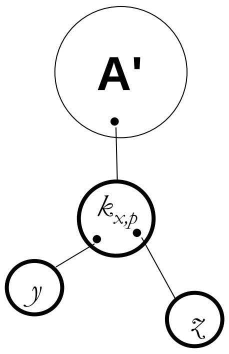

Figure 19: Recognition of node uniqueness.

This leaves no doubt about **y** and **z** being properties of **x** *within a specific context*.
The mechanism of context orientation can be repeated indefinitely. It is specifically not as overall objects that **y** and **z** make up the intext of **x** in its capacity of $\mathbf{k_{x,p}}$. In turn, they will also be nodes ($\mathbf{k_{y,q}}$ and $\mathbf{k_{z,r}}$— see Figure 20).

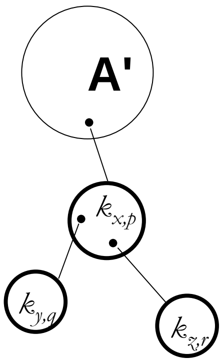

Figure 20: Universal uniqueness of nodes.

In the above figures, the relationship with the nil node of the overall object is not shown. It is presupposed where applicable.

Precision versus ambiguity

Starting at a particular node, the context is always an unambiguously-ordered collection of relationships and (other) nodes. The intext, however, is often ambiguous. Each node usually has various other nodes as its "properties." Figure 21 presents a series in which the point of view changes with different nodes. Therefore, what constitutes context and intext changes accordingly.

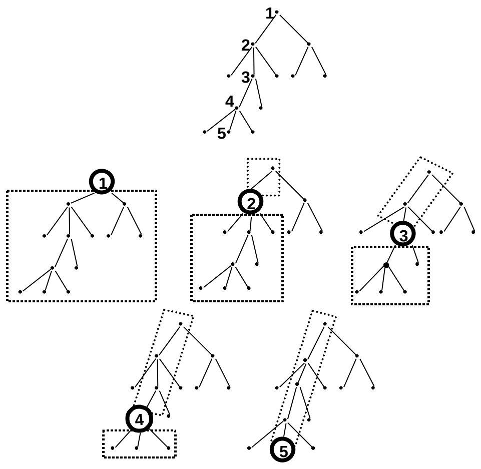

Figure 21: Relative nature of context and intext.

In this series, the nil object is not documented. Actually, the complete information set/system could be defined as the intext of the nil object.

The Figure above really captures the central idea of metapattern. Any view of information is dependent on a particular *point* of view. As point of view changes, so do the contents of context and intext. Metapattern can therefore be considered an exhaustive procedure to assign a unique identity to every single point of view.

As metapattern is primarily concerned with conceptual information modeling, i.e. ontological engineering, "point of view" means how differently real-world object behaviors may be viewed. With each and every (possible) point of view uniquely accessible, precision of behavior differentiation is optimized. How these conceptual models are translated onto construction specifications for software engineering lies beyond the scope of this article. Of course, metapattern’s first priority of conceptual understanding serves as the best guarantee to draw up effective construction models. Although metapattern is not designed for digital construction modeling, as it is aimed at realizing practical information systems many of metapattern's essential concepts have direct relatives at construction and implementation. However straightforward, an explicit translation from ontological to software engineering is always in order. For translation purposes, an approach specifically oriented to construction/implementation modeling needs alignment with metapattern.

Primitive information objects

The (whole) information set never solely consists of nodes and their context-oriented identifiers. Although this nodal network is in itself vital information, sooner or later properties must be made known through a “primitive intext” in which all corresponding information objects are directly managed by their immediately-superior node.

Suppose a person is newly-registered. Suppose, too, that the corresponding last name must be entered *in the context of* person. In such a case, the last name is part of the primitive intext. The primitive information object is shown in Figure 22 as a rectangle.

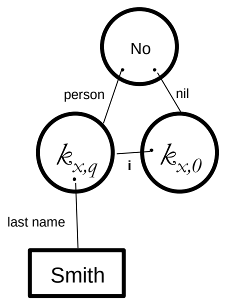

Figure 22: Primitive information.

The last name can be understood differently—as, for example, a component of the complete name. Another component would then be the first given name. Such a complete name is an example of a “composition,” always an object in its own right. In the context of node-as-person, the composed name is represented by another node. Only through relationships for given name and last name are primitive information objects introduced.

The nil object below is abstracted into a solid, horizontal line from which all relationships and (other) information objects “hang down.” The nil object is the preconditioned ground of any information model. For practical reasons, this ground is always drawn on top, as in Figure 23.

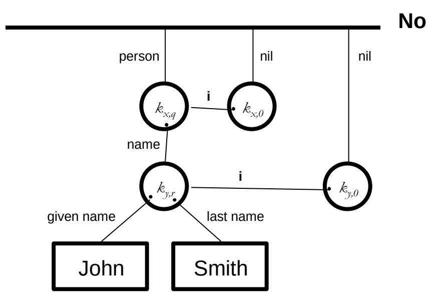

Figure 23: Abstraction of the nil object into a ground.

Whether a particular composition is useful or necessary—or not—depends on information requirements. In Figure 24, the complexities and possibilities are somewhat reduced.

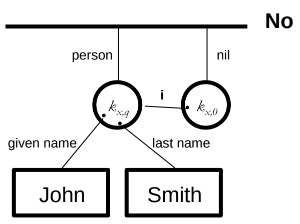

Figure 24: A simpler alternative.

An information object may act as part of a primitive intext. When this occurs it always has a certain context, but never any intext. Its context minimally consists of one node identifying another object.

Pointer information objects

Now suppose that country of birth must also be registered. There are not that many countries; more important, it is simpler to refer to, or, in more direct terms, point at an instance in the country set. Within the context of a particular person, country of birth does not appear as a primitive information object. Rather, it is a pointer information object. The "value" of the reference is the node in the relevant (other) context. In all Figures beginning with 25, a circle-within-a-rectangle is used as the symbol for a pointer information object. Of course, this mention of pointers must not be confused with implementation, as in computer programming. Pointers here are meant as purely conceptual devices. Metapattern favors showing such a reference explicitly as an object-in-intext, even though a mere relationship would convey the same information. Overview is simplified by objectifying the reference/pointer. It also makes it easier to shift from a pointer information object to an intermediary information object (see below) and vice versa.

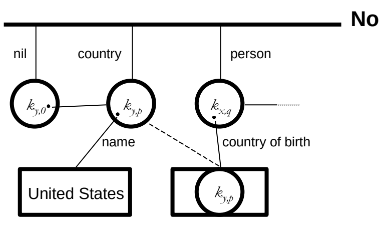

Figure 25: Pointer information for reference.

