

Meaning Matters: Three Kinds of Ontology.

Dr. Tom Johnston

August 21, 2007

Revised August 27, 2007.

Desiring to focus my otherwise somewhat peripatetic thoughts on matters IT, I am starting, with this essay, a series of articles on semantics in IT, initially focusing on ontologies and taxonomies. Having authored a series at DataWarehouse.com several years ago, entitled "Modeling Matters", this series, which I will publish at this website, will be entitled "Meaning Matters".

"Ontology" is currently a very hot buzzword. I have unpublished data modeling notes, going back over ten years, in which I was working out data modeling problems using that term as well as an entire framework of related concepts to think through how meaning is expressed in data models in particular, but more generally how it is expressed by different components of an IT system. Those meaning-contributing components are:

1. Data structures, such as a foreign key in a table enforcing the constraint that no row of that table can be related to more than one row in the foreign-key-referenced table;
2. DBMS-enforced semantic constraints, such as Referential Integrity;
3. Developer-written code, whether database stored procedures or application-specific code; and, finally,
4. The user who uses the application, or accesses the database directly, and the semantics he supplies by means of (a) conventions imposed on data entered into the system, and (b) interpretations placed on data retrieved from the system.

I am, of course, aware that this list is no more than a promissory note, to be cashed in by means of an in-depth commentary, which I will supply later on.

Three Kinds of Ontology.

But back to "ontology". I will begin by enumerating the three senses of the word "ontology" that I will discuss. They are:

1. Classical ontology, as found in the history of philosophy, associated with such names as Thales, Heraclitus, Parmenides, Plato, Aristotle, Augustine, Aquinas, Spinoza, Leibniz, Berkeley, the German Idealists, Hegel, Russell, Quine, Putnam, Kripke and others. For reasons that will become clear shortly, I will also refer to classical ontology as "content ontology".

Meaning Matters: Three Kinds of Ontology.

© Copyright 2008, Dr. Tom Johnston.

Only personal, non-commercial copies, are permitted.

Page 1.

2. Computer science ontology, combining classical ontology with formal logic. The emphasis, not surprisingly given that computer science types are fundamentally engineers, is on the logic, not on the ontology. As a philosopher participating in some of their discussions (for example, in the work of the IEEE 1600.1 to develop a Suggested Upper Merged Ontology), I have found the understanding of content ontology on the part of the computer science community to vary a great deal, but to be, overall, not very good. Regardless, I will call this combination of content ontology with formal logic "formal ontology".
3. Vendor ontology, being the minimal amount of formal ontology which those who will use ontology management tools must understand. It is vendors and their ontology management tools which will provide the only access to the concepts of formal ontology that most IT professionals will get. I will call this minimal level of acquaintance with formal ontology "toolset ontology". It is definitely "ontology-lite", but it does have its place. The development of formal ontologies within businesses cannot require that all workers in the field be proficient in first-order predicate logic, let alone possess a deep understanding of classical or formal ontology.

Let's Get the Job Done.

Most IT professionals are down-to-earth "let's get the job done" types. "Skip the theory, just tell me how to solve this problem" is their mantra. And problems do get solved. Work does get done. So the theory wasn't that important after all, was it?

Work does indeed get done. That's the problem. As long as we can muddle through with systems that break frequently, and don't work well, in various ways, when they do work, then there is no incentive to improve. Indeed, there is little awareness of the need to improve. It's as if an aeronautical engineer designed an airplane that wouldn't fly on cloudy days, that crashed every couple of months, and that required its passengers to carry helium birthday balloons on board to insure that, once aloft, the plane would stay in the sky. There are no airplanes like that. But there *are* plenty of IT systems like that.

