Ontological Engineering for Conceptual Modeling

Ryszard (Richard) Raban[1], Brian Garner[2]

[1] Faculty of Information Technology
University of Technology, Sydney
richard@socs.uts.edu.au
[2] School of Computing and Mathematics
Deakin University
brian@deakin.edu.au

Abstract. In acknowledging the importance of ontologies in conceptual modeling, database integration and business process modeling, this paper introduces a set of principles for building ontologies. Starting from Guarino's meta-properties of ontological terms, the paper describes the denotational semantics of the meta-properties and derives from them some engineering rules and checks for constructing domain specific conceptual models, based on the overarching requirement to assign meanings to concepts using tags and labels. Parallel research by the authors into the use of contextual references and roles to restrict such meanings will be published elsewhere.

1 Introduction

Ontologies grew out of being a philosophical endeavor of finding the top level categories of existence (those appear in the works of Aristotle through Kant, Heidegger to the ontological categories introduced recently by Sowa [1]) into an important tool in domain knowledge representation, and recently, in information systems modeling. Whenever a model is built, a certain ontological commitment is assumed. In most cases, however, this commitment is not explicitly stated. It makes information exchange, interoperability and integration very difficult. XML [2], XMI [3] or UML [4], [5], [6], to mention just some of the prominent modeling and integration tools, will not be able to realize their full potential in the area of knowledge sharing and integration unless the tags and labels used have ontology-based semantics.

As ontologies become more and more part of knowledge engineering and modeling [7], [8], [9], it is imperative to have precise rules for engineering ontlologies themselves.

A similar problem also arises in managing changing contexts within reasoning processes which, as argued in [10], often requires dynamic ontology modifications. In order to be able to manipulate ontologies without loosing their internal consistency, strict ontology construction rules are required.

This paper aims to advance the basis of ontological engineering by specifying rules for creating a hierarchy of properties that individuals might have and by exploring some implications of the rules for conceptual modeling. We start by defining the denotational semantics of ontological meta-properties, also referred to in this paper as criteria, and then derive from them rules for properties subtyping. The meta-properties, which were partially introduced in [11], [12], [13], [14], [15] and have been recently refined and formalized by Guarino and Welty in [16], were adopted as the starting point for this research.

2 Extensional semantics of meta-properties.

Let's assume that in a particular Universe of Discourse there are individuals *x* that can be dynamically created and destroyed.

Let ω_t be the existence predicate defined on all *x* such that $\omega_t(x)$ means that an individual *x* exits at a point in time *t*. We assume that each individual might have exactly one creation time t_{cx} such that

$$\forall_{t < tcx} \neg\omega_t(x) \ \wedge \ \exists t_\Delta \forall_{tcx \leq t < tcx + t\Delta} \ \omega_t(x)$$

and exactly one destruction time t_{dx} such that

$$\forall_{t > tdx} \neg\omega_t(x) \ \wedge \ \exists t_\Delta \forall_{tdx - t\Delta < t \leq tdx} \ \omega_t(x).$$

Let $\tau_x = [t_{cx}, t_{dx}]$ be the period in which an individual *x* exists or the individual *x*'s *lifetime*.

Let $\Omega_t = \{x \mid \omega_t(x)\}$ be a set of all individuals existing at a point in time *t*.

First-order properties correspond to monadic predicates over Ω_t. If *P* is a property, $P_t(x)$ means that an individual *x* has property *P* at a point in time *t*. This, of course, implies that the individual *x* must exist at the point in time *t* to have the property. For example, individuals of a Universe of Discourse might have properties like *PERSON*, *EMPLOYEE*, *RED*, *SHORT*, etc. Note that properties are not attributes like *LENGTH*, or *COLOR* which are functions defined on a domain of individuals and return attribute values. For example, *LENGTH*(*x*) = 175cm or *COLOR*(*y*) = 'red'.

A individual *x* acquires a property *P* at a point in time $t_{cP(x)}$ such that

$$\forall_{t < tcP(x)} \neg P_t(x) \ \wedge \ \exists_{t\Delta} \forall_{tcP(x) \leq t < tcP(x) + t\Delta} \ P_t(x)$$

and discards it at a point in time $t_{dP(x)}$ such that

$$\forall_{t > tdP(x)} \neg P_t(x) \ \wedge \ \exists_{t\Delta} \forall_{tdP(x) - t\Delta < t \leq tdP(x)} P_t(x).$$

A period of time during which an individual *x* holds a property *P* is called *an attribution period* of *P* to *x* and is defined as $\tau_{P(x)} = [t_{cP(x)}, t_{dP(x)}]$ such that

$$\forall_{t \in \tau P(x)} P_t(x).$$

In this paper, the discussion is limited to such properties that apply for some period of time to at least one individual, that is $\forall_P \exists_x t_{cP(x)}, < t_{dP(x)}$.

A property *P* can have many attribution periods for an instance *x*. For example, a person can hold property *STUDENT* many times in different periods of time.

We call a property *P* *static* if;

$$\forall_x P_t(x) \rightarrow (\tau_{P(x)} = \tau_x)$$

Otherwise, a property is called *dynamic*.