Having a pointer information object in the intext amounts to a direct relationship, shown here with a dotted line, between two identifier nodes of the same overall object. They already maintain an indirect relationship, by definition, through their mutual nil identity. The circle within the rectangle symbolizes the relational aspect on which a pointer is based.

Like the primitive information objects, pointer information objects do not contain intext. The rectangle indicates that such pointers simultaneously reflect a "primitive" aspect.

Intermediary information objects

The power of the context-oriented approach to information modeling
lies in the assignment to an object (i.e., an object's part) of a specific-but-rounded identity corresponding to a specific context. This requires a third kind of building block, the intermediary information objects.

Let's say a company specifies a person as its customer and registers additional information about the person *as a customer*. For this purpose, an unambiguous node is created (Figure 26).

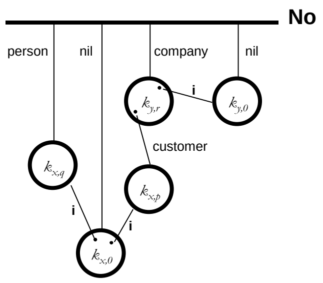

Figure 26: Reference at intermediary node: coordination through nil identity.

The customer's intext, with the customer being identified by the node $\mathbf{k}_{x,p}$, may consist of primitive, pointer and/or intermediary information objects.

Let's assume that, despite the specific customer context, direct information covered by the person context is needed. Under such requirements, the node $\mathbf{k}_{x,p}$ should be supplied in its intext with a pointer information object. In addition to their indirect relationship, (i.e., through a joint nil identity), a direct relationship exists between the object-as-person and the object-as-customer-of-company-**y**. This is convenient for presenting general information about the customer; such information is taken from the (general) information about the "corresponding" person. Figure 27, by the way, leaves out some details contained in Figure 26.

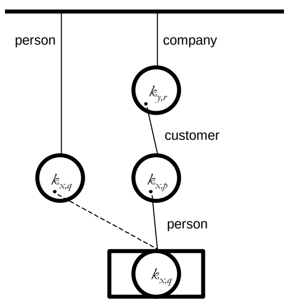

Figure 27: Direct coordination of partial identities.

Intermediary character of context

As stated earlier, from the point of view of a certain node, its intext has a pluriform character. Information objects in a node’s intext may be primitive, pointers and/or intermediary.

The context, however, is of a uniform nature—a consequence of the rule that primitive and pointer information objects do *not* themselves have any intext; they are not so much nodes as final destinations. It follows that only an intermediary information object can lead to another intermediary information object. Therefore, the uniform, all-intermediary character of the context of *any* information object amounts to a series of (unique) nodes. The first paragraph 1 gives the clearest description of this character, treating all objects purely within their respective contexts.

Range for object identity

The nil identity, as related through the nil relationship with the nil object, leads to a radical conclusion—it is now possible, apart from the nil object, to model the complete information set/system on the basis of a single overall object. With specific contexts the meaning of specific behavior is always guaranteed, as shown in Figure 28.

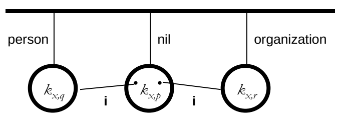

Figure 28: Radical differentiation through contexts, alone.

This seems like the wrong idea for most information sets. Traditional object orientation poses opposite problems because a certain object type is often too limited. When more detailed types no longer conform to a neat, manageable hierarchy, the classification fails to function properly. Against the background of metapattern, the cause of such failure becomes evident: contexts with no modeling or only minimally/ implicitly modeled.

Metapattern forces no such limitation on the modeler and the modeling approach. Contexts are always explicit and juxtaposed. (Below, the difference between OO subtyping and metapattern *juxtatyping* will become clearer.) Only the context consists of the nil relationship to the nil object separately defined as a boundary condition. Being typeless has now been given a (positive) value.

Metapattern allows a wide range of model alternatives. At one end is the radical possibility of a single overall object; at the other, as many overall objects may be modeled as there are separately identifiable contexts. Within these extremes, the proper balance must be modeled in light of prevailing information requirements.

Being aware of what the context-oriented approach tries to accomplish is helpful. Its goal is the simultaneous treatment of differentiation *and* integration. An object is split according to contexts but, at the same time, the overall object continues to exist comprehensively. A balanced demarcation into an overall object is achieved when behaviors can be both meaningfully differentiated (through contexts) and meaningfully integrated (through the joint nil identity). Given this criterion, the modeler must base choices for overall objects on their integrated responsibility and accountability. Differences exist—for example, between behaviors as family member, customer, citizen, taxpayer, voter and employee. At the same time, those qualities are appearances of an encompassing "unit." In our society, at least, their respective responsibilities are contained by what is called a (natural) person. It's thus meaningful to specify each appearance as a node in a corresponding context.

In "real life" the person context has a special status. Such a priori, however, needn't be maintained in the information system. There, a person can (and with metapattern available, *should*) be treated as one of many juxtaposed contexts; nil identity provides the generic mechanism for coordination between all parts of an overall object.

In social life, responsibilities are not always similarly ordered. An organization, for example, is held accountable on equal *and* on different terms as compared to a natural person. Another overall object could then be contextualized after organization, tax payer, supplier, employer, customer, and so on.

Context-oriented normalization

For relational information models, a well-known method exists to avoid unwanted duplications. This heuristic is called normalization. Metapattern offers similar opportunities (see below for metapattern's behavioral forms). A primitive information object is registered only once, by definition, but another primitive information object could hold an identical value at the same time. This does not count as duplication, as each object follows its own unique life cycle.

Pointer information objects are also unique, by definition. The same reason holds for intermediary information objects (i.e., nodes for object identification). What makes them unique is the specific context. Both pointer and intermediary information objects contribute fundamentally to constraining, and even eliminating, information duplication.

Please note that metapattern focuses on conceptual considerations. Such terms often reappear at the stage of construction modeling, but they carry different meanings and consequences in that different context.

3. Types

To provide a sharp outline of concepts for context and intext, discussion of metapattern has thus far been restricted to specific relationships and objects. However, to support a somewhat complex specific structure, another structure is needed on a higher level of abstraction. This paragraph, which sketches that addition by referring to types, takes an inverse approach to highlight metapattern's characteristics. Traditional object orientation takes the class or type as the starting point; a specific object is then exclusively regarded as a member of the type-defined set/class. For metapattern, however, the various contexts of objects occupy place of principle.

Node as type

A specific object, in a specific context, can and will behave specifically. Within the complete information set, this level of behavioral specification reflects one extreme on the range for typing. The other extreme reflects identical behavior for all objects, all objects being identical.

