April 2017

# Integrating GoodRelations in a Domain-Specific Ontology

Andrea Westerinen, Rebecca Tauber
Nine Points Solutions, LLC

**Abstract.** GoodRelations has existed since 2008 to aid businesses in describing their products and services. It provides a variety of definitions and terminology useful for improving a company's online retail presence. In this paper, we go beyond using GoodRelations simply for structured data in web pages, and combine it with a domain-specific ontology. The goals of the work are threefold, to define a set of retail-oriented Ontology Design Patterns (ODPs) that makes it easier to understand and reuse GoodRelations, to show how ontologies can be integrated with GoodRelations, and to illustrate several retail applications enabled by the use of ontologies. The paper presents a climbing gear ontology which is integrated with GoodRelations to define a set of retail ODPs and enable creating customer- and business-oriented solutions that can be used in several scenarios.

**Keywords**: GoodRelations Ontology, schema.org, Semantic Interoperability, Ontology Design Pattern, Domain-Specific Ontology, Data Reuse

## 1. Introduction

According to the U.S. Department of Commerce, online retail sales in the United States totaled over $291 million USD for the first three quarters of 2016 ("U.S. Census Bureau News, Quarterly Retail E-Commerce Sales", 2016). Search Engine Optimization (SEO) is one of the keys to online retail, dealing with design and documentation techniques that make a web site both user- and search engine-friendly. The focus of this project is to improve search results via the use of schema.org ("Schema", n.d.) and to enhance business opportunity and a customer's online or in-store experience via the use of ontologies.

Search results are improved by embedding "structured data" markup within a web site's pages, structured by the semantics of the schema.org vocabulary (also referred to as "Schema"). In an analysis done by Google in 2015, 31.3% of a sample of 10 billion web pages included Schema markup, with an average of six entities described per page (Guha, Brickley & Macbeth, 2015). Web pages are clearly utilizing this vocabulary, but it is still a small number compared to the huge number of pages that exist. Many, especially smaller, companies do not provide structured data at all. A study by Searchmetrics in 2014 found that under 1% of all web sites leveraged schema.org ("Schema.org analysis 2014", 2014). The businesses providing these web sites potentially lose customers because they rank lower in online searches. There are multiple reasons for this lack of

use: those companies may not know that Schema exists, or do not understand how to use it effectively.

Schema integrated concepts from GoodRelations, an OWL ontology, in 2012 to simplify the descriptions of products, offers, services and other related information for e-commerce (Guha, 2012). The importance of this integration can be seen in the Searchmetrics 2014 study that showed that over a third of all Google search results included information derived from Schema. The same study revealed that the second and third most popular concepts from Schema were "Offers"[1] and "Products". Both of these concepts come from GoodRelations.

GoodRelations has existed since 2008 to aid businesses in describing their products and services. It provides a variety of definitions and terminology useful for improving a company's online retail presence. But, it takes some experience to understand how to use GoodRelations. For this reason, this paper defines a set of simple Ontology Design Patterns (ODPs) for describing companies, products and offers. Beyond just reusing the patterns themselves, it is also possible to reuse the individuals (companies, product types and specific product details, etc.) within an industry. This second kind of reuse reduces the overall time and expertise required to create the ontologies, as the effort can be shared among multiple retailers.

Although describing products and offers in a standard way is valuable, one can go beyond simply adding structured data to web pages. In this paper, Schema/GoodRelations concepts are combined with, and enhanced by, a domain-specific ontology. The goals of the work are to show how ontologies can be integrated with Schema, and to illustrate the value of the integration by describing several, concrete, retail solutions. The work expands on several topics discussed in the Ontology Summit 2014 Communiqué (Gruninger, Obrst et al., 2014), specifically:

- The role that ontologies can play in applications
- Engineering of ontologies to address reuse and domain-specific modeling concerns

The paper is structured as follows: Section 2 is an overview of the GoodRelations components of Schema, while Section 3 is an overview of Ontology Design Patterns in general and the authors' proposed retail patterns. Section 4 discusses the development of an ontology for climbing gear (the Retail Ontology for Climbing, ROC), and its integration of GoodRelations. Section 5 describes various constraints and inferences made possible with the use of ontologies. Section 6 concludes the paper with details on future work.

[1] The concept of "Offer" from GoodRelations provides the details on products and/or services that are made available for a specific time, at a certain price, etc,

## 2. GoodRelations Ontology in schema.org

As explained above, the GoodRelations Ontology was incorporated into Schema in 2012. Some concepts were integrated using different terminology (for example, *schema:Product* is equivalent to *gr:ProductOrService*), but the semantics behind the concepts remain roughly equivalent.