A static property is inherent to an individual; it is acquired at its creation time and holds for its entire lifetime. Properties like *PERSON*, *LIVING-BEING* are static. A dynamic property is acquired by an individual temporarily, and it can be acquired and discarded many times in an individual's lifetime. The previously mentioned property *STUDENT* is an example of a dynamic property.

The denotation of a property P is defined as $\delta P_t = \{x \mid P_t(x)\}$ and represents a set of all individuals that have property P at a point in time t.

It is also valid to use the denotation for the existence predicate - $\delta\omega_t$ is a set of all individuals existing at a point in time t.

2.1 Subtyping

If a property Q is a subtype of a property, denoted by $Q \leq P$, then

$$\forall_t\ (\delta Q_t \subseteq \delta P_t).$$

If additionally

$$\exists_t\ (\delta Q_t \subset \delta P_t),$$

then Q is a *proper subtype* of P, which we denote by $Q < P$. If a property Q is a (proper) subtype of a property P, it can be said that a property P is *a (proper) supertype* of a property Q.

There could be two different types of subtyping. Firstly, we can have time independent or *static subtyping*. In this case, a static property Q is a subtype of a static property P. For example, property *PERSON* is a static subtype of property *LIVING-BEING* and it means that a *living-being* can also be a *person,* and if it is so, it remains a *person* for its lifetime.

Secondly, we can have time dependent subtyping or *dynamic subtyping*. In this case, a dynamic property Q is a subtype of a property P, which could be either static or dynamic. It means that an individual x with the property P might have the property Q at some periods of time and might not have this property on some other occasions. For example, property *STUDENT* is a dynamic subtype of property *PERSON* as a *person* might be a *student* only for some periods of time. Similarly, properties *PART-TIME-STUDENT* and *FULL-TIME-STUDENT* are dynamic subtypes of property *STUDENT* and here too a *student* can change its status from being a *part-time-student* to being a *full-time-student,* and vice-versa, many times during the period of enrolment.

Using the above terminology let's define the semantics of the three meta-properties introduced by Guarino [16]: identity, rigidity and dependence.

2.2 Identity

Following the original definition, we say that *a property P has identity*, and denote it by +I, if there is a relation R such that

$$\forall_t\ \forall_{xy}\ (P_t(x) \wedge P_t(y)) \rightarrow (R(x,y) \leftrightarrow (x = y)).$$

The relation R is called an identity condition of P. For example, property *PERSON* has identity as its instances can be distinguished from each other by their DNA structure (relation *HAS-SAME-DNA*) or the makeup of their brains (relation *HAS-SAME-BRAIN*). On the other hand, individuals of property *THING* cannot be distinguished by any identity condition inherent to all things.

Further we say that *a property P has its own identity*, and denote it by +O, if there is a relation R such that

$$\forall_t \forall_{xy} (P_t(x) \wedge P_t(y)) \rightarrow (R(x,y) \leftrightarrow (x = y))$$

and

$$\forall_Q (\neg(Q < P)) \rightarrow (\neg(\forall_t \forall_{xy} (Q_t(x) \wedge Q_t(y)) \rightarrow (R(x,y) \leftrightarrow (x = y))).$$

In other words, a property P has its own identity, if it does not share its identity condition defined by the relation R with any property Q which is not a subsumption of the property P. For example, relation *HAS-SAME-DNA* allows us to distinguish between any two individuals having property *LIVING-BEING*, but it cannot be used to distinguish between individuals having a property being a supertype of *LIVING-BEING*. Thus, property *LIVING-BEING* has its own identity.

However, property *PERSON* shares its identity condition *HAS-SAME-DNA* with all individuals with property *LIVING-BEING*. And unless property *PERSON* introduces its own identity condition, we say that the property does not have its own identity, and denote it by –O.

Obviously, if a property has its own identity (+O) it also has identity (+I). Conversely, not having identity (-I) rules out its own identity, and therefore, implies lack of own identity (-O). This leaves only three possible identity criteria: -I-O, +I-O, and +I+O.

Let's explore under what conditions a subtype Q of a property P with given identity criteria can inherit or change the identity criteria of its supertype.

If a property P has identity, it means that there is an identity condition defined for its individuals. For any subtype property Q of P, individuals with property Q have also property P and, therefore, can be distinguished using the property P's identity condition. Thus, having identity is always inherited.

If a subtype property Q of P has its own identity, it becomes +I+O irrespective of what kind of identity criteria the supertype has. If, however, a subtype property Q of P does not have its own identity, it always becomes +I-O, unless its supertype does not have identity at all.

These subtyping rules for identity are summarized in TABLE I. The boxes with the phrase 'not allowed' signify the fact that identity cannot ever be lost in subtypes or acquired from a subtype, which does not have identity. The symbol $\mathbf{I}(\top)$ means that subtyping is allowed under the identity criteria, and the symbol $\mathbf{I}(\perp)$ means that subtyping is not allowed under the identity criteria.