At the extreme possibility, mentioned earlier, the indicator of the specific type *is* the specific node as a unique identifier. Type($\mathbf{k}_{x,p}$) = $\mathbf{k}_{x,p}$. Thus the node $\mathbf{k}_{x,p}$ is uniquely typed. Such a type is described not only by the relationships in the particular context but *also* by the particular nodes higher up the hierarchy of inspection levels. Figure 29 shows a case in which the same person is an employee with two different organizations.

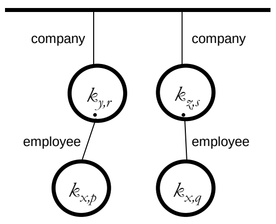

Figure 29: Unique behavior at node.

Indeed, when the node itself is also considered a type, the behavior of employee **x** in company **y** can be "typed" differently from the same person **x**'s behavior in company **z**. There are, in fact, two different employees, both of them "grounded" in the same person. With each node a completely singular type, even their existence as employees has no consequence for behavior.

Context as type

It is already less extreme to consider the context of a node—not the node itself—as the relevant type, since several nodes may share the same context. In this case the type is indicated just like the context. From Context($\mathbf{k}_{x,p}$) = {**a**, $\mathbf{k}_{y,q}$, Context($\mathbf{k}_{y,q}$)}, it follows that Type($\mathbf{k}_{x,p}$) = {**a**, $\mathbf{k}_{y,q}$, Type($\mathbf{k}_{y,q}$)}. This agrees with the remark made earlier, i..e. the context is an ordered collection of relationships and objects (nodes). As such, context orientation is equivalent to a hierarchical classification structure.

At this point, perhaps once again one of the main differences between metapattern and most traditional approaches to conceptual modeling should be stressed. Insofar as other approaches apply the concept of context, they appear to do so from a macro perspective in which any context itself is left unspecified. This serves as a single framing device. Within such a macro-context, a complete (but now relative) information model develops. This macro perspective, when present, offers little precision. It's often too little for differentiating behavior.

As a contrast, metapattern's micro perspective allows real-world differences in behavior to be specified with ultimate precision. After all, a particular context does not remain outside an information model (which, by applying such a procedure, is always relative as a whole). Context according to metapattern is a variable to be *valued within* conceptual information models. It leads to the same information model to be able to support a great variety of contexts or, rather, context instances. As shown in the previous paragraph, behavior may even be typed at the extreme of singular nodes, the equivalent of each node having a singular context. More precision in differentiating behavior cannot possibly be supported.

Everything depends, then, on how contexts are *developed* using object-and-relationship pairs to describe them. What is optimal should reflect relevant information requirements and their underlying problem. At the conceptual modeling stage, the life as opposed to tool focus needs to be applied. With new problems occurring, developing conceptual models is very much a creative process. No definitive recipe can be assumed to arrive at the overall modeling result. Most of life's problems cannot be solved mechanically. But the modeler may already feel familiar with some parts of the problem, seeing opportunities for reuse of conceptual models. A more or less stable conceptual solution may then be called a pattern, but even then a modeler should always question its stability (perhaps an established pattern does not really serve a particular new requirement). With microscopic possibilities for contexts, existing patterns are easily modified or entirely new patterns developed.

Typical generalization

The first mentioned extreme in typing—that of the individual node itself—should not be considered a purely theoretical boundary case (that is, a mere rarity). Sometimes a real, practical need exists for such variety.

Of course, the relevant types are often more general in nature. Take again the example given in Figure 29. Suppose it matters which employee and which company are involved; an employee's behavior follows the same model everywhere. The type then is still determined by both relationships; that is, by the ordered collection {**employee**, **company**}. When the upper node is also irrelevant (i.e. it doesn't matter whether the employee works for a company, a foundation or a government agency), the type is further generalized to **employee**.

Through subsets of a context, metapattern offers the opportunity for highly differentiated typing. In the next subparagraph (relational typing), the generalization is restricted to the discrete relationships, resulting in fairly abstract types. However, this already represents an extension of traditional object orientation. An even more elaborate typing schema would include individual nodes. A subparagraph on (node selection), a concise explanation is offered on dealing with nodes in typing, resulting in correspondingly differentiated behavior. Even more concisely, yet another subparagraph (relationship selection) states that a relationship can also serve to localize a type (not as a discrete unit). As such, a relationship does not act as a discrete unit but supports selection on the basis of certain criteria.

Relational typing

Relationships between objects in any information set should, at a more basic level, also be counted as information objects. This opens the opportunity to position them in a relevant context which is, in turn, oriented toward relational typing.

Referring again to Figure 29, three possibilities exist for *relationally* typing the behavior of employees in companies. First, both relationships can matter. Note that their order in the context is important; type indicators are gathered from the original information objects in their contextual and intextual structure.

In Figure 30, nodes are no longer recognizable as instances. For intermediary information objects, the “aggregated” node is supplied with a short description to its right or left. As introduced earlier as a schematic device, the nil object is no longer shown as a separate (boundary) node; instead, it appears as a thick, solid line at the top of illustrations. Type details are not specified in illustrations; the idea is to present a fundamental description of metapattern. At the other extreme of the type relationship, primitive, pointer and/or intermediary information objects may be used to specify a certain type. Instances of unspecified intext are indicated by a text balloon; a dotted line is used to suggest the existence of intext. No indicators are used if the possibility of confusion doesn’t exit.

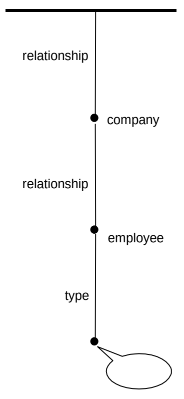

Figure 30: All relationships determining type.

As a second alternative for a type (still assuming the concrete information structure of Figure 29), only the “lowest” relationship may be needed to differentiate the objects’ behavior. This is shown in Figure 31.

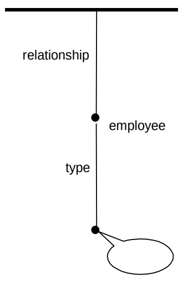

Figure 31: Type determined by subset of relationships from context.

The third alternative to typing takes the other extreme. A general type for *all* objects is assumed; it is relationship-less—that is, it has no reference to relationships in the structure of original information objects. Figure 32 depicts this ultimate abstraction:

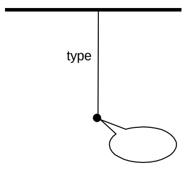

Figure 32: Ultimate abstraction in typing.

A general type such as this can also be used for company behavior. If a more specific type for company is needed, there is but one other possibility. Shown in Figure 33, it is based on Figure 29.