The following list defines the main concepts of interest in this paper and provides a brief description of each one and its usage. Both the Schema and GoodRelations terms are provided under the respective namespaces "schema" and "gr". This convention is continued in the subsequent sections.

- *schema:Product* / *gr:ProductOrService* – the general concept of something offered for sale
    - *ProductOrService* can be an individual, a collection or a "model" as seen in the next bulleted items, and has many data and object properties such as description, condition, identifying codes, etc.
- *schema:ProductModel* / *gr:ProductOrServiceModel* – a kind of *gr:ProductOrService* that is a "datasheet or vendor specification … a prototypical description"
- *schema:IndividualProduct* / *gr:Individual* – a subclass of *gr:ProductOrService* that represents a single, identifiable instance
- *schema:SomeProducts* / *gr:SomeItems* – a subclass of *gr:ProductOrService* that represents a collection of multiple objects of the same model
- *schema:Brand* / *gr:Brand* – a label for a type of product or service
- *schema:QualitativeValue* / *gr:QualitativeValue* – a superclass for any enumerations (such as sizes being restricted to "S", "M", "L" or "XL")
    - Different *gr:QualitativeValue* individuals may be defined as equal to each other or ordered (for example, "S" is less than "M")
- *schema:QuantitativeValue* / *gr:QuantitativeValue* – a superclass for numerical values or intervals (such as height or duration)
    - *gr:QuantitativeValue* individuals may have properties of minimum and maximum values and units of measurement
- *schema:Organization* and *schema:Person* / *gr:BusinessEntity* – a legal entity per commercial law
    - *gr:BusinessEntity* has properties such as mailing address and contact information, as well as sale locations/areas served, identifying information (such as a Dun & Bradstreet number), etc.

There are many additional concepts and properties that could be discussed. One example is *schema:Offer* (*gr:Offering*). "Offer" defines the terms and conditions (typically, a payment) for providing/acquiring a product or service. It is discussed further in Section 4.5 related to a retailer's inventory.

## 3. Retail Ontology Design Patterns

Schema.org can be used alone to improve a retailer's ranking in search results, by providing product, pricing and related information. GoodRelations can be used to provide more complex information, and to reason and infer new data from schema.org's information. And, all this information can be integrated with domain-specific ontologies to support a variety of applications (some examples of which are discussed below).

In Section 4, the specific integration of GoodRelations with an ontology for classifying and understanding climbing gear is described. But, before delving into the integration details, this section outlines a set of general ODPs for retail.

An ontology describes the concepts, relationships, properties, axioms and individuals of a domain, whereas an ODP is focused on a specific and commonly recurring modeling problem. In this paper, both the semantics of a set of retail ODPs and their rendering using OWL 2 ("OWL 2", 2012) are discussed. For the retail ODPs, OWL 2 was chosen because it is built on RDF, which is an underpinning of the Linked Data and schema.org environments.

The retail ODPs were created to allow a set of merchants to consistently describe their products (and instances of products) across companies and individual stores. Consistency allows a customer to better compare different products and offers, thereby improving the customer experience. The adoption of the ODPs also spreads the cost of development across multiple retailers. Product descriptions are coupled with a domain-specific ontology that defines how the products are related and used, with the goal of further improving the customer experience.

To start using the retail ODPs, the various manufacturers and products in a particular retail domain need to be described. First, the manufacturers and their brands are defined, and then their various product models are detailed. The relationships between products are also classified by the domain ontology. Lastly, retail inventories and individual personal collections/purchases can be assembled.

This approach organizes an industry's (domain's) concepts into a hierarchy of modular ontologies. The ontologies "higher" in the hierarchy are more general, have few instances and change less frequently. The top-most ontology in the hierarchy describes the kinds of products in the domain as OWL 2 classes. New classes are needed only when new categories of products are developed.

The next level in the hierarchy defines the industry's manufacturers and brands. These can evolve, but change relatively infrequently. The "Manufacturers and Brands" ontology (and all ontologies lower in the hierarchy) are also defined using OWL 2, but these

ontologies are only collections of ABox statements[2] (based on concepts from higher in the hierarchy). Next, product instances (provided by the manufacturers) are specified. Lastly, collections of products (inventories and personal purchases) are defined. The collections would be the most-often changed, with instances and their properties being modified and extended frequently. Putting the collections at the bottom of the hierarchy allows this flexibility without the need to edit any higher-level data sets. Also, these bottom-level ontologies can easily reuse the manufacturers, models and domain concepts so that these instances are consistent and not continuously redefined.