TABLE I. Subtyping Rules for Identity.

$Q \leq P$	-I-O	+I-O	+I+O
-I-O	allowed $\mathbf{I}(\top)$	not allowed $\mathbf{I}(\perp)$	not allowed $\mathbf{I}(\perp)$
+I-O	not allowed $\mathbf{I}(\perp)$	allowed $\mathbf{I}(\top)$	allowed $\mathbf{I}(\top)$
+I+O	allowed $\mathbf{I}(\top)$	allowed $\mathbf{I}(\top)$	allowed $\mathbf{I}(\top)$

2.3 Rigidity

A property *P* is *rigid*, which is denoted by +R, if it is a static property. It means that all individuals with property *P* meet the rigidity condition

$$P_t(x) \rightarrow (\tau_{P(x)} = \tau_x).$$

A property is *non-rigid*, which is denoted by -R, if

$$\exists_{t'} \exists_{t''} \exists_x (P_{t'}(x) \wedge \omega_{t''}(x) \wedge \neg P_{t''}(x)).$$

It means that there is at least one individual that violates the rigidity condition at some point in time.

A property is *anti-rigid*, which is denoted by ~R, if

$$\exists_{t'} \exists_{t''} \forall_x (P_{t'}(x) \rightarrow (\omega_{t''}(x) \wedge \neg P_{t''}(x))).$$

Anti-rigidity is a special case of non-rigidity, in which all individuals holding a property *P* violate the rigidity condition, and as such is subsumed by non-rigidity.

A rigid property *P* can have a rigid subtype property Q, i*f Q* holds for its individuals for their entire lifetime. For example, property *LIVING-BEING* and its subtype *PERSON*. This amounts to static subtyping. But also, a subtype property *Q* of a rigid property *P* can be a result of dynamic subtyping, and then it can be either non-rigid or anti-rigid, depending whether some or all of its individuals violate the rigidity condition. Therefore, rigidity is not inherited.

It is possible to have a rigid proper subtype *Q* of a non-rigid property *P*, since the denotation $\delta Q_t \subset \delta P_t$ could contain only those individuals out of δP_t, which fulfill the rigidity condition. Equally, it is possible that a proper subtype *Q* of a non-rigid property *P* has the denotation $\delta Q_t \subset \delta P_t$ that contains only those individuals out of δP_t, which violate the rigidity condition, or contains some that do and some that do not. Hence, non-rigidity is not inherited.

Since all individuals that have an anti-rigid property *P* hold it for periods of time always shorter than their lifetimes, any subtype property *Q* of the property *P* cannot hold it longer than *P*'s lifetime. Therefore, all individuals of any subtype property *Q* must violate the rigidity condition, which means that anti-rigidity is inherited.

The summary of subtyping rules of the rigidity conditions is presented in TABLE II. The symbol **R**(T) means that subtyping is allowed under the rigidity criterion, and the symbol **R**(⊥) means that subtyping is not allowed under the rigidity criterion.

TABLE II. Subtyping Rules for Rigidity

Q* ≤ *P	+R	-R	~R
+R	allowed **R**(T)	allowed **R**(T)	not allowed **R**(⊥)
-R	allowed **R**(T)	allowed **R**(T)	not allowed **R**(⊥)
~R	allowed **R**(T)	allowed **R**(T)	allowed **R**(T)

2.4 Dependence

Following Guarino, we confine this discussion to one type of dependence that is "*notional dependence*, which holds for a property if its instances require instances of another property to exist" [16]. We say that a property P is *dependent*, which is denoted by +D, if

$$\forall_t \forall_x (P_t(x) \rightarrow \exists_{Q \neq P} \exists_{y \neq x} Q_t(y)).$$

In other words, for an individual x to have a property P ,it is required that there exists an individual having a property Q. In this definition, it is also assumed that Q is not a part of P. For example, individuals with property *PARENT* require individuals with property *CHILD* to exist.

A property P is *independent*, which is denoted by -D if it not dependent.

If a property Q is a subtype of a dependent property P, all individuals with the property Q also have the property P, and are therefore subject to the same dependency condition, and as such, are dependent. Thus, dependence is inherited by subtypes. On the other hand, an independent property P can have dependent or independent subtypes.

The summary of subtyping rules of the dependence criterion is presented in TABLE III. The symbol **D**(⊤) means that subtyping is allowed under the dependency criterion, and the symbol **D**(⊥) means that subtyping is not allowed under the dependency criterion.

TABLE III. Subtyping Rules for Dependence

$Q \leq P$	**+D**		**-D**	
+D	Allowed	**D**(⊤)	allowed	**D**(⊤)
-D	not allowed	**D**(⊥)	allowed	**D**(⊤)

3 Ontological Engineering Rules

All desirable combinations of the three property criteria give rise to eight property types (a metalevel classification of properties), which were described in [16]. Three of them are called Formal as they do not carry identity, and the remaining five are called sortals as they have either their own or an inherited identity. The property types are shown in TABLE IV. Note that we have split Category, Type and Merely Rigid Sortals according to their dependency criterion and Material Role according to its own identity criterion. This will enable us to have a closer look at subtyping allowed for those types of properties.