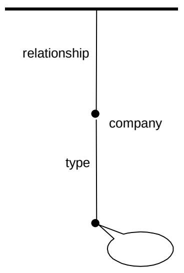

Figure 33: Company type.

Because the types for company and employee are split, Figures 30 and 33 may be brought together in Figure 34.

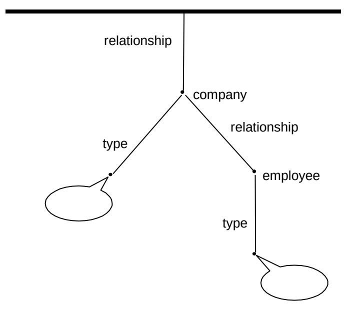

Figure 34: Integrating relationship-based types into an overall model.

When relationships themselves are related, as in Figures 30 and 34, some information about types is revealed. The interpretation, according to strictly relational typing: all nodes occupying a relationship with the nil object (in a context capacity) may entertain in their intexts an employee relationship with information objects. The more detailed behavioral rules for such information objects are then specified under the type for relationship.

To retrieve applicable behavior rules from the context, as many relationships as possible should match the relational pattern which serves as a type context. To obtain a straightforward matching procedure, it's reasonable to proceed from the bottom up—the "direction" in the illustrations. The type in Figure 31 applies only when the type in Figure 30 does not exist.

Care must be taken not to interpret the suggestions for type retrieval as direct construction specifications. They are supplied here to support unambiguous interpretation at the stage of *conceptual* modeling. Of course, not only separate information systems but even a tool technology may be developed to incorporate the metapattern perspective on reality. The contents of this paragraph may then be taken as general hints, nothing more or less, on how to deal technologically with digital information objects at the (first) abstraction level of typing.

Node selection

Relational typing may be extended by adding nodes which govern selection. That is, such nodes contain a prescription for defining a subset holding elements with similar behavior. Not all companies, for example, are required to make their annual accounts public; only companies with statutory limited liability are. In this way, relational typing may be enhanced, yielding a larger number of types. Please note that those are not exactly subtypes as in traditional object orientation; more variety is supported.

These prescriptive nodes for type determination (see Figure 35) must be placed between two other nodes representing a contextual relationship in a specific structure—a structure of instances, not types. A number of nodes and selection procedure outcomes must be included that reflect the type-based behavioral variety to be modeled. Each selection node has an intext where criteria are stated. If the node—that is, the node elsewhere in the specific information structure—complies with a given set of criteria, the next step in assigning type is determined. Without sufficient validation at the time of type definition there is always a risk that matching context with type does not yield disjunctive results. In such a case, a specific information object's type is indeterminate.

At this point, matching concerns the original node together with its complete context on the one side, and something that can be called a type intext on the other. As suggested earlier, type intext contains matching rules from the bottom to the top of the node-plus-context. However, that context was originally specified top-down. The ordered collection that *is* the context must thus be adjusted for the matching procedure to work properly.

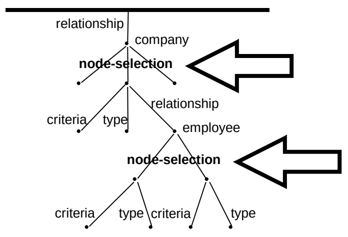

Figure 35: Typing beyond relationships in context.

Suppose, as in Figure 35, five relevant subsets exist; three refer to companies, the others to employees. The mechanism for node selection can be structured as shown in the next illustration. For the sake of overview, the information object for the company relationship is given just one more detailed intext for node selection.

Figure 35 should only be considered as an example, as differences often exist between what is relevant to the typing of company and employee behavior. The behavior of employees might need a different classification of companies than the behavior of the companies themselves. When this occurs, node selections must be differentiated accordingly.

A special case of node selection consists of the complete absence of any criterion. Every node in the specific, original information structure complies. This most-general type can be used as a default to implement the more specific approach of relational typing. The same holds for node-specific types and/or for any combination.

On the other hand, a specific node may define its own, unique type. In such cases it would be cumbersome to create a parallel structure for the type. It is more efficient to place the typing information directly in the intext of the particular node. But such an implementation requires an additional mechanism to include such uniquely-localized typing information in general reports on types.

Relationship selection

So far, the relationships for relational typing have been considered as discrete instances. Additional differentiation is possible (top to bottom, operating like node selection) by applying relationship selection in order to arrive at the generalized type of a specific node. This mechanism also allows all types to be catered for in a general structure. In this type-oriented intext of the nil object, information about relationship and node selections are interchanged until a match is reached with (part of) the original node and its intext.

Tailor-made typing

A central assumption of metapattern is this: contexts must be modeled, each context having a structure consisting of relationships between objects. As this paragraph explains, context-equals-classification offers a confusing array of opportunities for type variety. The simplest option is to consider the bottom-most relationship of a context as the node type. This amounts to the typing paradigm of traditional object orientation.

By contrast, metapattern offers a wide range of typing possibilities, up to defining a specific node as its own, unique type. This variety is supported because types, modeled after the metapattern approach, are both integral and integrating elements of the complete information set. In its turn, each specific type is also modeled as an object-in-context containing a correspondingly specific intext. At such a “typical” node, the intext describes and prescribes, among other aspects, the other objects-in-context for which it acts as type.

The typing information is actually limited to the highest level of that other node’s intext. No more is required, as all intermediary information objects at the highest intext level are typed themselves. This continues until the lowest-level nodes. Metapattern implies recurrence of typing at all inspection levels of the information structure.

The way in which types are described in technological detail is not within the scope of this article. Such details would only detract from an understanding of how metapattern functions in conceptual information modeling.

A different inheritance

The context-oriented approach eliminates the need for inheritance of behavioral rules for different appearances (or identities) which constitute an overall object. In various contexts, the same overall object “owns” a corresponding number of partial identities; each has been developed on the basis of a disjunct type, and takes its behavior from those contextual types. Bill at work, Bill at the gym, etc. The conclusion: a certain type is *not* a subtype of any other. Types for different identities within a single overall object are fundamentally juxtaposed, as are their contexts (see Figure 36).

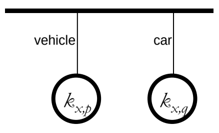

Figure 36: Juxtaposed types.

Or, resembling subtyping, one context can be part of another, as shown in Figure 37.

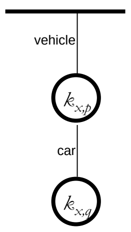

Figure 37: Types through recursion of context.

The right model depends on the particular information requirements. For instance, car behavior is additional but *completely disjunct* to vehicle behavior.