*Figure 1* shows how the ontologies are organized in the hierarchy.

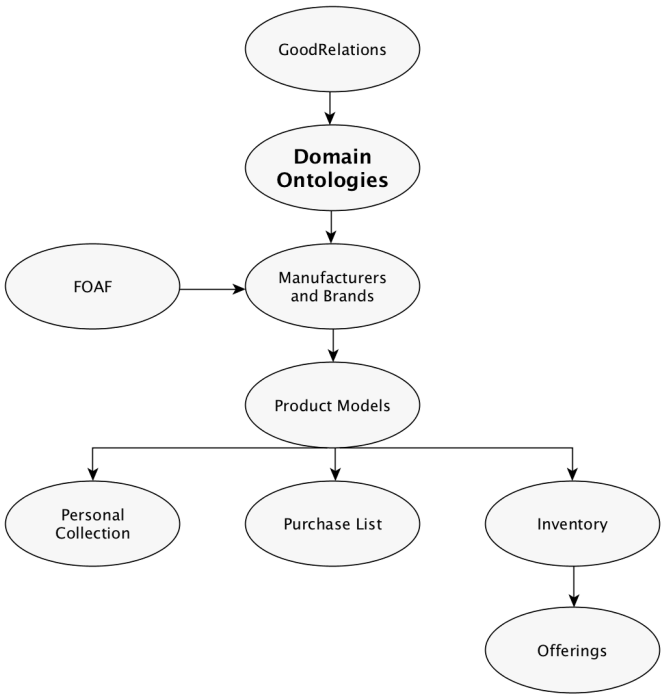


**Figure 1. Concepts in a Retail Ontology Design Patterns**

## 4. Retail Ontology for Climbing and GoodRelations Integration

The Retail Ontology for Climbing (ROC) was originally proposed as a way to test our methods for knowledge engineering and ontology reuse and integration. However, once initial development was complete, we worked with several retailers to expand it to address better managing their inventory and pricing strategies, and to assist customers

[2] An ABox is a statement of fact about an individual, such as "John is a Person". ABox statements describe individuals (instances) using the concepts (terminology) of the TBox. In ROC, the domain ontologies are definitions of the TBox.

regarding their gear and purchases. These use cases are discussed in more detail in the following sections.

Although ROC is specific to the climbing gear retail industry, its use of GoodRelations implements the ODPs described in Section 3. Note that the prefix, "roc", is used when referring to any class or property from the Retail Ontology for Climbing.

## *4.1. Climbing Domain Ontology*

ROC's top-most ontology defines categories of climbing products. It imports and extends the GoodRelations Ontology and specifies a product class hierarchy under a *roc:ClimbingGear* superclass, as shown in *Figure 2*. The superclass is used to anchor the classification and distinguish climbing gear from other products and services that a retailer may offer. For example, many retailers selling climbing equipment also sell a wide variety of outdoor gear and may also offer training classes as a service. Therefore, other possible superclasses (peers of *roc:ClimbingGear*) could be other ontologies' *Clothing*, *BikingGear* or *Training* classes. These product hierarchies could easily be integrated with ROC, all reusing and building on the retail ODPs.