TABLE IV. Property Types Resulting from Property Criteria.

	Property Type	**Property Criteria**	**Examples**
F O R M A L	Category+ (**Cat+**)	-O-I+D+R	*SOCIAL-ENTITY*
	Category- (**Cat-**)	-O-I-D+R	*THING, LOCATION, ENTITY*
	Formal Role (**FRole**)	-O-I+D~R	*PART, PATIENT, ACTOR*
	Atribution (**Attr**)	-O-I-D-R	*MALE, RED*
S O R T A L S	Type-Atribution Mixing (**ATMix**)	-O+I-D-R	*MALE-PERSON, RED-FLOWER*
	Type+ (**Type+**)	+O+I+D+R	*ORGANIZATION*
	Type- (**Type-**)	+O+I-D+R	*LIVING-BEING*
	Phasal Sortal (**PSort**)	+O+I-D~R	*CATERPILAR*
	Material Role+ (**MRole+**)	+O+I+D~R	*STUDENT*
	Material Role- (**MRole-**)	-O+I+D~R	*FOOD*
	Merely Rigid Sortal+ (**MRSort+**)	-O+I+D+R	*LORD*
	Merely Rigid Sortal- (**MRSort-**)	-O+I-D+R	*INVERTEBRATE-ANIMAL*

TABLE V summarizes the allowed subsumptions between property types shown in TABLE IV. Cells contain three subsumption conditions originating from identity, dependence and rigidity criteria discussed earlier. These are shown in the table, as appropriate, for the respective property types. For example, at the intersection of Formal Attribution (**Attr** row) and Formal Role (**FRole** column) there are:

- **I**($\top$) indicating that the identity criterion allows the subsumption, as per TABLE I.
- **D**($\perp$) which indicates that the dependency criterion does not allow the subsumption, as per TABLE III.
- **R**($\perp$) which indicates that the rigidity criterion does not allow the subsumption, as per TABLE II; and since it is a conjunction of the three conditions, it follows that the subsumption is not allowed. All allowed subsumptions are highlighted by shading of the cells.

3.1 Guarino's ontology engineering rules.

Let us examine subtyping rules given in [16] and assess their coverage in the discussion so far:

G1. “*Antirigid class cannot subsume a rigid class.*” This rule has already been accounted for in the discussion of rigidity.

G2. “*ICs cannot be ‘overriden’ by a subclass, merely augmented*” (in other words, sortals cannot subsume formal concepts). This has already been accounted for in the discussion of identity.

G3. “*A dependent class cannot subsume an independent one*.” This has already been accounted for in the discussion of dependence.

G4. “*A material role can be subsumed by rigid sortals, since they carry identity ... They are the roles that normally specialise formal roles…* ” In the discussion so far, a material role is the most flexible of all the property types in terms what it can subsume (see TABLE V). Except for material roles without own identity, which cannot be subsumed by formal properties, all the other subsumptions are allowed. However, the cited rule makes good sense, since it would be desirable to have in the ontology some indication of what property the role applies to. Otherwise, the role specification would be somehow incomplete, like saying that there is a material role STUDENT without any indication that it applies to PERSON. But there is no reason to make the assumption that a rigid property must be subsumed by a rigid one; there could be some other material roles or even phased sortals between the material role and the subsuming rigid property in the subsumption hierarchy.

G5. “*Phased sortals must be subsumed by a type*.” This is required to be able to establish the identity of the changing entity. But since a merely rigid, independent sortal, type-attribution mixing, material role and phased sortal should all be subsumed by a type, all of them can directly subsume a phased sortal.

G6. “*Merely rigid sortals ... are always subsumed by at least one type*.” This is to provide an identity condition, as merely rigid sortals do not carry their own identity condition.

G7. “*Types can only be subsumed by categories and strictly non-rigid formal attributions*.” In other words, types sit between formal properties and the rest of the sortals in the subsumption hierarchy. In fact, it has already been said that the other sortals should be eventually subsumed by a type. Of course, types can form an hierarchy and subsume each other before being subsumed by a category or formal attribution.

TABLE VI summarizes all the conditions discussed so far, and shows when subtyping of properties of different types is allowed, as indicated by the content of respective cells. If “not allowed” is written in a cell, it means that there is a restriction on the subsumption originating from TABLE V. In cases where “G(*n*) not allowed” is written, it indicates that one of the Guarino’s rule *n* has been invoked to prohibit this subsumption. For example, a formal attribution (**Attr** row) cannot be subsumed by a formal role (**FRole** column) since the cell on the intersection of the two contains phrase “not allowed” derived from TABLE V.

TABLE V. Conditions for Subtyping Originating from the Three Property Criteria.

