CATEGORIES AND CLASSES

By: Dave McComb

semantic arts

Getting the category/class distinction right is one of the key drivers of the cost of traditional systems.

We've been working with two clients lately, both of whom are using an ontology as a basis for their SOA messages as well as the design of their future systems. As we've been building an ontology for this purpose we became aware of a distinction that we think is quite important, we wanted to formalize it and share it here.

In an ontology there is no real distinction that I know of between a class and a category. That is: classes are used for categorizing, and you categorize things into classes. If you wanted to make a distinction, it might be that category is used more in the verb form as something you do, and the class is the noun form.

Categories and Classes in Traditional Apps

But back in the world of traditional applications there is a quite significant difference (although again I don't believe this difference has ever been elaborated). In a traditional (relational or object oriented) application if you just wanted to categorize something, (say by gender: male and female) you would create a gender attribute and depending on how much control you wanted to put on its content you would either create an enum, a lookup table or just allow anything. On the other hand if you wanted behavioral or structural differences between the categories (let's say you wanted to distinguish sub contractors from employees) you would set up separate classes or tables for them, potentially with different attributes and relationships.

We've been studying lately what drives the cost of traditional systems, and getting this category/class distinction right is one of the key drivers. Here's why: in a traditional system, every time you add a new class you have increased the cost and complexity of the system. If you reverse engineer the function point methodology, you'll see that the introduction of a new "entity" (class) is the single biggest cost driver for an estimate. So every distinction that might have been a class, that gets converted to a category, provides a big economic payoff.

It's possible to overdo this. If you make something a category that should have been a class, you end up pushing behavior into the application code, which generally is even less tractable than the schema. So we were interested in coming up with some guidelines for when to make a distinction a category and when to make it a class.

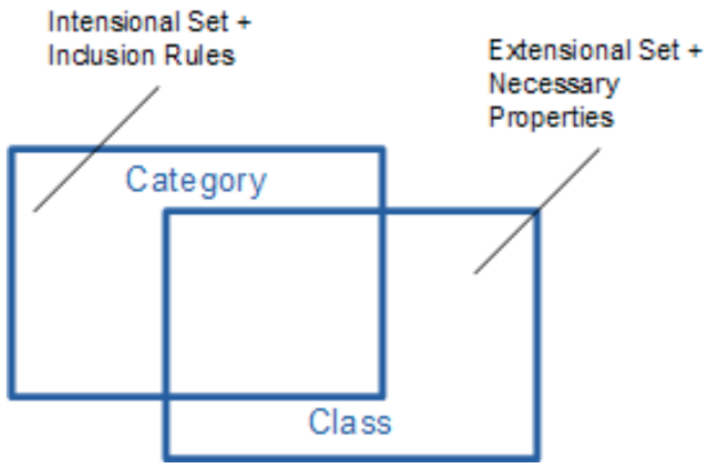
Category and Class in gist

As it turns out, we had foreshadowed this distinction, although not for this reason, in gist, our upper ontology. Gist has a class called "category" whose intent is to carry categorical distinctions (from one lower level ontology to another) without necessarily carrying their definitions.

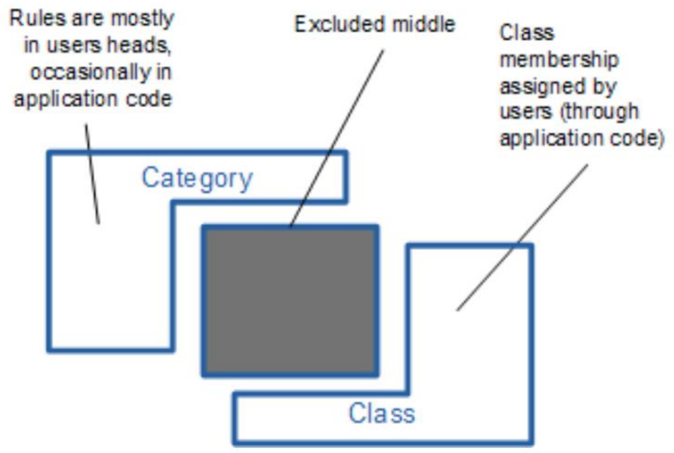
For instance when we worked with a State Department of Transportation, we had a class in their enterprise ontology called "Roadside Feature." A Roadside Feature has properties such as location and when by what process it was recorded. Several of their applications had specific roadside features, for instance "fire hydrants." In the application fire hydrant is a class, and therefore is one in the application ontology. But in the enterprise ontology "fire hydrant" is an instance of the category class. Instances of fire hydrant are members of the roadside feature class at the enterprise ontology level, and associated with the category "fire hydrant" via a property "categorizedBy." A fire hydrant can therefore be created in an application and communicated to another application that doesn't know the definition of fire hydrant, with almost no loss of information. The only thing that is lost on the receiving end is the definition of fire hydrant, not any of the properties that had been acquired by this fire hydrant.

Category and Class, a Formal Distinction

So we came to this: a category is an intensional set with criteria for defining membership. A class is an extensional set where membership is explicitly asserted and specific properties can be defined as necessary.



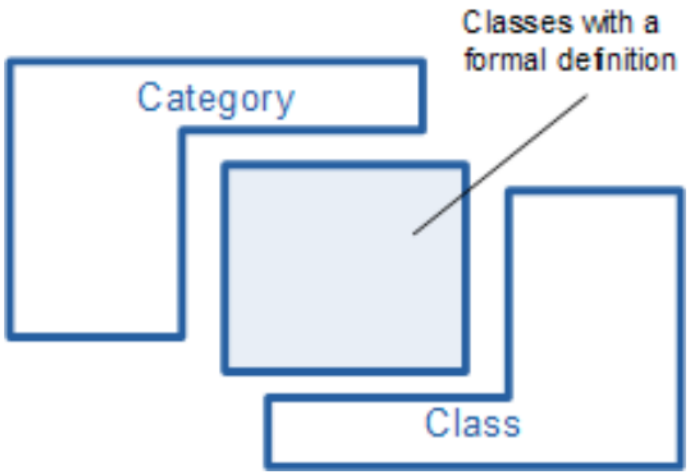
In an ontology these two definitions can, and almost always do, overlap. But in a traditional system they don't. In a traditional system the overlap is "the excluded middle."



semantic arts

This is helpful for us if we're using our ontology to generate artifacts for traditional systems, but on closer inspection we're finding interesting use even in ontology driven systems. The area in the class rectangle outside the category box is essentially the primitive classes, those that cannot be defined in terms of other classes and properties. The intersecting region are classes that are formally defined, and therefore we could infer membership. And the category outside class is something where we accept that a distinction has been made, we may know the sufficient properties, but we don't necessarily know any other necessary properties. Nor do we need to know how the individual got categorized.

In an ontology, we focus most of our effort on the intersection:



…these are the classes that have formal definitions. But if we think deeply about it, what we are doing when we define a class and give it a formal definition is: we name a class and say it is equivalent to a restriction. The two things we traditionally haven't focused on are classes without definitions (primitive classes) and categories without classes (many instance based taxonomies fit this pattern).

Our initial work suggests that this is a key pattern for getting large constellations of ontologies to work together. Feedback, comments, brickbats all welcome.