Modeling with Objects and Values: Issues and Perspectives

Gabriel Eckert, Magnus Kempe

Gabriel Eckert is a doctoral candidate pursuing research on object-oriented analysis and specification. Magnus Kempe is a lecturer, pursuing research in the foundations of software engineering methods.

They can be reached at the Software Engineering Lab, Swiss Federal Institute of Technology in Lausanne (EPFL-DI-LGL, CH-1015 Lausanne, Switzerland) or at Gabriel.Eckert@di.epfl.ch and Magnus.Kempe@di.epfl.ch.

Abstract

The nature, purpose and assumptions of modeling are discussed. The concepts of "value" and "object" are assessed in this context. Their fundamental nature is compared and different uses are investigated. Various approaches are illustrated with comparative examples, and we indicate in what respect they may be reconciled beyond their (perceived) oppositions. The paper has the form of a Socratic dialogue.

Introduction

Since most object-centered approaches still use the notion of primitive data types (e.g. boolean and integer values), it is quite natural to ask whether values are necessary in object orientation, what objects and values are and what purpose they serve in software engineering methods and modeling techniques.

This paper deals with a range of challenging issues: perspectives on the modeling task, structure of models and their building-blocks, as well as implementation techniques. This discussion of fundamental modeling notions does not lead to immediate technical results, but will help, we hope, the practicing software engineer to face modeling decisions with objects and values. The aim of this research is to contribute to the foundations of a sound, explicit framework for software development, and to stress a continuity in structure from analysis methods to design to implementation.

The paper has the form of a Socratic dialogue, in order to develop threads of different views in argumentation. This allows us to simultaneously present separate perspectives, while raising questions that the reader will probably ask himself when reading the arguments.

The dialogue develops according to several dimensions:

- We delimit the domain of debate, trying to reach agreement on terminology.
- We look for advantages and inconveniences of two perspectives which can be summarized as follows: "everything is an object," and "objects have values."
- Commonalities and oppositions between these perspectives are identified.
- The reader is led to consider to what extent each approach influences the theory and practice of software engineering.

In summary, we try in this paper:

- to describe the nature, purpose, and assumptions of modeling;
- to establish to what extent the usefulness of a model is affected by the presence or absence of the fundamental concepts of "value" and "object";
- to compare constructively the positive and negative influences of different uses of these fundamental concepts;
- to relate the discussion to some prominent development methods; and
- to illustrate with examples the applicability of each approach, and determine in what respect they may be reconciled in practice, beyond their (perceived) oppositions.

The reader is assumed to have a working knowledge of object-oriented techniques and conceptual modeling approaches.

Dialogue

To introduce the reader to the dialogue, we begin by setting the stage with two extreme opinions. These opinions are designed to throw the subject matter—objects and values—open to the floor. The two first "arguers" very soon leave the stage, then two thinkers take over.

EXTREMUS. Everything is an object. Even so-called "values" are really objects. Take a "red" door; "redness" is the object being designated. The world is made of objects, nothing but objects.

UTMOSTUS. All I see is values in qualities: length, weight, color. "Objects" are merely aggregates of values; you know an "object" through its values and what happens to these. Values count.

EXTREMUS. I object. Only counters count, and a counter is an object.

UTMOSTUS. Rubbish! One, two, and three are values, not objects. Anyone can see that.

EXTREMUS. Wrong, Utmostus. Numbers are mathematical objects. To start with, there is no definition of values.

UTMOSTUS. Yes, Extremus, there is: A value is a value is a value. What and where are objects?

EXTREMUS. Objects are a frame of mind, and as [Meyer 88] stated "The objects are just there for the picking." (p. 51)

UTMOSTUS. "Nuts!" as [Coad 91] answered (p. 52). When do you know you have "picked" an object?

EXTREMUS. I recognize an object when I see one.

UTMOSTUS. Your objects are of no value to me. Good-bye, Extremus. (Apart) Is "good" an object?

EXTREMUS. Good-bye, Utmostus.

They walk out. Meanwhile, Objectus and Valorus, two thinkers, were listening; they take over and develop their arguments.

Modeling

VALORUS. Well, the subject seems quite interesting. I haven't heard any deep thoughts, though. What is the real issue in thinking with objects? Anyway, what is the purpose of discussing it?

OBJECTUS. The core of object-orientation is the notion of an object as a representation of an entity of the Universe of Discourse (some slice of reality) about which we want to record information. Such an entity has properties, both having objective existence as discrete chunks of reality.

VALORUS. I don't know if properties are really discrete chunks of reality.