		F O R M A L				S O R T A L S							
	⊆	**Cat+** O-I+D+R	**Cat-** -O-I-D+R	**FRole** O-I+D~R	**Attr** O-I-D-R	**ATMix** O+I-D-R	**Type+** O+I+D+R	**Type-** O+I-D+R	**PSort** O+I-D~R	**MRole+** O+I+D~R	**MRole-** O+I+D~R	**MRSort+** +I+D+R	**MRSort-** O+I-D+R
F O R M A L	**Cat+** -O-I+D+R	**I**(T)**D**(T) **R**(T)	**I**(T)**D**(T) **R**(T)	**I**(T)**D**(T) **R**(⊥)	**I**(T)**D**(T) **R**(T)	**I**(⊥)**D**(T) **R**(T)	**I**(⊥)**D**(T) **R**(T)	**I**(⊥)**D**(T) **R**(T)	**I**(⊥)**D**(T) **R**(⊥)	**I**(⊥)**D**(T) **R**(⊥)	**I**(⊥)**D**(T) **R**(⊥)	**I**(⊥)**D**(T) **R**(T)	**I**(⊥)**D**(T) **R**(T)
	Cat- -O-I-D+R	**I**(T)**D**(⊥) **R**(T)	**I**(T)**D**(T) **R**(T)	**I**(T)**D**(⊥) **R**(⊥)	**I**(T)**D**(T) **R**(T)	**I**(⊥)D(T) **R**(T)	**I**(⊥)**D**(⊥) **R**(T)	**I**(⊥)**D**(T) **R**(T)	**I**(⊥)**D**(T) **R**(⊥)	**I**(⊥)**D**(⊥) **R**(⊥)	**I**(⊥)**D**(⊥) **R**(⊥)	**I**(⊥)**D**(⊥) **R**(T)	**I**(⊥)**D**(T) **R**(T)
	FRole -O-I+D~R	**I**(T)**D**(T) **R**(T)	**I**(T)**D**(T) **R**(T)	**I**(T)D(T) **R**(T)	**I**(T)**D**(T) **R**(T)	**I**(⊥)**D**(T)**R**(T)	**I**(⊥)**D**(T) **R**(T)	**I**(⊥)**D**(T) **R**(T)	**I**(⊥)**D**(T) **R**(T)	**I**(⊥)**D**(T) **R**(⊥)	**I**(⊥)**D**(T) **R**(⊥)	**I**(⊥)**D**(T) **R**(T)	**I**(⊥)**D**(T) **R**(T)
	Attr -O-I-D-R	**I**(T)**D**(⊥) **R**(T)	**I**(T)**D**(T) **R**(T)	**I**(T)**D**(⊥) **R**(⊥)	**I**(T)**D**(T) **R**(T)	**I**(⊥)**D**(T) **R**(T)	**I**(⊥)**D**(⊥) **R**(T)	**I**(⊥)**D**(T) **R**(T)	**I**(⊥)**D**(T) **R**(⊥)	**I**(⊥)**D**(⊥) **R**(⊥)	**I**(⊥)**D**(⊥) **R**(⊥)	**I**(⊥)**D**(⊥) **R**(T)	**I**(⊥)**D**(T) **R**(T)
S O R T A L S	**ATMix** -O+I-D-R	**I**(⊥)**D**(⊥) **R**(T)	**I**(⊥)**D**(T) **R**(T)	**I**(⊥)**D**(⊥) **R**(⊥)	**I**(⊥)**D**(T) **R**(T)	**I**(T)**D**(T) **R**(T)	**I**(T)**D**(⊥) **R**(T)	(T)**D**(T) **R**(T)	**I**(T)**D**(T) **R**(⊥)	**I**(T)**D**(⊥) **R**(⊥)	**I**(T)**D**(⊥) **R**(⊥)	**I**(T)**D**(⊥) **R**(T)	**I**(T)**D**(T) **R**(T)
	Type+ +O+I+D+R	**I**(T)**D**(T) **R**(T)	**I**(T)**D**(T) **R**(T)	**I**(T)**D**(T) **R**(⊥)	**I**(T)**D**(T) **R**(T)	**I**(T)**D**(T) **R**(T)	**I**(T)**D**(T) **R**(T)	(T)**D**(T) **R**(T)	**I**(T)**D**(T) **R**(⊥)	**I**(T)**D**(T) R(⊥)	**I**(T)**D**(T) **R**(⊥)	**I**(T)**D**(T) **R**(T)	**I**(T)**D**(T) **R**(T)
	Type- +O+I-D+R	**I**(T)**D**(⊥) **R**(T)	**I**(T)**D**(T) **R**(T)	**I**(T)**D**(⊥) **R**(⊥)	**I**(T)**D**(T) **R**(T)	**I**(T)**D**(T) **R**(T)	**I**(T)**D**(⊥) **R**(T)	(T)**D**(T) **R**(T)	**I**(T)**D**(T) **R**(⊥)	**I**(T)**D**(⊥) **R**(⊥)	**I**(T)**D**(⊥) **R**(⊥)	**I**(T)**D**(⊥) **R**(T)	**I**(T)**D**(T) **R**(T)
	PSort +O+I-D~R	**I**(T)**D**(⊥) **R**(T)	**I**(T)**D**(T) **R**(T)	**I**(T)**D**(⊥) R(T)	**I**(T)**D**(T) **R**(T)	**I**(T)**D**(T) **R**(T)	**I**(T)**D**(⊥) **R**(T)	(T)**D**(T) **R**(T)	**I**(T)**D**(T) **R**(T)	**I**(T)**D**(⊥) **R**(T)	**I**(T)**D**(⊥) **R**(T)	**I**(T)**D**(⊥) **R**(T)	**I**(T)**D**(T) **R**(T)
	MRole+ +O+I+D~R	**I**(T)**D**(T) **R**(T)	**I**(T)**D**(T) **R**(T)	**I**(T)**D**(T) **R**(T)	**I**(T)**D**(T) **R**(T)	**I**(T)**D**(T) **R**(T)	**I**(T)**D**(T) **R**(T)	(T)**D**(T) **R**(T)	**I**(T)**D**(T) **R**(T)	**I**(T)**D**(T) **R**(T)	**I**(T)**D**(T) **R**(T)	**I**(T)**D**(T) **R**(T)	**I**(T)**D**(T) **R**(T)
	MRole- -O+I+D~R	**I**(⊥)**D**(T) **R**(T)	**I**(⊥)**D**(T) **R**(T)	**I**(⊥)**D**(T) **R**(T)	**I**(⊥)**D**(T) **R**(T)	**I**(T)**D**(T) **R**(T)	**I**(T)**D**(T) **R**(T)	(T)**D**(T) **R**(T)	**I**(T)**D**(T) **R**(T)	**I**(T)**D**(T) **R**(T)	**I**(T)**D**(T) **R**(T)	**I**(T)**D**(T) **R**(T)	**I**(T)**D**(T) **R**(T)
	MRSort+ -O+I+D+R	**I**(⊥)**D**(T) **R**(T)	**I**(⊥)**D**(T) **R**(T)	**I**(⊥)**D**(T) **R**(⊥)	**I**(⊥)**D**(T) **R**(T)	**I**(T)**D**(T) **R**(T)	**I**(T)**D**(T) **R**(T)	(T)**D**(T) **R**(T)	**I**(T)**D**(T) **R**(⊥)	**I**(T)**D**(T) **R**(⊥)	**I**(T)**D**(T) **R**(⊥)	**I**(T)**D**(T) **R**(T)	**I**(T)**D**(T) **R**(T)
	MRSort- -O+I-D+R	**I**(⊥)**D**(⊥) **R**(T)	**I**(⊥)**D**(T) **R**(T)	**I**(⊥)**D**(⊥) **R**(⊥)	**I**(⊥)**D**(T) **R**(T)	**I**(T)**D**(T) **R**(T)	**I**(T)**D**(⊥) **R**(T)	(T)**D**(T) **R**(T)	**I**(T)**D**(T) **R**(⊥)	**I**(T)**D**(⊥) **R**(⊥)	**I**(T)**D**(⊥) **R**(⊥)	**I**(T)**D**(⊥) **R**(T)	**I**(T)**D**(T) **R**(T)