Metapattern does provide for some sort of inheritance. It generally understands "inheritance" to mean that a particular node may implicitly avail of any properties (intext) of any node from its context; that is, from nodes at higher levels in the modeled hierarchy. Suppose that (1) an information object exists within a context; (2) another information object is part of that context; and, (3) an address is specified in the intext of that other information object. That address could be declared applicable for all information objects emanating at all lower levels from the information object that actually contains it as a distinct property. In the example below, all employees inherit the address from the respective organizations of which they are a "property." The possibility for an employee to have a different address is maintained. An instance at a lower level blocks the procedure to look at a higher level for an address to inherit, a mechanism underlying Figure 38.

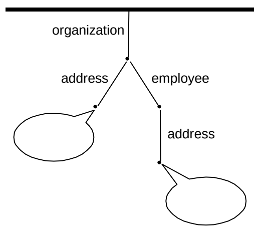

Figure 38: Instance-level, rather than type-level inheritance.

Each hierarchy invites such a mechanism to aggregate its domain of applicability (not information itself). Inheritance is a powerful mechanism for avoiding, and sometimes even eliminating, duplicate information. Through context orientation, metapattern offers an intuitively natural aggregation strategy.[3]

As types are also modeled using metapattern, a certain kind of subtype is (re)introduced. The difference is a consequence of the disjunct character of contexts. Where types are layered, their structure reflects the layering of the original, specific information (with information objects at different inspection levels). This corresponds to the equivalence of context (or parts of contexts) with types. For example, because both organization and employee reserve a place for address in their respective intexts, *in that sense* an employee may be considered a subtype of organization.

What type **a** inherits as a property (or behavior) from type **b**, an information object of type **a** inherits as behavioral rules (or properties) from the related information object of type **b**.

As mentioned earlier, types are represented by discrete information objects. Those type objects also have types, as metapattern allows for explicit metatyping—an additional reason to call the modeling approach *meta*pattern. A single approach is supposed to encompass all semantic levels at which information is structured. For practical purposes, a difference is assumed between (1) specific information and (2) information controlling the behavior of that (other) specific information. It is customary to classify the former as instances and the latter as types. Another familiar term indicating the latter is meta-information. But the concrete expression of a particular type is also grounded in specific information. The next-level type is then called the metatype. While such abstraction can proceed indefinitely, it invites confusion.

Always starting from specific information, whatever it may represent, metapattern only recognizes its meta-information directly as a type. But in its own right, every type can also be seen *as specific information*, leading to its own particular meta-information. By changing the point of view, any abstraction/aggregation of behavioral structure can be achieved as a chain of interchanging information and meta-information.

Strong polymorphism

Context orientation makes type inheritance almost disappear. As a complement, polymorphism greatly increases. The individual contexts, and consequently the contextual types, assure precision within the variety. What is known by a general name always follows the rules specified within the relevant context. Bill may, for example, write both at home and at work, but there are most likely very different behaviors involved.

It must be emphasized again that an overall object does *not* have a single, overall type. Types are “limited” to the overall object’s contextual identities; that is, to its parts, which own a unique identity within a particular context. As a corollary, parts of otherwise different overall objects can share a type. Their equally typical differences show in the types they do not share for their other identities/parts.

Suppose it does not matter whether a customer is a person or an organization. That is, the required properties are always similar—the structure of all customer intexts is identical. Under those circumstances, person and organization share the customer type for one of their partial identities. Those nodes lead to their respective nil identities and to identities belonging to different types. This flexibility represents a major benefit of metapattern. Figure 39 sketches the example, with nodes as instances of specific information objects.

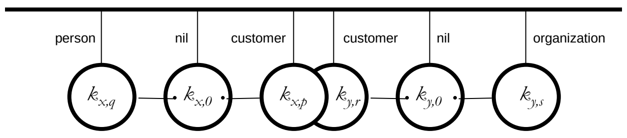

Figure 39: Mixing partial identities across contexts.

Behavioral forms and encapsulation

Analogous, at least in terminology, to normal forms in relational information sets, behavioral forms exist for the context-oriented approach to information modeling. Behavioral forms mainly control the extent of encapsulation. Like the normal forms that inspired the concept of behavioral forms, even the latter show an implementation bias. Metapattern is the next step in the development toward decreasing that bias, emphasizing conceptual treatment of information requirements still more.

Every higher behavioral form presupposes all lower forms. Information in second behavioral form is, therefore, also in first behavioral form, etcetera. Definitions for the five identified behavioral forms are: [4]

1. Information is formally in *first behavioral form* when the pertaining information object has the opportunity for unique behavior. Every object, therefore, is uniquely ... itself. Or, put in yet another way, every object defines in itself always one class. For it is always a class in itself. This uniqueness condition is necessary to support diversity of behavior.
2. Beyond first behavioral form, *second behavioral form* stipulates that the object's unique identification is totally independent of any external information requirements. In other words, the object's key is entirely non-value based.
3. Every object, except the complete information base, is positioned within a context. And that determines *third behavioral form*.
4. By turning the words around used to typify third behavioral form, the *fourth behavioral form* comes into focus. Of course, first of all, the fourth presupposes the third behavioral form. Then, in addition, an object is not only familiar with all its contexts but also encapsulates them. All relevant contexts are contained within an object in fourth behavioral form. Remark: The corresponding mechanism is the unique node allowing part of an overall object to "own" an identity within a particular context. All other nodes belonging to the same overall object maintain an identity relationship with its nil identity. The nil identity is not yet explicitly specified by the fourth behavioral form.
5. Any object that is in fourth behavioral form and, in addition, limits encapsulation of intext to the next lower order is in what I define as *fifth behavioral form*.

4. Time

Context, in an ontological sense, precedes object. This idea—the main tenet of metapattern—determines how a modeler (also read: ontological engineer) who follows the metapattern approach presumes reality to be fundamentally structured.

Practically, context provides an overall object with different identities for its contextual parts (also called partial identities). Indeed, the number of its separate partial identities equals the number of contexts in which an overall object appears or becomes manifest through its corresponding parts. Every contextual/partial identity of a real-world, overall object is conceptually *represented* by an information object. In conceptual information models, an information object in its capacity of individual identity equals a unique (intermediary) node.

The contextual differentiation also supplies an excellent starting point to deal fundamentally with the aspect of time in conceptual information models. A uniform, compact treatment rests on the recognition of the variable nature of real-world objects, and thus of the variability of their corresponding information objects. For metapattern, change rather than continuity guides the life of information. Change, however, always implies the possibility of continuity—defined as the absence of change during a particular period of time. Continuity is change with zero value (which is nonetheless a value). In general, practical success with metapattern can be significantly and consistently improved by abstractions such as those applied here to modeling pervasive time management of information.