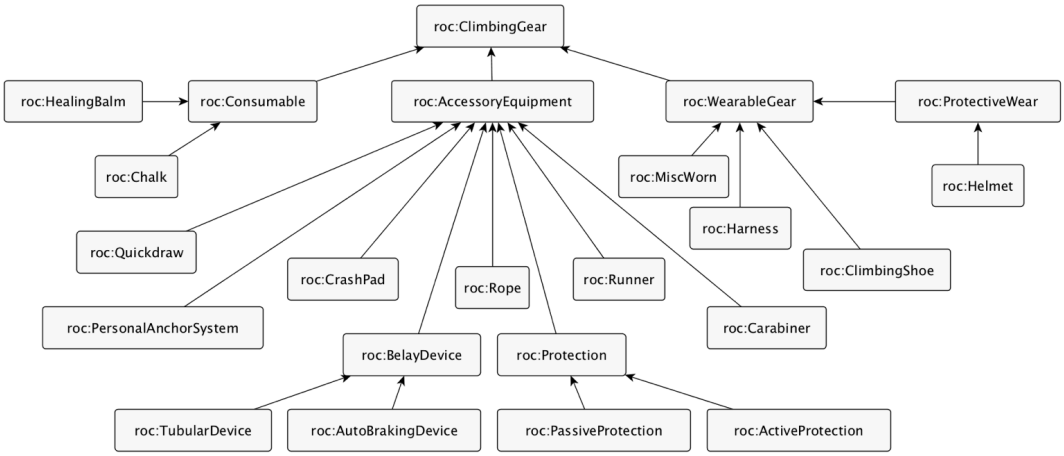


**Figure 2. Classes in the Retail Ontology for Climbing**

ROC also defines properties, as shown in *Figure 3*, specific to the climbing industry. All the properties are functional[3], as indicated by an asterisk in the diagram. For example, a carabiner is used for fastening ropes, among other purposes, and may have a lock on its gate. Hence, a specific locking property is defined for the *roc:Carabiner* class, with a Boolean range. Other properties such as condition and size are specified using enumerations which are subclasses of *gr:QualitativeValue*.

[3] A functional property is defined as having a single value for each instance where it is applied.

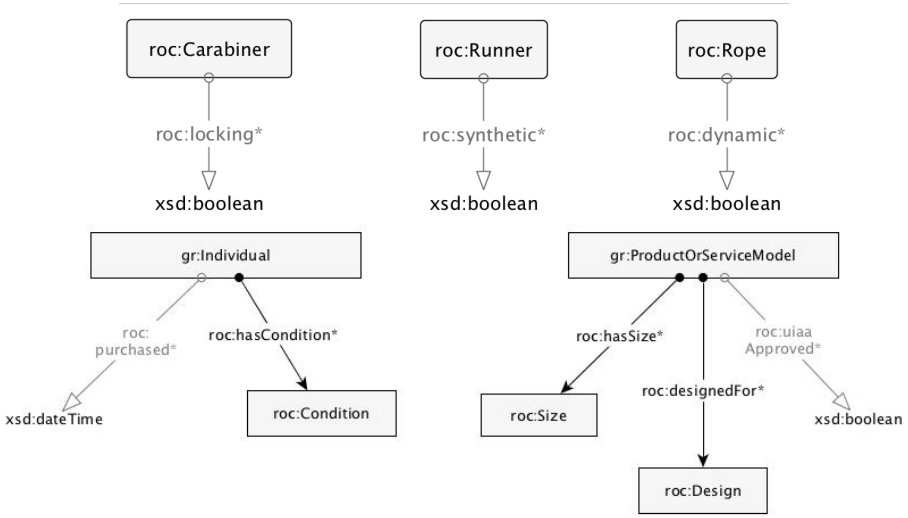


**Figure 3. Properties in the Retail Ontology for Climbing**

The details of the *roc:Size*, *roc:Usage* and *roc:Condition* enumerations are shown in *Figure 4*. Each enumeration is defined as a class restricted to a specific set of individuals (using an *owl:oneOf* restriction) which details all of the allowed values. Therefore, the value of a property using any of those enumerations must be one of those individuals.

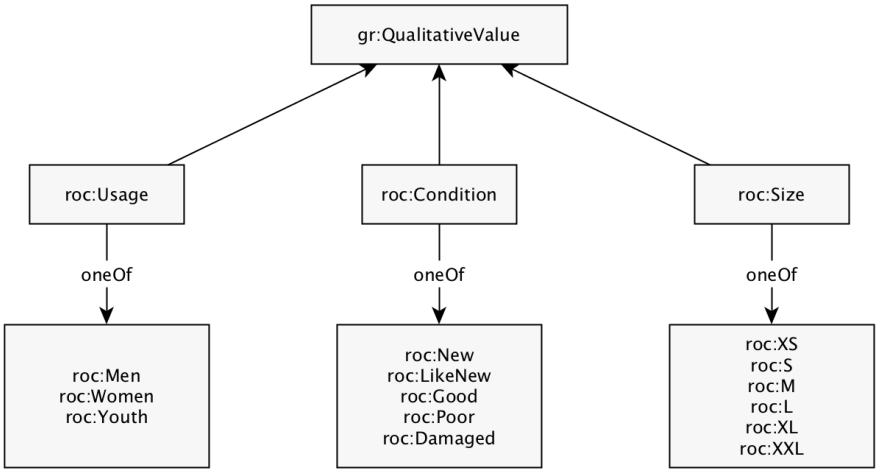


**Figure 4. Enumerations in the Retail Ontology for Climbing**

It is valuable to discuss the *roc:Condition* enumeration further, because the *roc:hasCondition* property appears to duplicate the semantics of the similarly named *gr:condition* property. The *roc:Condition* enumeration was defined to address a specific application - to recommend replacement gear from a personal collection based on its condition. The *gr:condition* property was not used because it has a literal range.