Purpose of Modeling

OBJECTUS. Actually I don't know either. I don't even know if *entities* are really discrete chunks of reality. Reality is probably too amorphous, disordered and elusive to ever be described precisely by such a simple model. We are looking at the real world, but always with a view toward modeling it for our purpose, i.e. to develop software. The question is: What is a useful way to perceive information for that purpose? What constructs are useful for organizing our perceptions and discussions about the real world?

VALORUS. We do model the world for our purpose, in this case to have a machine compute something useful. Nevertheless, an underlying fact must not be evaded: we do not choose how we perceive and can understand the world. As [Piaget 26] and [Montessori 67] have observed, children start by grasping discrete objects, before grasping e.g. quantitative relationships.

Dependance on Perception

OBJECTUS. However, it is important to recognize that the choice of a modeling tool (a set of basic constructs) will influence our perception because it "is more than a passive medium for recording our view of reality. It shapes that view, and limits our perceptions. If a mind is committed to a certain [tool], then it will perform amazing feats of distortion to see things structured that way, and it will simply be blind to the things which don't fit that structure." [Kent 78]

VALORUS. Our thoughts and values will certainly influence what we *do* with the evidence provided by our perception, but the perception remains the same. How we interpret it is a later, conceptual stage. The perceptual is the given, one doesn't choose how one perceives; one simply does perceive. Where one takes it from there is up to one's conceptual choices.

Given the source and form of our knowledge, we can ask ourselves: what does it mean for the way we control machine computations? You seem to advocate a reverse approach: given what we want/ expect to do with our machines, let's define (by an act of pure will?) the way we perceive and know the world, so that our thinking fits within the schema of the machine.

OBJECTUS. Not quite. I would rather say: given a portion of the universe that we want to model in order to automate it, let's critically review the way we tend to perceive it, so that our current thinking schema does not distort that world too much.

VALORUS. Maybe you were using a loose meaning of "perception" which is: an understanding or vision of the world. This is a confusing usage. Understanding is abstract, conceptual work; perception is direct, concrete, non-conceptual.

Perspective of Modeling

OBJECTUS. Our values and habits have an effect on our concrete perception: reasoning is based on interpretations of perception. Remember that "... the person holding a hammer tends to see everything in the world as a nail." [Booch 91]

VALORUS. At least, the hammer isn't seen as a nail!

OBJECTUS. Yes, but analysts tend to concentrate on nails when they have a hammer. For example, the Entity-Relationship approach has changed the analyst's perception of the world. Analysts do not model reality, but *the way people perceive it*.

VALORUS. There is nothing new in your ideas. Furthermore, I see an infinite regress: "A model models a model which models…" If people perceive reality (and thus "model" it), why can't analysts too? If analysts merely modeled what people have already modeled, no one would have found a need for programming languages, scroll bars, etc.

OBJECTUS. Analysts model the problem space. You are now talking about the solution space. This is a part of design, not of modeling.

VALORUS. The *need* for solutions is part of the problem space. Also, the solution space itself should be organized around objects ("components") [Cox 91].

OBJECTUS. Modeling deals exclusively with reality as perceived. Analysts should recognize that their chosen modeling tool influences in turn their perception.

Assumptions

VALORUS. You seem to deny that: we are aware of a real, independent world; the world is what it is and our awareness controls neither the nature of the world nor its own nature.

OBJECTUS. I do not deny that the world is what it is. I just wonder what the relation between the world and our model of the world is. Modeling cannot avoid assumptions: ontological (about the nature of the world) and epistemological (about the nature of knowledge).

Realism postulates that the universe comprises objects which can be modeled independently of the observers. In contrast, subjectivism holds that reality is a subjective construction of the mind influenced by culture and social environment. These different conceptions result in two different interpretations of models.

VALORUS. Indeed, Objectus. Realists believe that a model *reflects* reality, whereas subjectivists argue that a model *constructs* reality [Klein 87].

OBJECTUS. Precisely, Valorus. And the two positions are irreconcilable, although each has its validity in a given perspective.

Truth versus Use

VALORUS. The realists are looking for an absolute truth, inherent in the real world, while subjectivists, being pragmatic, look for what "works." But it seems that what is true *is* what works, and conversely.

OBJECTUS. There is an important difference between truth and utility. Users only want models which are useful. The first goal is therefore to reconcile the views of the users in order to devise an acceptable and useful model. This is the purpose of analysis.

VALORUS. Only true models are useful. They are the only "acceptable" models.

OBJECTUS. A simplified model of reality is not a true model, but is often a very useful one.

VALORUS. A false model would by necessity imply contradictions, i.e. it would to some extent clash with reality and for this very reason be useless. The proper meaning and purpose of analysis is to grasp the essence of the problem one wants to solve; such a grasp does not allow for falsehoods. Seeking unprincipled "utility" is a dead-end. The useless is that which has no practical use—in reality. Usefulness is a consequence of *correspondence to reality*, not of contradiction.