TABLE VI. Summary of Subtyping Conditions.

		F O R M A L				S O R T A L S							
	⊆	**Cat+** O-I+D+R	**Cat-** O-I-D+R	**FRole** O-I+D~R	**Attr** O-I-D-R	**ATMix** O+I-D-R	**Type+** O+I+D+R	**Type-** O+I-D+R	**PSort** O+I-D~R	**MRole+** O+I+D~R	**MRole-** O+I+D~R	**RSort+** O+I+D+R	**RSort-** O+I-D+R
FORMAL	**Cat+** -O-I+D+R	allowed	allowed	not allowed	allowed	not allowed	not allowed	not allowed	not allowed	not allowed	not allowed	not allowed	not allowed
	Cat- -O-I-D+R	not allowed	allowed	not allowed	allowed	not allowed	not allowed	not allowed	not allowed	not allowed	not allowed	not allowed	not allowed
	FRole -O-I+D~R	allowed	allowed	allowed	allowed	not allowed	not allowed	not allowed	not allowed	not allowed	not allowed	not allowed	not allowed
	Attr -O-I-D-R	not allowed	allowed	not allowed	allowed	not allowed	not allowed	not allowed	not allowed	not allowed	not allowed	not allowed	not allowed
SORTALS	**ATMix** -O+I-D-R	not allowed	not allowed	not allowed	not allowed	allowed	not allowed	allowed	not allowed	not allowed	not allowed	not allowed	allowed
	Type+ +O+I+D+R	allowed	allowed	not allowed	allowed	(G7) not allowed	allowed	allowed	not allowed	not allowed	not allowed	(G7) not allowed	(G7) not allowed
	Type- +O+I-D+R	not allowed	allowed	not allowed	allowed	(G7) not allowed	not allowed	allowed	not allowed	not allowed	not allowed	not allowed	(G7) not allowed
	PSort +O+I-D~R	not allowed	(G5) not allowed	not allowed	(G4) not allowed	allowed	not allowed	allowed	allowed	not allowed	not allowed	not allowed	allowed
	MRole+ +O+I+D~R	(G4) not allowed	(G4) not allowed	allowed	(G4) not allowed	(G4) not allowed	allowed	allowed	allowed	allowed	allowed	allowed	allowed
	MRole- -O+I+D~R	not allowed	not allowed	not allowed	not allowed	(G4) not allowed	allowed	allowed	allowed	allowed	allowed	allowed	allowed
	MRSort+ -O+I+D+R	not allowed	not allowed	not allowed	not allowed	allowed	allowed	allowed	not allowed	not allowed	not allowed	allowed	allowed
	MRSort- -O+I-D+R	not allowed	not allowed	not allowed	not allowed	allowed	not allowed	allowed	not allowed	not allowed	not allowed	not allowed	allowed