Although the definition of *roc:Condition* can be mapped to a literal, this is one example of how relationships/properties in GoodRelations may need to be revised or updated to meet specific requirements. Allowing arbitrary, free-form text was deemed problematic

in ROC because it would be difficult to understand and reason over the intended semantics. Therefore, to consistently and accurately represent condition, a custom enumeration and property were defined.

In addition to *gr:condition*, Schema offers an *itemCondition* property with a range of *schema:OfferItemCondition*. *schema:OfferItemCondition* is also a closed enumeration with the members, *schema:NewCondition*, *schema:RefurbishedCondition*, *schema:UsedCondition*, and *schema:DamagedCondition* ("ItemCondition", n.d.). Schema's *Product* class (which GoodRelations' *Individual* subclasses) is a valid domain for *schema:itemCondition*. The reuse of *schema:OfferItemCondition* is certainly applicable to some domains, such as electronics where a retailer can sell refurbished goods, but we determined it was necessary to have more specific conditions for climbing gear and the personal use application. But, in case the Schema enumeration was used, the *owl:sameAs* property was employed to define equivalences between these two enumerations. For example, ROC's *New* and *Damaged* are equivalent to Schema's *NewCondition* and *DamagedCondition*, respectively, while ROC's *LikeNew* and *Good* map to Schema's *UsedCondition*.

## *4.2. Manufacturers Ontology*

The Manufacturers ontology imports the climbing domain ontology described in the previous section, which in turn imported GoodRelations. While the subclasses and properties of *roc:ClimbingGear* are not specifically referenced in the Manufacturers ontology, they are used in subsequent ontologies with respect to a manufacturer's products.

The Manufacturers ontology is the next level of the ODPs defined in Section 3. All of the companies defined in the ontology are types[4] of the *gr:BusinessEntity* class (Schema's *Organization*). For illustration, some of the major manufacturers of climbing gear, whose details are provided in the ontology, are shown in *Figure 5*.

[4] The word, "type" is used to describe an rdf:type declaration, which states that the instance is a member of the specified class.

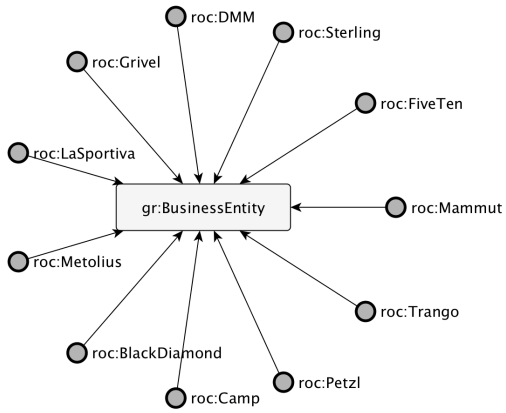


**Figure 5. Examples of Manufacturers in the Climbing Retail Market**

The following data is provided for each *gr:BusinessEntity*:

- Name (*schema:name* / *gr:name*)
- Address (*schema:address*, specifically a *schema:PostalAddress* and its related properties)
- Contact information (*schema:telephone*, *schema:faxNumber*, *schema:email*)
- Webpage URL (*schema:url* / *foaf:page*) ("FOAF (2000-2015+)", n.d.)
- Logo (*schema:logo* / *foaf:logo*)[5]
- DUNS number (*schema:duns* / *gr:hasDUNS*, where available)

Where a manufacturer sells products from brick-and-mortar shops, their store locations can also be defined to provide more complete information for a potential customer. This is mentioned here for completeness. In Section 4.4., we describe how a retailer lists their places of business. The same pattern can be used for a manufacturer's stores.

## *4.3. Product Model Ontology*

The Product Model Ontology imports the Manufacturers Ontology (which in turn imported the climbing domain ontology). All instances of product models are firstly instances of *schema:Product* / *gr:ProductOrServiceModel* - an "intangible entity that specifies some characteristics of a group of similar, usually mass-produced products, in the sense of a prototype" (Hepp, 2011). Beyond the Schema / GoodRelations class, all product models also are a member of one of the classes defined in the ROC climbing ontology, as graphed in *Figure 6*.

[5] In GoodRelations, a logo would be associated with a brand. But, for the particular business entities in the climbing industry, the manufacturers do not typically breakdown their products by brands. So, in this space, only the manufacturers are described.