OBJECTUS. You are giving in to your realist tendency, Valorus. From a subjectivist point of view, contradiction is part of the reality to be modeled; distinct stakeholders will often express contradictory requirements. This is not only natural, but common sense, because they have divergent needs, i.e. multiple and conflicting goals. Modeling aims at deriving consensus and finding an acceptable compromise among partially contradictory preferences.

VALORUS. In due time the understanding of two different persons will be the same, because both look at the same universe, at the same facts. An understanding *of the world* is not subjective, otherwise it would be arbitrary, i.e. unrelated to anything real.

OBJECTUS. But a formalism for modeling is arbitrary—an artificial structure, with rules and limitations.

VALORUS. Since the Universe also has rules and limitations, maybe a good formalism *is* possible.

How useful is useful?

OBJECTUS. During modeling, analysts have to force a useful fit between the real world and a model expressed in this formalism. The mapping is not at all obvious and it is not even certain that the formalism is expressive enough to construct a true model.

Sometimes analysts have to perceive the real world in a new way in order to achieve this mapping. Sometimes the real world has to be altered to fit a possible model (a company's view of its activities changes as they are automated). We hope to provide modeling constructs that are close to the "natural" perception of their real world counterparts, but we have to admit that current approaches are far from this ideal. This does not imply that no useful model can result, although we know that they are not true models of the world.

VALORUS. If a model is not true *to* reality, it isn't a model *of* reality.

OBJECTUS. The quest for truth is valuable, but it is unlikely that such an absolute goal allows the effective reconciliation of different user views. A model doesn't have to be true. It has to be correct (verification against some rules, internal coherence) and, most important, it has to be useful (validation or evaluation of its worthiness).

VALORUS. Your requirements for correctness follow from what the world is: lawful, ordered, and non-contradictory—i.e. logical. So a better formulation would be that a correct model has to be true to reality.

In addition, your requirements for usefulness presuppose a standard of judgement. If the standard is set by "users," where do they get it from? From reality: what will this program achieve, in fact? how efficient will it be, compared to today, in fact? etc.

OBJECTUS. The standard is not objectively given; it only results from the functions the users want. Modeling is not a deterministic process, i.e. there are many different models for the same universe of discourse.

VALORUS. Reality rules the modeling process. User requirements are a *part* of reality. If a program is designed around arbitrary ideas, if the user requirements are not ruled by reality, the program will be of little use, since its assumptions clash with reality. So the subjectivists may need some advice from the realists.

Objects for Simplicity

OBJECTUS. On the other hand, how badly oversimplified is the view of the real world in current modeling techniques? In computer science, people have gotten used to viewing things in a highly simplistic way, dictated by the kind of tools they have at their disposal. "The thing that makes computers so hard to deal with is not their complexity, but their utter simplicity." [Kent 78] Objects are an appropriate foundation but the object model must be extended.

VALORUS. Indeed, the goal is to extend the power of the object model to capture more elements of the real-world problems we face, and to enhance our ability to deal with the complexity and maintainability of large programs.

OBJECTUS. But this improved object model must still be simple and uniform: a small number of primitive constructs must be used coherently and systematically in modeling.

VALORUS. I fully agree.

Reductionism

OBJECTUS. Most development methods distinguish between objects (or entities, or classes) and values (or attributes). However, this distinction is not formally grounded and stays at a very intuitive level. Since a value can be modeled by an object, why then should we use separate constructs for objects and values?

VALORUS. We shouldn't go too far in a reductionist search for simplicity and uniformity. As I said earlier, objects are natural starting points. So the proper question is: what is the need for a concept of value?

OBJECTUS. I agree on that. Modeling tools result from a difficult trade-off. What are the advantages and drawbacks of a distinct concept of value? What is the need for such a concept (in terms of increased modeling power) and what is its cost (in terms of loss of simplicity)?

Values and Objects

VALORUS. We need values to know what the specific attributes and properties of an object are. For example, Coad and Yourdon recommend to make "each attribute capture an 'atomic concept,' meaning a single value or a tightly-related grouping of values." [Coad 91] (p. 123) They add that it might be a "natural grouping of data elements."

Attributes have values—e.g. a length of 10 feet, or a red color. Two objects may have the same attribute or characteristic, say length, but the specific quantities, and therefore measurements of these attributes, differ: they have different values (e.g. a ball is larger than another ball if the radius of the former is larger than the radius of the latter). Attributes and their values allow us to relate objects to each other, and to group them according to similarities. Values are indispensable.