Because IT systems do get built, and continue to be maintained, and by workers who generally have little understanding of relevant theory, it is hard to make the argument that "theory is practical". But I believe that it is. So even though the mantra "theory is practical" was introduced by Chris Date, whose understanding of *how* theory is practical I find to be lacking, it is exactly what I want to say. So, with credit given to Date for the phrase, I will say that one of my tasks here is to explain exactly how theory is important to those of us who are paid to be practical, who are paid to build and maintain IT systems and databases.

How This Discussion of Ontology Will Proceed.

Having made, or having attempted to make, that case, I can then move on to discuss content ontology. A brief history of philosophy, focusing on ontology, will begin that discussion. After that, I will explain how I apply my own knowledge of content ontology to data modeling and, more generally, to the problem of expressing meaning in databases and codebases.

Formal ontology has its roots in content ontology and logic, both of which were developed within the field of philosophy. Indeed, Aristotle was the first to develop a system of formal logic. And together with Plato, he is rightly acknowledged as the father of ontology also.

Logic and ontology. Form and content.

Computer scientists make use of systems of logic more powerful than the ones developed by Aristotle. Given certain strings of symbols, software which realizes these systems of logic can transform them into other strings of symbols.

So what's the big deal? Of course we can write code for string manipulation. In fact, certain languages are optimized for doing just that. One of them that I remember using is SNOBOL.

But here's the big deal. Here is the miracle that, together with the scientific method, has lifted mankind to our current dominance over the material world. We can write string manipulation rules to do a very important thing. Given an interpretation to an initial set of strings, an interpretation which makes them a symbolic expression of a statement (a sentence that can be true or false), we can write those rules so that from initial strings which represent true statements, we can generate any number of other strings which represent other true statements! And often, we won't have realized that those other statements were true.

Software that can do that is what computer scientists call an "inference engine". String manipulation rules that generate new truths from old ones are the transformation rules of truth-preserving systems of deductive logic. It is the evolution of relational databases and associated application codebases into these more powerful, logic-driven inference engines, that is behind "ontologies", "taxonomies" and related buzzwords.

Toolset ontology, the third sense of "ontology", is the sad sense. It is the sense in which mastery of a tool is taken for mastery of the craft which uses that tool. This is a sad sense, not only because it is an impoverished sense, but also because it is all that most IT practitioners will ever know about ontology.

Yet we do have a way of recognizing that toolset mastery is not all there is to the practical value of theory. We can ponder on what makes a guru a guru. In IT, a guru is

the guy who can be given a description of a problem, and who can figure out what to do to fix it. Not by trial and error, by patch the code and hope it doesn't come back up and bite you. But by understanding something that it's hard to put a name to. So we often say, by understanding how the system *really* works. This is nothing other than an understanding of theory, the micro-theory relevant to the specific IT system in question.

Content Ontology: a Quick Sketch.

{to be filled in later. At this point, some names may be misspelled (I never was good at spelling), some dates may be off. Right now, this is all from memory, not even google-checked. However, I can now see that the term “content ontology” begs some questions. I’d like to answer those questions before resuming the use of that label. So until that’s accomplished, I’ll call this kind of ontology “classical” or “philosophical”, since its practitioners are those that history (and academic divisions) calls philosophers, since its (post-mythological) roots extend back almost three-thousand years, and since it constitutes, along with other accomplishments of the ancient Greeks, the foundations of classical European civilization.

- Myth.
 - Focus on explaining processes, not things or stuff.
 - anthropomorphism: the gods as powerful personalities.
 - Control of the physical world = pleasing the gods.
 - Lesson learned: This kind of explanation of physical phenomena doesn’t really worked too well. If “It’s the god’s will” can explain anything, then it can explain nothing.

- Thales.
 - “Father of philosophy”.
 - 700 BC.
 - Everything and all stuff a transformation of water.
 - (Thales lived on Miletus, an island.)
 - control of the physical world = managing stuff and things.
 - Human thought
 - Explanations of human behavior in terms of personalities, motives, emotions, etc. are not useful when applied to inanimate natural phenomena – floods, earthquakes, drought, a season of bountiful harvests, etc.
 - Control based on a distinction between appearance (effect) and reality (cause). Appearance evident; reality usually hidden.
 - Drive for unity. Reality is one, not many. (compare contemporary physics: four forces are symmetry-broken manifestations of a single force; matter and also force-mediating particles all manifestations of a single particle.)
 - Lessons learned.