Time-based relationships

Suppose nodes $\mathbf{k_{y,q}}$ and $\mathbf{k_{x,p}}$ are connected (Figure 40) through relationship **a**, with $\mathbf{k_{y,q}}$ being an element of the context of $\mathbf{k_{x,p}}$.

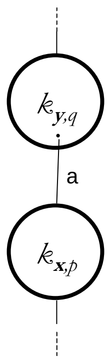

Figure 40: A relationship of nodes.

A consistent management of time first requires that the existence of a relationship between nodes, whatever its nature, has been established. Figure 41 shows how an upper and lower node together constitute a unique relationship.

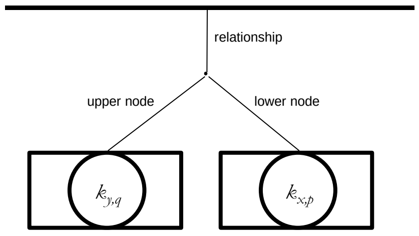

Figure 41: Relationship defined through nodes.

As properties of such a relationship, one or more existence states must be included. Their cardinality is specified alongside the node representing the set of states within the relationship context.

A single existence state contains a value for the existence mode and a time value (including date) for which the given existence value holds (Figure 42). Figure 41 is integrated, but its details are not repeated. Actually, the choice of *starting* times is not fundamental. Consistent application of final times delivers the same effect.

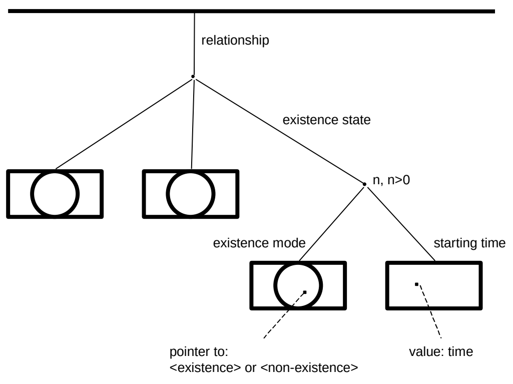

Figure 42: Information on relationship existence.

Starting time of effect

When every change is registered as a state, including the time from which that state takes effect, then state at *any* time can be derived. Suppose that $\mathbf{t_1}$ is the starting time for the state of existence. When the existence mode is changed at time $\mathbf{t_2}$ to the state of non-existence, the state of existence is limited to the period between $\mathbf{t_1}$ and $\mathbf{t_2}$. Implicitly, the state of non-existence holds for the whole period preceding $\mathbf{t_1}$. Figure 43 tries to summarize this time-based mechanism.

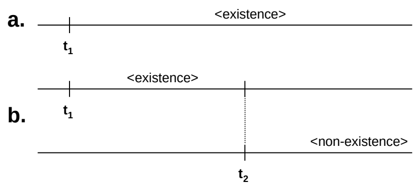

Figure 43: Succession of states.

As a benefit, this approach allows interchange between states of existence/non-existence. The state of existence is no longer restrained to a single, uninterrupted time period enclosed by two periods for which the state of non-existence is implicitly valid.

A consequence of reading time (or information about time) in the proposed direction is that the state of non-existence is not explicitly registered. It must be derived from the earliest starting time to appear in an existence state holding the value **existence**.

From existence to value

Whenever *any* relationship exists it is necessary to focus on its *value*, which is at least an information object in the intext of the node representing the relationship itself.

There are many ways to model that part of its intext. Here, priority is given to changes in value; that is, to value entries. In its turn, each value entry has an intext consisting of a pointer to the value of the relationship and, again, one or more existence entries. Figure 44 concentrates on this mechanism by leaving out any details from Figure 42.

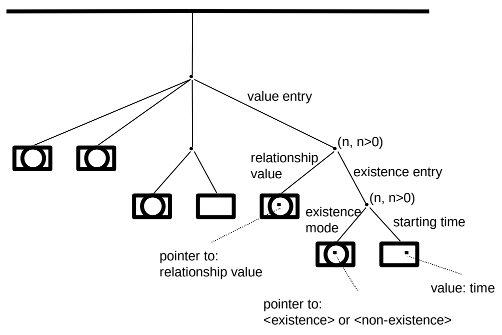

Figure 44: Relationship values in time.

Using this model, or any similar model, the relationship between $\mathbf{k_{y,q}}$ and $\mathbf{k_{x,p}}$ can hold value **a** from time $\mathbf{t_1}$, value **b** from time $\mathbf{t_2}$, from time $\mathbf{t_3}$ possibly again value **a**, and so on. Actually, the registration of relationship values and their non-existence is not dependent on registration of the relationship itself, regardless of its value. This poses no problem as long as the validity check on the relationship is given priority above checking the state of value.

By changing the value of the relationship, context changes (context $\mathbf{k_{x,p}}$ in the example). This raises the question of whether, at the same time, type $\mathbf{k_{x,p}}$ changes. Indeed, this could happen and would then constitute dynamic typing.

Time-based information objects

Any intermediary information object—a common node in a conceptual model—needs only one or more existence entries to establish its position along the time dimension. Such nodes, as shown in Figure 45, do not “contain” any value themselves (but may “lead” directly to such a value, either in its own intext or in an intermediary manner, through a pointer information object in its intext; or indirectly via its nil identity).

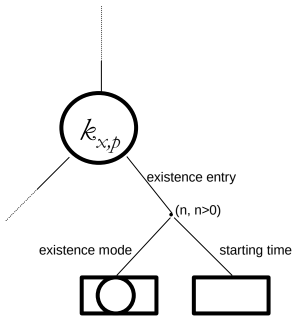

Figure 45: Intermediary nodes are essentially value-less.

The second paragraph defines primitive and pointer information objects as having no intext. An exception: existence entries. Figure 45 applies for those, substituting a primitive or a pointer information object for the node (i.e., the intermediary information object).

The rule stating that primitive and pointer information objects *do* have existence entries for their intexts is cancelled, for practicality, for information objects constituting those existence entries themselves. These are (1) the pointer information object referring to the value **existence** or **non-existence**; and, (2) the primitive information object containing a value for starting time. This new rule for the presence/ absence of intext is necessary; the previous rule, applied to existence objects, would lead to an infinite regression.

One might erroneously conclude from this an inconsistent metapattern application and a focus switch from life to tool. Indeed, an important part of the reality being conceptually modeled is now the information set/system to be constructed and operationally used, rather than the reality outside it. But there really is no conflict with the essential life focus of conceptual modeling. The tool, too, will eventually be integrated into life. It is thus logical that operational aspects need conceptual integration. A conceptual model without consideration of all relevant reality would be lacking in precision. It would be a mistake to view time management of information objects as a predominantly technological issue. It needs a strong conceptual foundation. The same holds for establishing an audit trail, conceptually described below.