OBJECTUS. You seem to refer to the Entity-Relationship approach which is based on three primitives: entity, attribute and relationship. In my opinion, the object notion includes both concepts of entity and attribute. However, the object notion lacks the concept of relationship. Thus I advocate two basic, sufficient constructs for structural modeling: objects and relationships among objects.

VALORUS. This is an old and stale discussion. Why is it unreasonable to have three basic concepts? Why should we absolutely reduce everything to two basic elements? If reduction is the ultimate criterion, why not go further and have only one building block?

OBJECTUS. Yes, Valorus, for the sake of simplicity and uniformity, one basic construct would be ideal; for the sake of expressiveness and generality, it probably wouldn't.

VALORUS. Measuring simplicity only according to the number of building blocks is simplistic. Reduction to exclusively one construct doesn't make everything simple and uniform. We need intellectual tools and distinctions to manage the complexity of our problems.

Attributes and Relationships

OBJECTUS. A distinct attribute concept is not needed. Each object can be an attribute of another object. The term attribute does not refer to a special kind of object but only to a special relationship between objects. An object may be related to another by an attribute relationship.

Entity and Attribute

VALORUS. Objectus, I'm afraid we need to rehash old arguments: An attribute is an inseparable aspect of an entity. An attribute cannot be taken "out" of an entity, unlike parts of a whole. "Entities are the only primary existents. (Attributes cannot exist by themselves, they are merely the characteristics of entities …)" "An attribute is something which is not the entity itself. No one attribute constitutes the whole entity, but all of them together are the entity—not 'possessed by' but 'are' the entity." [Rand 90]

OBJECTUS. You use "attribute" with the meaning of "characteristic" or "property." My definition is that an "attribute" is an (internal) structural element of an object, a part of a whole. This approach simplifies modeling. An attribute object characterizes another object to which it is related. This attribute relationship may have various meanings: "has-color," "has-name," "has-part," etc.

VALORUS. On the contrary, attributes do not exist by themselves and are thus not proper objects. Rumbaugh *et al* say that if "the independent existence of an entity is important, rather than just its value, then it is an object." [Rumbaugh 91] (p.162)

OBJECTUS. But they also acknowledge that "the distinction often depends on the application." We don't need "pure data values" for attributes. We don't have to decide whether a real-world thing is to be modeled as either an object or an attribute. It may be both. In Fusion, a more recent method, the duality of objects and values is even explicitly recognized: "Any classes, which do not have any attributes, and which are related to only one other class, can be turned into attributes." [Coleman 94] (p.40)

VALORUS. *From the perspective of programming*, would you go one step further, and say that an object must be usable as both an object or an attribute, depending on the situation? You don't seem to make any distinction between objects and their attributes (other than that the latter are parts of the former).

OBJECTUS. *It's also true from the perspective of modeling.* However, most people are inclined to distinguish between relationships of an entity and attributes of an entity, and therefore express them in different ways; so what are the objective distinguishing criteria to consistently support this intuition?

VALORUS. The objective basis is, from the point of view of knowledge or modeling, that attributes are inseparable aspects of an entity (e.g. the color, the shape of a table), whereas relationships are not intrinsic to one entity but involve two or more existents. So it is not a matter of "intuition." One can observe an entity with multiple attributes on its own, but a relationship does not exist on its own.

Example: The Color of a Car

OBJECTUS. Consider "the color of the car." According to you, "color" is an attribute of "car" entities. I suppose you hold that "red" is the value of this attribute in "your car is red," while "your car" is an entity.

What is inseparable from the car? Neither the concept of color nor the concept of redness is an inseparable aspect of the car because we may consider these concepts without regard to a car: "Red is my favorite color." We relate color and redness to existents: for instance, you order a red car which is built (without color), painted (with red paint) and delivered (as a red car). Is there any reason not to consider the red color as an independent existent standing in a relationship with the car?

VALORUS. An attribute can be mentally separated, isolated, but such a *mental* operation doesn't take e.g. color away from any car. A car is colored; there is no way one could, in reality, take out "color" from the car. It makes a difference whether we look for real objects, mental objects, or software objects. I think your formalism is taking an inseparable attribute (the color of the car) and making a mental separation, which can be carried into a software separation.

OBJECTUS. What you mean by "inseparable" is the general observation that every car has a color and therefore the consequence that your car must have a color, which happens to be the red one now. This color may change. "Inseparable" means that the relationship between car and color is mandatory. Multiplicity (or cardinality) of this relationship allows for more than one color.

VALORUS. Yes, Objectus, I have to acknowledge it: Your formalism of objects and relationships allows to capture without attributes a modeling scheme based on entities and attributes.