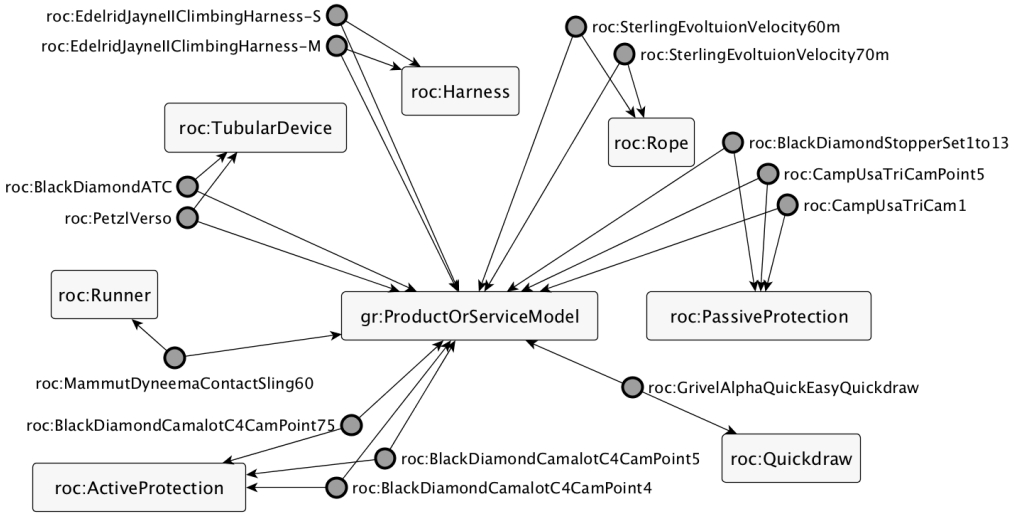


**Figure 6. Examples of Climbing ProductOrServiceModels**

Each product model includes the following properties, at a minimum, and may have additional detail both from GoodRelations and ROC, depending on the type of product being described:

- Name (*schema:name* / *gr:name*)
- Description (*schema:description* / *gr:description*)
- Manufacturer (*schema:manufacturer* / *gr:hasManufacturer*)
- Manufacturer part number (*schema:productID* / *gr:hasMPN*)
    - An identifier for the product unique for a manufacturer
- Additional properties (color, height, width, …)

As noted, "additional properties" are used to describe some climbing products, and may not apply in other domains. These properties include, but are not limited to, *gr:color, gr:weight, gr:width, gr: height*, and finally several custom properties from ROC, *roc:size* and *roc:usage*. Other than the custom properties, GoodRelations has a variety of properties that can be used to describe products and services, but only a few have been implemented due to difficulties in determining their values. For example, 13-digit UPC codes can be specified for a product using Schema's *gtin-13* / GoodRelations' *hasEAN_UCC-13* properties. This level of detail would be valuable in creating a complete product description with a globally unique product identifier, but the information was not readily available for the climbing products. On the other hand, searching a manufacturer's web site for manufacturer product numbers/identifiers was much more straightforward.

It is valuable to discuss product identifiers in a bit more detail as it highlights the difficulties of maintaining and evolving an ontology. At the present time, GoodRelations documentation maps all of its specific identifier information (*hasMPN*, *hasEAN_UCC-13*, *hasGTIN-14*, …) as subproperties of *schema:productID* (Guha, 2014). But, Schema has evolved to map the individual properties directly.

Other aspects of a product are also described by Schema and GoodRelations. For example, one can describe how products are similar (*gr:isSimilarFor*), or how one product is an "accessory" or a "consumable" for another (*gr:isAccessoryOrSparePartFor* and *gr:isConsumableFor*). Again, this is mentioned for completeness because consumables[6] do not apply in the climbing gear space, and a climber typically replaces a complete product instead of repairing gear with spare parts for safety reasons.

## *4.4. Instances in a Retailer's Inventory*

The next level in the ROC ontology hierarchy contains instances of actual, physical products. For a retail inventory, the *gr:SomeItems* class (*schema:SomeProducts*) is used to represent the items in inventory and their inventory level (*schema:inventoryLevel* / *gr:hasInventoryLevel*). *gr:SomeItems* is defined as an "anonymous set", while *schema:SomeProducts* is defined more explicitly as a "placeholder for multiple similar products of the same kind" ("SomeProducts", n.d.). This concept is different from the instances in the Product Model Ontology (which define a "prototype" of a product) because the inventory items physically exist. Also, it is different from a particular instance of an owned product, that exists once.