4 Using Property Types in Conceptual Modeling

In knowledge bases [17], [18] and databases [19], [20], there is a commonly accepted distinction between definitions of terms, their characteristics and the relations between them (often referred to as a schema or the TBox), and assertions about the world stated in instances of those terms (often referred to as a fact base or the Abox). This section will discuss implications of the taxonomy of properties for defining domain models, i.e. classes, their characteristics and the relations between them (the TBox).

In conceptual modeling of an application domain, not only properties are given, but also all the sufficient and necessary conditions for an individual to carry the specific property are defined. These property definitions are typically called classes. Individuals that hold the property defined by a class are called instances of the class. In order to maintain the distinction between properties and classes, the former will be denoted by all capital (hyphenated) names (e.g. *LIVING-BEING*), and the latter by non-hyphenated names with the first letter of each word capitalized (e.g. LivingBeing). Classes based on properties will have corresponding names consisting of the same wording.

A class defines domain specific conditions for the class membership. The conditions give characteristics that are pertinent to all the class members in the context of the domain and the purpose for which the modeling is performed. Properties of type Attribution seems to provide appropriate terminology for specifying class characteristics. However, property type Attribution, as explained in [16], is

> “*refering … to an attribution as the property of having an attribute with certain value, i.e. having gender MALE or color RED*.”

And therefore, while useful to describe characteristic of individuals, attributions are not suitable for declaring that class Person should have gender stated for its instances. This requires a relation, which maps instances of the class to a set of attributions relevant to the characteristic being defined. The sets of attributions will be called attribution types. This brings us to the first relation type of conceptual modeling, which has the following format:

Characteristic(Class, AttributionType).

For class Person, this relation type can be instantiated as

Gender(Person, {“*Male*”, “*Female*”}) – enumerated attribution type

Height(Person, {x: x *is distance measurement in meter*s}) – measured attribution type

Address(Person, {x: x *is a character string*}) – description attribution type

Dependent properties come into existence in relation to some other properties holding over different individuals. For example, *STUDENT* is a property of a person being enrolled for a university course, or *BANK-CUSTOMER* is a property of a person having a bank account. The statements have the following elements in them:

- they tell us that *STUDENT* and *BANK-CUSTOMER* are subtypes of property *PERSO*N, and

- they also point out what properties *STUDENT* and *BANK-CUSTOMER* are dependent on; in this case the properties depend on the existence of respectively, a *UNIVERSITY-COURSE* and of a *BANK-ACCOUNT.*

It is easy to show that there are relationships between classes based on these properties by creating pairs (Student, UniversityCourse) and (BankCustomer, BankAccount). What remains to be answered is what type of relationships do they represent.

In order to answer this question, it is useful to call upon Charles Peirce's three basic categories Firstness, Secondness and Thirdness which were described in the quotation from the original provided in [1]:

"First is the conception of being or existing independent of anything else. Second is the conception of being relative to, the conception of reaction with, something else. Third is the conception of mediation, whereby a first and a second are brought into relation. (1891)"

Obviously, independent properties are related to the Firstness. Dependent properties relate to the Secondness. The Thirdness, which mediates the relationship, provides an answer to the original question. The mediating element that brings *STUDENT* and *UNIVERSITY-COURSE* together is *ENROLMEN*T, and the one that brings *BANK-CUSTOMER* and *BANK-ACCOUNT* together is *BANK-ACCOUNT-CONTRAC*T. In general, dependent properties will result in a conceptual relation type defined as follows:

DependencyMediator(DependentClass, Class)

which for the two examples is instantiated as:

Enrolment(Student, UniversityCourse)
BankAccountContract(BankCustomer, BankAccount).

The only other issue here is what property type DependencyMediator is based on. Mediating properties seems to be rigid, dependent and carry their own identity, and therefore they are of type **Type**+.

Another type of relationships that can exist between properties is the structural relationship. In this case, the relations bring about certain arrangement of individuals. For example, PartOf(Engine, Car), Above(Roof, Basement). This type of relationship can be defined as

StructuralDependency(Class, Class).

Structural dependencies are properties that seem to be both dependent and non-rigid. It is the type of properties, which are not included in the taxonomy as they are "*too weak … to capture the rigor … intende*d" [16]. However, if a structural dependency is combined with one of the related properties, for example, *PART-OF-CAR* or *ABOVE-BASEMEN*T, it becomes independent and non-rigid and therefore is of type Attribution.