OBJECTUS. In conclusion, Valorus, I do not deny that it is useful to include a notion of attribute which satisfies our intuition. I am just arguing that this construct is not a fundamental and distinct concept. One basic linking convention, which we are free to call either "relationship" or "attribute," is sufficient.

Basic building blocks

VALORUS. But, you will have to avoid attributes and values all the way back to basic building blocks.

OBJECTUS. Yes, so let's turn to that issue. Modeling is primarily a decomposition process guided by the purpose of the system under development. The end of this decomposition is reached when every relevant characteristics of the considered universe has been described in terms of *atomic* objects, together with their relationships.

Note that these ultimate components are not objectively given but result from a chosen level of detail in the description and from a subjective perspective. For the car owner, the car's color is atomic, whereas it is not for the car maker. At a different level of abstraction, the manager of a parking lot will probably consider the car itself as atomic: the car is there or isn't, nothing more.

VALORUS. Leaving relationships aside, do you hold that every "thing" in the *world* is an "object"?

Entities, Quantities and Qualities

OBJECTUS. No. At least we should distinguish between objects (things, items, pieces, entities) and quantities (masses, volumes, intensities, lumps) [Wieringa 91]:

- An object is anything that can be put in a set. Objects are discrete and can be counted. Of objects we can ask *how many* of them there are.
- By contrast, a quantity cannot be put into a set or counted. Of quantity we can ask *how much* of it there is. Splitting or merging a quantity yields a quantity.

Quantities are not directly represented in conceptual models. Instead, we define a measure for every kind of quantity about which we want to record information. A measure represents how many units go into a certain quantity and is a function from quantities to discrete objects which proceeds in two steps [Kent 78]:

- From a given quantity to an abstract number, using another quantity as measure.
- From an abstract number to a symbol, using a data type or another notational system.

Measurements of one same quantity can be taken with various units: 140 pounds, 2240 ounces, 63.48 kilograms.

VALORUS. This seems reasonable. On the other hand, enumeration values represent qualitative, not quantitative, relationships; or ordinal, not cardinal, relationships—e.g. low, medium, high, or red, green, blue.

OBJECTUS. Usually we are not interested in the specific symbols used to express a measurement, but in an *occurrence* of the represented *concept*. In other words, measurement results, your "attribute values," are modeled as object instances. There is no need for a distinct notion of "attribute value."

VALORUS. The notion of "attribute value" is important because we should "match the technical representation of a system more closely to the conceptual view of the real world." [Coad 91] The conceptual view is that an attribute has a given quantity, on its own, independently of our knowledge. The measure of this quantity is the value of the attribute. Since values represent measurements, they are a basic building block.

So, Objectus, we agree that not everything in the *world* is an object. But do you hold that every "thing" in the *model* is an "object"?

Stability of the Model

OBJECTUS. Yes. Considering attributes as objects themselves leads to a simpler model with a small number of modeling constructs. Furthermore, it is a nice theoretical approach and, I believe, a sound basis for modeling at various levels of abstraction. It is an advantage to keep the same mental tool at various level of detail while consistently applying our object model. Since our attributes are always objects with full rights, adding further details should not change much of our model, but only refine it. The capability to progressively incorporate more knowledge, while retaining and refining the established framework, is a claimed advantage of the object-oriented approach and the main reason for applying a uniform object model. By focusing first on existing objects in the application domain, we hope to establish a firm and stable model which can later be extended.

Where to start?

VALORUS. Yes, I agree. Objects are stable over time (they stay around), but their attributes may evolve and their behavior is even more volatile (see [Meyer 88, Coad 91]). This is a fundamental principle of an object-centered approach. But a requirement is that additional objects and attributes won't invalidate past models. Having first identified and programmed something as an attribute, using it as an object (and vice-versa) shouldn't require a complete change. So in this sense, objects and attributes should be modeled similarly, an object being made of its attributes ("objects") if necessary.

But we can't have an infinite regress. We must start with some basic objects (to build other objects, for instance); and these basic building blocks, to be useful, must have some sort of characteristics: everything the object is, has, and may do. So what are these characteristics?

Atomic Objects

OBJECTUS. Objects without attributes (e.g. true and false) are the basic building blocks. Such atomic objects, like all objects, may only be acted upon by operation invocation. The complete behavior of an atomic object is statically defined by its operations.

Attributes are used to internally store the changing state of non-atomic objects. Atomic objects are attributeless and hence *immutable*: their state cannot change. Immutability is externally observable (no update operations are available) whereas atomicity is an internal characteristic (the structure is defined without attributes). Atomicity implies immutability, but the converse is not true.