- For true knowledge, must probe beyond the appearances.
 - Ultimately, reality is one.
 - (Skip Anaximander and Anaximenes – just Thales redux).
- The Atomists (Democritus, Leucippus).
 - Mythological trappings of the Milesians (basic reality is water, air) discarded. Basic reality are particles too tiny to be visible. Everything we see around us is a combination of them.
 - Lessons learned.
 - 500 BC? Boy, those Greeks were smart!
 - Reality may not be something available to us through perception, but only through reasoning.
- Heraclitus, Parmenides.
 - Heraclitus: change. *Panta rei*, all things change. How, then, permanence?
 - Parmenides: being is. How, then, can it change?

I'm going to wrap up for today with an observation on the relevance of material like this to the challenge of expressing semantics in information systems. Among computer scientists working on formal ontologies, I think the prevalent opinion is that classical ontology is interesting, but not particularly germane to their work. There are even those who claim that there is no connection at all other than the accident of one of the early comp sci researchers using the term "ontology" to characterize what he and his peers were doing. If this is true, "ontology" is a homonym, effectively two words that are spelled and pronounced the same, but distinct words as far as meaning is concerned. More to the point, it means that the review of philosophical ontology that I am writing here is of little or no value for we IT practitioners.

There are exceptions, of course, and one of the most prominent among them is Dr. John Sowa (check out his website, also his book [Knowledge Representation](#)). He believes that many of the specific conclusions reached by these classical philosophers can help us develop structured sets of concepts whose semantics can be manipulated by software, not just understood by people. He's done such an excellent job that I'm actually kind of jealous. Especially since he's not a professional philosopher himself (as I am), but rather a professional mathematician working within computer science.

But the kind of relevance that John finds in philosophy is a specific kind. It's as if the history of philosophy is the site of an archeological dig at which many important objects have been unearthed. Aristotle's categories, and his logic; the medieval disputes between realists and nominalists; Kant's categories; Whitehead's process metaphysics; and, most especially, the categories of C. S. Peirce.

I agree that this kind of relevance does exist. Because of my different background and interests, however, I would dig in several areas of the site that John has not concerned himself with, and would ignore other areas where he has worked diligently. For this "bits

and pieces” relevance of classical ontology to formal ontology in particular and to IT system semantics in general, my short list would look like this:

- Aristotle*
- Ockham
- Duns Scotus
- Frege*
- Wittgenstein*
- Sellars
- Quine*

The asterisked items on this list are those that are relevant in this first sense, as well as in a second sense. If this first kind of relevance lies in the answers, or precursors to answers, to questions we already know to ask, the second kind of relevance lies in a way of looking at things, and thinking about things, that leads us to ask certain questions and, just as importantly, to not even think of asking other questions.

This second kind of relevance is a much harder case to make. It’s an intuition about where to dig in the archeological site of the mind. It’s a skill in using the trowel and the series of ever-finer sieves of the mind to discard the dirt and keep the shards. It’s the imagination to see the greater whole that these conceptual shards were once part of. It’s a way of thinking about things.

The value of “a way of thinking about things” is something taken for granted, by and large, within the academic community. Basically, it’s what distinguishes the best senior academics from the junior faculty. It’s what distinguishes the experienced surgeon from the young man just completing his internship. In IT, it’s what distinguishes the guy who designs elegant solutions from the guys who just design some kind of solution and then, via brute force, somehow get it to work (most of the time, anyway). In IT just as much as in medicine, it’s what distinguishes the person who can diagnose underlying causes from the others who can merely treat symptoms.

{ a pause here. 8/27/07. }