Variations in time

All changes in the information set can be reduced to procedures outlined in the previous paragraphs. Suppose that, starting from time $\mathbf{t_2}$, node $\mathbf{k_{x,p}}$ is no longer connected to $\mathbf{k_{y,q}}$ through relationship **a** but to $\mathbf{k_{z,r}}$ through relationship **b**. This amounts, first, to an existence entry against the relationship between $\mathbf{k_{y,q}}$ and $\mathbf{k_{x,p}}$. This new entry contains a pointer to **non-existence** as the value for existence mode and the value $\mathbf{t_2}$ as starting time. The relationship value **a** no longer deserves attention, as such values depend on the existence of the relationship itself.

Secondly, a relationship between $\mathbf{k_{z,r}}$ and $\mathbf{k_{x,p}}$ must be established (assuming it does not exist). An existence entry is immediately added with a pointer to **existence** as the value of existence mode and $\mathbf{t_2}$ as the starting time. To the intext of the new relationship a value entry is added specifying that from time $\mathbf{t_2}$ the relationship value is **b**.

When the relationship between $\mathbf{k_{z,r}}$ and $\mathbf{k_{x,p}}$ is already available in the information set, an additional existence entry is sufficient (assuming that the value of the relationship was previously also **b**).

Such operations changing the structure of information are especially type-sensitive. As shown above, however, there is no fundamental problem with metapattern. Specifying types does reflect the additional variety which can be controlled.

The finely-tuned time management guarantees that normalization remains valid along the time dimension. A pointer information object, for example, will use a particular, user-defined *time of* *relevance* to direct its follow-up. The object pointed to “answers” by supplying the value of the primitive information object as valid at the requested time of relevance.

Another feature of metapattern allows a node to be moved from one context to another—an extremely powerful ability because the move does not affect the node’s intext.

Suppose the subsidiary relationship is changed for company **x**. It reports to **division 2** instead of **division 1**. The changes in the information set are limited to that particular operation. From the given starting time—assuming that all company **x** employees were originally registered in the intext of **x**-in-the-context-of-**division-1**—all employees are members of **division 2**, not **division 1**. By specifying a time of relevance before the time of reorganization, the original “picture” is shown. See Figure 46.

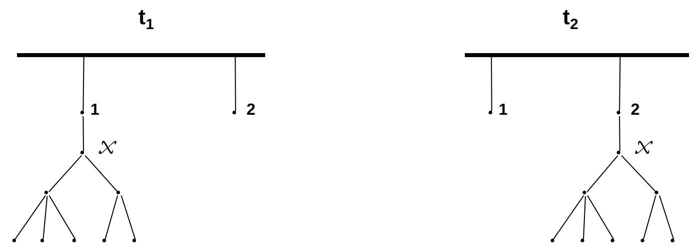

Figure 46: The power of a single relationship change.

To guarantee precision at each intermediary node, metapattern rules that at any point in time one context—and only one—is valid.

A change of nil identity

It is also intriguingly powerful that a particular node can, in the course of time, change its relationship to a nil identity. By exchanging nil identities, *its* nil identity may be different from a certain time. As the node always reflects an identity of an overall object in a particular context, changing its nil identity means that the contextual identity has become part of *another* overall object. Figure 47 shows the mechanism of this highly adaptive feature.

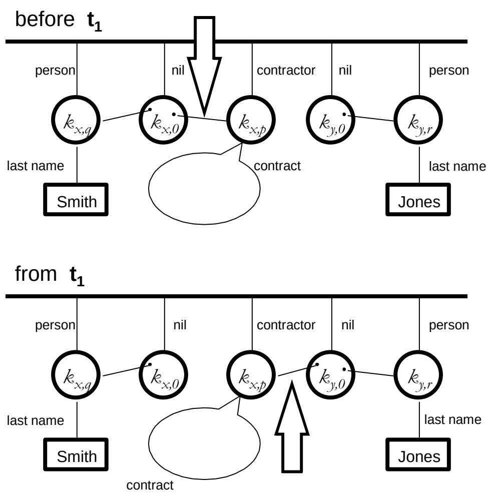

Figure 47: A partial identity may change to another overall object.

The possibility for a node to change nil identity reveals that even the nodal suffixes **x** and **y** could be misleading. In the example given, from time $\mathbf{t_1}$, node $\mathbf{k_{x,p}}$ does not belong to the overall object **x** but to the overall object **y**. Thus, all common nodes should be counted using only a single suffix, as shown in Figure48. Only the special nodes for nil identities are supplied with a second suffix fitted with the constant value of zero.

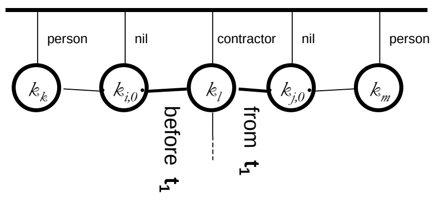

Figure 48: Uniqueness over time requires abstraction from particular overall object.

Changing a contextual identity from one overall object to another need not be limited to overall objects which share type for all identities. The new overall object may just as well count organization instead of person as type for one of its partial identities. Particularly when corrections are necessary, changing a nil identity offers a powerful mechanism to keep the information set realistic. An unambiguous audit trail needs additional entries available to specify which information object is valid/invalid from which time. (Note: validity mode is closely related to, but still different from, existence mode.)

Audit trail

Strategies guaranteeing all possible support for accountability are based on the integration of the time dimension into information models. A life focus offers strong reasons for adding fundamental mechanisms to provide easily-accessible, transparent audit trails.

The information system/set requires, as a structural facility, the option to reconstruct its user state at any point in time. By taking change as the rule rather than exception, a firm foundation for accountability has already been laid. Still lacking is an operation reporting that certain information once considered valid has been subsequently declared invalid (for whatever reason) and corrected.

However, when new information replaces old, thereby making the old information disappear, accountability is compromised. The radical but only viable solution is to keep all information available as a matter of principle. To introduce exactness and completeness into the audit trail, the once-valid-turned-invalid information must be labeled as such and maintained.

Just as a particular relationship or information object exists or non-exists, a difference can be made between states of validity and non-validity, as shown in Figure 49.

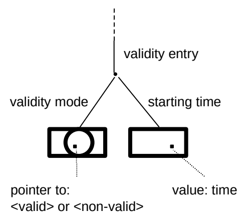

Figure 49: Support of comprehensive audit trail.

By perceiving the state of validity as a change applicable from a particular point in time, those periods during which pertaining information is either valid or invalid can interchange. For information to be reconstructed from the perspective of (in)validity, the registration of original information must include a validity entry (completely analogous to the required existence entry). Any validity entry must contain its time of registration. The validity mode of entry with the last time of registration determines the operational user state of the information set at the requested time of relevance.