VALORUS. The minimal thing one can say is that objects have a state, some sort of value (e.g. a bit, the ultimate building block of computer programs, has one of two values, "on" or "off").

OBJECTUS. They have a constant state which is not based on attributes or values (note that the term "state" is perhaps not appropriate in this context). Apart from operations, no other ultimate building block is required. It is not necessary to use a value (0 or 1) in order to define an object which exhibits the behavior of a boolean. This can be done on the basis of pure operation descriptions.

VALORUS. I can't imagine any object devoid of attributes. It would have to be literally nothing. So it wouldn't be an object. What are attributeless objects? I haven't seen any.

OBJECTUS. Objects devoid of attributes are not devoid of properties. Attributes reflect an internal hidden structure while externally observable operations reflect properties. Atomic objects exhibit a specific behavior because they have their own operations.

How is it possible? Read the definition of atomic Boolean objects, written in C++ and Ada 9X, two hybrid languages that incorporate both objects and values (see the Appendix). In this example, we use only objects, not values. Instances of Boolean (more precisely, instances of True and False, the two subclasses of Boolean) are atomic and immutable. They exhibit the expected boolean behavior while being full-fledged objects.

VALORUS. Your Boolean example implicitly relies on some sort of data, since instances of False and True are somehow tagged with their types. Aren't there hidden bits?

OBJECTUS. This is only because I haven't defined individual objects but a class (or a type) for many similar object instances. It is only a convenient shortcut with the consequence that every object must "know" its type.

VALORUS. The values of such atomic objects have a status of "primitiveness," and may even be considered "pure data values."

OBJECTUS. You must acknowledge that atomic objects are full-fledged objects. They are acted upon exactly as other objects are and they are defined in the same way. However, you may call them values, if you prefer. In my opinion, the term "pure data value" is not appropriate because they do not encapsulate data (attributes) and are not defined in terms of data but in terms of operations.

VALORUS. I grant you that atomic objects can replace values.

OBJECTUS. It looks like we are using different terms for basic building blocks which play a similar role in each case; "atomic object" emphasizes the similarity with objects in general, and "value" emphasizes the difference from objects.

A Fundamental Distinction

VALORUS. In a classic paper, MacLennan claims that programming languages should explicitly recognize the value/object distinction and incorporate facilities for handling both [MacLennan 82]. He argues that values are atemporal abstractions, unchangeable and non-instantiated (a view reminiscent of Plato's world of Forms). On the other hand, objects correspond to real-world instantiated entities, existing in time, with a changeable state. What do you think?

OBJECTUS. Well, MacLennan makes the point that "values are abstractions." Note that in modeling, this is also said of objects. [Rumbaugh 91], for example, writes that an object is "a concept, abstraction, or thing with crisp boundaries and meaning for the problem at hand."

VALORUS. But this use of "abstraction" is different. [Rumbaugh 91] speaks of abstractions from the real world—i.e. an abstraction is a selective focus on real-world objects; the abstractions identified during modeling are ultimately turned into software objects. This provides continuity from the real world to modeling to design to implementation.

MacLennan, focusing on mathematics and programming, uses abstractions in opposition to what can change; so, according to him, values are abstractions. Mathematical entities such as numbers are the best examples of values; the number 2 is an abstraction that subsumes all particular pairs.

OBJECTUS. The number 2 is a concept whose instances are concrete existing pairs. However, this does not imply that it cannot be an object. The abstract number 2 itself is an instance of the integer concept.

Mathematicians perhaps talk of the abstract number 2, but most users usually think of some concrete pair (like 2 dollars or 2 apples). The claimed atemporality of values may be true for the abstraction, but it is not true for its instances: the represented concrete pairs can be created or destroyed by assembling or separating the involved elements. A concrete 2 is tied to one particular pair in the real world.

Anyway, what is the difference between abstract values and typical objects? The notion of employee is also an abstraction that subsumes all particular employees. The concept of employee continues to exist even if there are no employees because its existence is not tied to the hiring or firing of particular employees. Does it mean that employees are values, as opposed to object?

MacLennan opposes values to objects. I think that values can be objects as well. Values are immutable objects.

VALORUS. Aha! Objects are mutable values...(!?) Your definition is useless. For instance, if my car is white (its color is an "immutable object," white), then what are we to say when the paint degrades in the sun and my car is now "polar white." Has it gotten another "immutable object," or has the color changed (thus its measurement, its value, is different)? I understand that you would like to reduce the number of primary modeling tools; to that end, I think you are defining values away. It is true that a value and an object are both "things," but a value is not an object.

Your immutable objects are my values. How are objects different from immutable objects?