5 Conclusion

This paper critically reviews the application of ontological engineering to conceptual modeling. The two activities, while related, differ in purpose. Ontological engineering deals with rules and principles of conceptualization of a problem domain. Conceptual modeling, on the other hand, is concerned with the problem domain structuring. While one may accept that no structure exists without underlying conceptualization, the requisite conceptualization is usually implicit and often derived on an intuitive, craft basis. Manifestation of an explicit basis for conceptualization will, in our opinion, not only result in IT solutions of greater integrity, but will also facilitate system and data integration vital to enterprise application integration, including semantic web applications.

In the first part, the paper presented denotational semantics of identity, rigidity and dependence used to define types of properties, and then introduced a set of property subtyping rules and checks. It established the foundations for creating clean, well-structured taxonomies. In the second part, the links between ontology engineering principles and conceptual modeling were explored. In particular, three types of emergent relationships were discussed.

Further work will concentrate on the development of domain ontologies as the basis for data and system integration for oceanographic data interchange and electronic business processes. Parallel work on the rules for context management and role constraints will also benefit from the explication of property subtyping rules and constraints.

References

1. Sowa, J.F., *Knowledge Representation*. 2000, Pacific Grove, CA USA: Brooks/Cole. 593.
2. Connolly, D., *Extensible Markup Language (XML)*. 1997, W3C.
3. Object Management Group, *XMI Metadata Interchange (XMI)*. 1999, OMG.
4. Jacobson, I., G. Booch, and J. Rumbaugh, *The Unified Software Development Process*. Object Technology Series. 1999, Reading, MA: Addison-Wesley.
5. Rumbaugh, J., I. Jacobson, and G. Booch, *The Unified Modelling Language Reference Manual*. Object Technology Series. 1999, Reading, MA: Addison-Wesley.
6. Booch, G., J. Rumbaugh, and I. Jacobson, *The Unified Modelling Language User Manual*. Object Technology Series. 1999, Reading, MA: Addison-Wesley.
7. Zhu, X., et al., *Ontology-Based Web Site Mapping for Information Exploration*, in *Proceedings of the Eighth International Conference on Information Knowledge Management*. 1999: Kansas City, MO USA. p. 188-194.
8. Embley, D.W., et al., *Ontology-based Extraction and Structuring of information from Data-Rich Unstructured Documents*, in *Proceedings of the 1998 ACM 7th International Conference on Information and Knowledge Management*. 1998: Washington, USA. p. 52-58.
9. Valente, A., et al., *Building and (Re)Using an Ontology of Air Campaign Planning.* IEEE Inteligent Systems, 1999. **14**(1 (Jan/Feb 1999)): p. 27-36.
10. Keyser, D., *Ontologically Yours*, in *Lectures in Artificial Intelligence*, M.-L. Mugnier and M. Chein, Editors. 1998, Springer-Verlag. p. 35-47.
11. Sowa, J.F., *Using a Lexicon of Canonical Graphs in a Semantic Interpreter*, in *Relational Models of the Lexicon*, M.W. Evens, Editor. 1988, Cambridge University Press: Cambridge.
12. Strawson, P.F., *Individuals. An Essay on Descriptive Metaphysics.* 1959, London: Routledge.
13. Griffin, N., *Relative Identity*. 1977, Oxford: Oxford University Press.
14. Guarino, N., M. Carrara, and P. Giaretta, *An Ontology of Meta-Level Categories*, in *Principles of Knowledge Representation and reasoning: Proceedings of the Fourth International Conference (KR'94)*, J. Doyle, E. Sandewall, and P. Torasso, Editors. 1994, Morgan Kaufman Publishers: San Francisco, CA.
15. Guarino, N., *Formal Ontology, Conceptual Analysis and Knowledge Representation.* International Journal of Human-Computer Studies, 1995. **43**(5/6): p. 625-640.

16. Guarino, N. and C. Welty, *A Formal Ontology of Properties*, in *Proceedings of EKAW-2000: The 12th International Conference on Knowledge Engineering and Knowledge Management*, R. Dieng and O. Corby, Editors. 2000, Springer-Verlag. p. 97-112.
17. Brachman, R.J., *An Overview of the KL-ONE Knowledge Representation System.* Cognitive Science, 1985. **9**(2): p. 171-216.
18. Levesque, H.J. and R.J. Brachman, *Knowledge Level Interfaces to Information Systems,* in *On Knowledge Base Management Systems: Integrating Artificial Intelligence and Database Technologies*, M.L. Brodie and J. Mylopoulos, Editors. 1986, Springer-Verlag: New York.
19. Abrial, J.R., *Data Semantics,* in *Data Base Management*, J.W. Klimbie and K.L. Koffeman, Editors. 1974, Noth-Holland Publishing: Amsterdam. p. 1-59.
20. Mylopoulos, H.J. and P.A. Bernstein, *A Language Facility for Designing Database-Intensive Applications.* ACM Journal on Database Systems, 1980. **5**(2): p. 185-207.