All information objects have, as part of their intext, one or more validity entries. However, an exception is specified. Only information objects in the intext of a validity entry do not themselves include validity entries. Otherwise, an infinite regression would result. Instead, when necessary, one entry succeeds the next. Through registration times, the audit trail is always precise and unambiguous.

A useful audit trail also leads to the person responsible for registered information. For that reason, each entry, and especially the entries for existence and validity, are “intexted” with a specific pointer information object. The pointer leads to additional information about the person handling the registration. Whether the actual user *really* was the person indicated cannot be completely guaranteed by the information system, but the pointer provides an important clue to accountability.

Besides pointing to the person involved in and/or responsible for the information change, metapattern assumes a pointer to the specific user interaction (i.e., administrative procedure) which controlled the operations effecting the change. Such procedures must be formally typed and implemented, and users should interact with their information system on the basis of such interaction types. Each information change/addition can then be addressed to an interaction instance, providing use and accountability integration. Such natural integration is always the best guarantee for a successful audit trail, requiring no extra effort from users.

Metapattern as infrastructure

By supplying contexts with finely-tuned integrations of the time dimension and accountability, metapattern holds a rich store of standardized facilities. As such, it greatly extends earlier approaches to modeling infrastructure for information processing. As infrastructure can also be viewed as all-that-is-standard, metapattern literally defines a higher, more encompassing standard. Increasing variety in the infrastructure helps minimize the effort needed for customizing information processing. Even more interesting: through customizing, qualitatively different, more powerful information systems can be developed and kept dynamically operational.

Past and future

Because all information objects explicitly include the time at which both their existence proper and their property values (intext) take effect, the time orientation of the information set is no longer limited to the present (i.e., time of registration). Factoring time into periods during which a particular existence mode and validity mode hold allows information to be entered both retrospectively and prospectively. As long as the present time has not caught up with a time indication once viewed as future, the information involved counts as planning. And all information with past starting times are (partly, at least) in an archival state.

All information about past, present and future is, in principle, retained in the information set. To really guarantee an audit trail, such a policy is mandatory. But for wide- and deep-ranging analyses, time-series information must be available. What is elsewhere called data mining considers such analysis as a separate subject. Metapattern favors, conceptually, at least, an integrated perspective.

The principle is to maintain all information, properly labeled, in the set. It should also be possible to remove the information. Metapattern makes a clear distinction between such removal on the one (exceptional) side, and the modes of non-existence and invalidity on the other (normal) side. The latter are automatically catered for when applying metapattern. Removal must always be a separate activity, with special attention paid to minimal requirements—not only for an audit trail, but also for operational analyses.

Conclusion: applying open interconnection

Driven on by developments in technology for open interconnection, conceptual information modeling is increasingly challenging. The need for a conceptually realistic balance in creating, handling and retrieving increasingly complex and often dynamic information is urgent, as the so-called information society will most certainly continue to evolve at an accelerated pace. Modern information systems & services are expected to contribute directly to overall processes. Only an integral strategy for information can reach this objective. Metapattern helps to find a balance in ontological engineering between the extremes of, say, distribution and integration—that is, autonomy and coordination.

Attempts to establish an integral information model based on uniform concepts, when the world-out-there must be characterized by conceptual pluriformity, can only fail. Regretfully, examples of over-optimistic modeling exercises abound. Behind such failures is the mistaken reversal of means and ends. The real *end* should always be a configuration of information services in the pluriformity of reality as experienced during the execution of processes. Information models can only serve as real *means* to such ends, nothing more or less. A viable approach to conceptual information modeling, therefore, must start with the recognition of differences as they *really* exist. It deserves the highest priority. Metapattern starts from the premise of recognizing differences; this is its first principle. Yet, another principle, the need for cohesion, is recognized by metapattern. Both principles are joined by the ideas of a context and an object-in-context. When applying metapattern, contexts are always specified as integral, even integrating, parts of the conceptual information model. As a result, an overall object disappears as an entity in its own right, modeled instead as a flexible, dynamic collection of partial identities tied together to provide cohesion when and where necessary. In this way, metapattern removes the major obstacle to an integral strategy for the informational aspect of complex, dynamic processes. With contexts *inside*, not outside, the conceptual information models, pluriform concepts have acquired an operational, practical status. The combined effect of context and time further guarantees that relevant differences can all be cohesively modeled. This feature extends beyond, say, first order information. For metainformation is also modeled using metapattern. Therefore, even types are dynamic. A particular time of relevance does not lead directly to a selection of information corresponding to a fixed type. The relevant type, too, is determined as it existed, exists or will exist (and is/was registered as valid at that time). The explicit recognition of simultaneous and/or consecutive differences, even at the metalevel(s) of information, makes metapattern eminently suited for gradual development and introduction of related information systems/sets/services. As explained, with existence and validity entries fundamental mechanisms are available to correct information, even retrospectively. The audit trail is never compromised. In a most literal sense, metapattern handles errors and their corrections constructively. It is possible to change information types, it is possible to change information relationships, and it is possible to change primitive information. Through its “forgiveness” to errors, metapattern supports development by trial, thereby avoiding the many pitfalls of the blueprint approach to complex information systems. Cohesion may grow gradually, as insight into the pluriformity of concepts grows. Any unsuccessful trial can easily be controlled by correcting the errors encountered.

Underlying metapattern's potential for integrated information systems is a paradox. By recognizing real differences, and giving them first priority, eventual cohesion is strongly promoted. The strength of metapattern lies in its high uniformity to support pluriformity.

Notes

[1] A comparison with OO analysis is included in my book *Metapattern* (Addison-Wesley, 2001) which also contains analysis patterns based on application of metapattern.

[2] Actually, the equivalence of situation and context reflects a limited orientation. Though limited, it is optimally suited when the ontological status of an objective reality does not pose problems. In fact, human behaviors in daily life, including most conceptual information modeling in practice, can easily rest on the assumption of *the* objective reality. But a more general, philosophical orientation should recognize the difference between situation and context. In *Semiosis & Sign Exchange: design for a subjective situationism, including conceptual grounds of business information modeling* (Information Dynamics, 2002), I have developed such themes in depth.

[3] In *Metapattern* (see note 1), chapter 5 treats composition as a Cartesian product. Such compositions contain something like a hidden hierarchy. Awareness of their structure helps drive out duplication even more; using those information objects can be considered a special case of inheritance.

[4] These definitions are taken from 'Multicontextual paradigm for object orientation: a development of information modeling towards fifth behavioral form,' published in my book *Informatiekundige ontwerpleer* (Ten Hagen & Stam, 1999).