OBJECTUS. There is only one difference: mutability, the ability to dynamically change behavior or properties. Immutable objects are full-fledged objects, with all their expected characteristics.

Numbers

VALORUS. So "2" is an "immutable object." Well, why not. What does "2 + 5" do?

OBJECTUS. "2" is an immutable object. It may be an atomic object or it may be defined in terms of other objects like digit or bit. Its internal structure (atomic or not) should not be confused with its externally observable properties (to behave according to an integer 2). The operations of "2" are part of its observable properties (but are implemented in terms of its internal structure). Such operations comprise the usual arithmetic operations between numbers. Addition, for example, takes two immutable objects ("2" and "5") and returns another one as result ("7").

VALORUS. Where did the latter come from? has it always been around? is it the same object as the one returned by "3 + 4"?

OBJECTUS. There are two possibilities: a) reference and b) non-reference semantics:

a) Operations (and therefore programs) are expressed by combining references (pointers) to particular objects. In this case, “2” is a reference onto the integer object 2; the result of “2 + 5” is a reference identifying the integer object 7. The same object 7 is also referred to as the result of “3 + 4.” There is only one integer object “7” with multiple references.

b) Operations create new objects. E.g. a concrete existing pair (2 apples) is added to another grouping (5 apples). These groupings cease to exist and a new concrete grouping of 7 apples is created. There are many individual objects 7, each one representing a separate grouping. Integer literals allow to create and designate such objects.

VALORUS. “2” is a symbol for the concept “two.” It is a value which can be held by attributes of objects (two legs of a man, two drawers of a table). It is also a value which can be held as data for computation; it is then the value of a measurement (two apples). Since it is a value, “2” may be compared to other (integer) values.

In fact, comparing two objects (establishing what their relationship is in some respect) entails comparing the values, the measurements, of these objects’ attributes (do cars A and B have the same color? do beds C and D have the same size?). To this end, it is necessary that objects have commensurable characteristics. If they didn’t (say, if they were made of objects made of objects made of …) it would seem possible but quite hard to explain how two objects are compared, how assignment works, etc. I thought we were looking for simplicity.

OBJECTUS. Comparisons or assignments can be defined within a pure object paradigm, without resorting to additional notions like pure values. If you meet me tomorrow morning at the market, I’ll show you that it is not only possible but simple. And I will also show you how to deal with the *identity* of atomic objects, an important aspect not addressed today.

VALORUS. I’m looking forward to that, Objectus. Tomorrow then, at sunrise.

OBJECTUS. Yes, Valorus. Shall we summarize before we part?

Conclusion

VALORUS. Modeling must take into account two viewpoints: the subjectivist, which adheres to the preferences of users, and the realist, which adheres to the facts of reality. Although the perception of the world revolves around objects, not everything in the world is an object; there are also quantities and, of course, relationships.

OBJECTUS. Modeling can be based on objects and relationships alone. A distinct value construct is not strictly necessary because atomic objects can play the role of values.

VALORUS. In software development, there is a marked tendency to take existents in the problem space and make them “tangible objects” (of analysis and design) to which actions are directed and around which structures (of other objects) are established. It is a natural, epistemological factor which stabilizes our models.

It still remains to determine how concepts of identity, change, encapsulation, typing and classification relate to these observations and may contribute to stability in analysis while enhancing flexibility in design. Also, while “attributes” and “values” seem to be optional constructs, they may be justified in relation to other issues—e.g. dynamic modeling and layers of abstractions.

OBJECTUS. In developing programs, we do not need values, but we do not need objects either. It is perfectly possible to build software based on only one of these two notions.

Pure object-oriented programming languages, like Smalltalk [Goldberg 83], do not provide values. Everything is an object and every computable system may be implemented in this way. Quite on the opposite, pure functional programming languages, like Haskell [Hudak 92], can be described as completely value-oriented. It is known that every computable system may be implemented in either manner.

However, choosing one or the other implementation technique should not be confused with the choice of a development method. Object-oriented development is not the same as applying an object-oriented programming language, and functional programming doesn’t have much in common with the analysis-and-design principles of functional decomposition.

The apparent conflict between these two extremes should not hide the fact that they are not totally at variance: pure object-oriented programming languages provide atomic objects, which are close to values; modern functional programming languages provide abstract data types, which are close to objects. Future work should be devoted to laying down guidelines on why and how to choose either perspective according to the problem and activity at hand.

VALORUS. We will start tomorrow.

Acknowledgments

The authors would like to thank Stéphane Barbey for his help with the code in the Appendix and Robb Nebbe for his comments and suggestions.