The range of the *gr:hasInventoryLevel* property is a *schema:QuantitativeValue* / *gr:QuantitativeValueFloat* class. This seems confusing when first encountered, as it may be more intuitive to define a simple integer value as the range. *gr:QuantitativeValue* is used to allow flexibility when specifying the inventory level. It brings together several pieces of information necessary to understand a "value". It has properties such as a specific value and min/max values (to permit the definition of an interval), and also units of measurement. Units are specified using *gr:hasUnitOfMeasurement*, which references the UN/CEFACT Common Code as its range ("Popular UN/CEFACT Common Codes for Units of Measurement", n.d.). Specific values can be defined using either the *gr:hasValue* or *gr:hasValueFloat* properties (defining the value as a literal or as an exact or approximate numeric value, respectively). In ROC, we simply use the *gr:hasValue* property (and translate back to an integer when processing the query result). For inventory quantities, we do not specify units, although the value "C62" could be used which indicates "no unit".

Each of the items in the Retailer Inventory Ontology has three type[7] declarations: *gr:SomeItems*, *owl:NamedIndividual*, and finally the product class from the ROC climbing ontology. *Figure 7* shows the relationship of these classes, where each circle represents an *owl:NamedIndividual*. In addition to its types, each instance references its product model using the *gr:hasMakeAndModel* property. By referencing a product

---

[6] A *Consumable* superclass is defined in ROC, but its semantics do not correspond to the *isConsumableFor* property. The former defines products that the user "consumes" while climbing (such as chalk to aid a climber when gripping a hold), while the latter describes other products that are consumed when using the referencing product (such as garbage bags that are used in a particular garbage container).

[7] As stated earlier, "type" is used to describe an rdf:type declaration, which states that the instance is a member of the specified class.

model, we can avoid duplication of properties such as manufacturer product number, description or height/weight/etc. For validation, we define a constraint that any instance of *gr:SomeItems* must have the same ROC climbing ontology class as defined for its referenced product model. For example, it is not reasonable to define a product that is an instance of *roc:Rope*, and then link it to a product model that is a type of *roc:Harness*. Constraints are discussed further in Section 5.1.

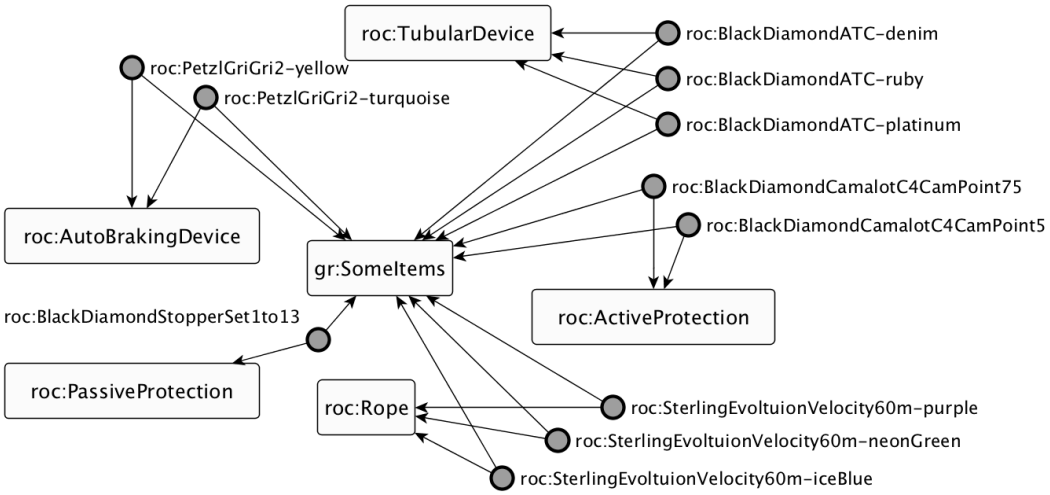


**Figure 7. Examples of Instances in a Climbing Retailer's Inventory**

It may happen that the granularity of the Product Model Ontology is different than that of the Retailer Inventory Ontology. For example, an instance of a product model may include multiple color options (grouped under a single manufacturer product number or UPC code). Unfortunately, how a manufacturer defines individual products is their prerogative. To accommodate this variability, ROC adds a specific *roc:color* property to distinguish inventory items. There is little more frustrating for a customer than to try to purchase a red climbing harness, only to be told to wait for an associate to check the actual stock, as the inventory system reports that there is one harness in either red, blue or black. The *roc:color* property would be repeated in each inventory instance in order to define the exact color(s) in stock.