References:

[Booch 91] Object-Oriented Design with Applications
G. Booch
Benjamin/Cummings Publishing Company, 1991

[Coad 91] Object-Oriented Analysis
P. Coad and E. Yourdon
Yourdon Press Computing Series, 1991 (2nd ed.)

[Coleman 94] Object-Oriented Development: The Fusion Method
D. Coleman, P. Arnold, S. Bodoff, C. Dollin, H. Gilchrist, F. Hayes, and P. Jeremaes
Prentice Hall, 1994

[Cox 91] Object-Oriented Programming: An Evolutionary Approach
B. Cox, A. Novobilski
Addison-Wesley, 1991 (2nd ed.)

[Goldberg 83] Smalltalk-80: The Language and its Implementation
A. Goldberg and D. Robson
Addison-Wesley, 1983

[Hudak 92] Report on the Programming Language Haskell
A Non-strict, Purely Functional Language — Version 1.2
P. Hudak (ed.)
ACM SIGPLAN Notices Vol.27, No.5, 1992

[Kent 78] Data and Reality: Basic Assumptions in Data Processing Reconsidered
W. Kent
North-Holland Publishing Company, 1978

[Klein 87] A Comparative Framework of Data Modeling Paradigms and Approaches
H.K. Klein and R.A. Hirschheim
The Computer Journal, Vol.30 No.1, 1987, pp.8-15

[MacLennan 82] Values and Objects in Programming Languages
B.J. MacLennan
SIGPLAN Notices, Vol.17 No.12, 1982, pp.70-79

[Meyer 88] Object-Oriented Software Construction
B. Meyer
Prentice Hall, 1988

[Montessori 67] The Absorbent Mind
M. Montessori
Dell Publishing Co., 1967

[Piaget 26] La représentation du monde chez l'enfant
J. Piaget, 1926

[Rand 90] Introduction to Objectivist Epistemology
A. Rand
New American Library, 1990 (2nd ed.)

[Rumbaugh 91] Object-Oriented Modeling and Design
J. Rumbaugh, M. Blaha, W. Premerlani, F. Eddy, and W. Lorensen
Prentice-Hall International, 1991

[Wieringa 91] Steps towards a method for the formal modeling of dynamic objects
R. Wieringa
Data & Knowledge Engineering, No.6, 1991, pp.509-540

Appendix: Definition of Atomic Boolean Objects in C++ and Ada 9X

C++

```
class BOOLEAN { // no attribute
  public:
    virtual BOOLEAN& operator && (BOOLEAN& right) = 0;
    virtual BOOLEAN& operator || (BOOLEAN& right) = 0;
    virtual BOOLEAN& operator ! () = 0;
};

class FALSE : public BOOLEAN { // no attribute
  public:
    BOOLEAN& operator && (BOOLEAN& right) {return *this;} // always FALSE
    BOOLEAN& operator || (BOOLEAN& right) {return right;}
    BOOLEAN& operator ! ();
};

class TRUE : public BOOLEAN { // no attribute
  public:
    BOOLEAN& operator && (BOOLEAN& right) {return right;}
    BOOLEAN& operator || (BOOLEAN& right) {return *this;} // always TRUE
    BOOLEAN& operator ! ();
};

BOOLEAN& FALSE::operator ! () {return *(new TRUE);}
BOOLEAN& TRUE::operator ! () {return *(new FALSE);}
```

Ada 9X

```
type BOOLEAN is
  abstract tagged null record; -- no attribute

function "AND" (this : BOOLEAN; right : BOOLEAN'class)
  return BOOLEAN'class
  is abstract;

function "OR" (this : BOOLEAN; right : BOOLEAN'class)
  return BOOLEAN'class
  is abstract;

function "NOT" (this : BOOLEAN)
  return BOOLEAN'class
  is abstract;


type FALSE is
  new BOOLEAN with null record; -- no attribute

function "AND" (this : FALSE; right : BOOLEAN'class)
  return BOOLEAN'class is
begin return this; end "AND"; -- always FALSE

function "OR" (this : FALSE; right : BOOLEAN'class)
  return BOOLEAN'class is
begin return right; end "OR";

function "NOT" (this : FALSE)
  return BOOLEAN'class;


type TRUE is
  new BOOLEAN with null record; -- no attribute

function "AND" (this : TRUE; right : BOOLEAN'class)
  return BOOLEAN'class is
begin return right; end "AND";

function "OR" (this : TRUE; right : BOOLEAN'class)
  return BOOLEAN'class is
begin return this; end "OR"; -- always TRUE

function "NOT" (this : TRUE)
  return BOOLEAN'class is
begin return FALSE'(null record); end "NOT";


function "NOT" (this : FALSE)
  return BOOLEAN'class is
begin return TRUE'(null record); end "NOT";
```