Continuing the color discussion, a second constraint is defined that restricts the color of an inventory item to one of the options from its product model. Where a single value of a property is specified for a product model, such as weight, the property does not need to be repeated in the inventory instances. In all cases, the necessary information can be retrieved by writing SPARQL queries ("SPARQL 1.1 Query Language", 2013) using the OPTIONAL keyword, where the property is optional in both the product model and inventory instances.

So, once a customer is set to buy an item, or at least interested in buying something, the next thing to determine is the price. GoodRelations and Schema address this via instances of an *gr:Offering* / *schema:Offer* class, respectively. All instances of

*gr:Offering* are defined in their own namespace in ROC, and are logically peers of the inventory instances.

It is recommended to use a separate namespace/ontology for offers/pricing, as these change more frequently than the products that are stocked in inventory. The reason for separating the namespaces is to more easily track changes to the instances and to use the software design principles of loose coupling and strong cohesion. (In order to avoid revealing any pricing/offer information specific to a retailer, the ROC inventory and offer details in GitHub are fictional. They were created for illustration purposes only.)

Returning to pricing, an instance of *gr:Offering* uses the property *gr:includes*, with a range of *gr:SomeItems*, to specify which product(s) are offered. Different products could be bundled together by using multiple *gr:includes* properties, each referencing another product.

By default, it is assumed that there is one instance of a product included in an offering. If this is not a valid assumption, *gr:Offering* can define the exact quantities using the *gr:includesObject* property, which references an instance of the class, *gr:TypeAndQuantityNode*. *gr:TypeAndQuantityNode* exemplifies the use of reification in ontologies to support n-ary relationships. It has the properties *gr:typeOfGood* (which references a product or set of products in an inventory) and *gr:amountOfThisGood* (which specifies the quantity).

The price of a product is defined by relating one or more *gr:PriceSpecification* instances using the property, *gr:hasPriceSpecification*. The general superclass, *gr:PriceSpecification*, provides details on the currency type and price as a specific value or range, as well as details such as the transaction or volume amounts (for example, the minimum transaction amount to qualify for free shipping). The semantics of "transaction amount" references minimum amounts, while volumes can be specific values or ranges (such as indicating that shipping costs are $5.00 USD for purchases in the range of $50-$100 USD).

*gr:PriceSpecification* is defined as the superclass for three explicit subclasses. These are: *gr:DeliveryChargeSpecification*, *gr:PaymentChargeSpecification* (for example, if there is a surcharge for credit card transactions), and *gr:UnitPriceSpecification*. Combining a per-unit offer with delivery charge information is accomplished using the *schema:addOn* / *gr:addOn* properties. (Schema adds a fourth subclass of *gr:PriceSpecification*, *schema:CompoundPriceSpecification*, but that is not used in ROC.)

Interestingly, Schema allows the full expressivity (and complexity) of GoodRelations but also allows this information to be specified as properties of the *schema:Offer* class. This simplifies defining a basic offer. Table 1 compares a simple *schema:Offer* with the full semantics available in Schema and GoodRelations.

| | Simplified Schema | Schema | GoodRelations |
|---|---|---|---|
| **Product** | itemOffered references (an instance of) SomeProducts | includesObject references (an instance of) TypeAndQuantityNode which has a property, typeOfGood which references (an instance of) SomeProducts | Includes references (an instance of) SomeItems OR includesObject references (an instance of) TypeAndQuantityNode, which has a property, typeOfGood which references an instance of SomeItems |
| **Add-on offer** | addOn references instances of other Offers | addOn references instances of other Offers | addOn references instances of other Offers |
| **Accepted payment methods** | acceptedPayment Method references one of the values of PaymentMethod (enum) | acceptedPaymentMethod references one of the values of PaymentMethod (enum) | acceptedPayment Methods references one of the values of PaymentMethod (enum) |
| **Price** | price (numeric) and priceCurrency (string of ISO 4217 codes) | priceSpecification references an instance of PriceSpecification which has properties, price or min/max price, and priceCurrency | hasPriceSpecification references an instance of PriceSpecification, which has properties, hasCurrencyValue or hasMin/MaxCurrencyVal ue (float), and hasCurrency (ISO 4217 code) |
| **Tax or VAT included in price** | valueAddedTax Included (boolean) | valueAddedTaxIncluded (boolean) | valueAddedTaxIncluded (boolean) |
| **Time period when the offer is valid** | priceValidUntil (date) | priceSpecification references an instance of PriceSpecification which has properties, validFrom and validThrough | hasPriceSpecification references an instance of PriceSpecification, which has properties, validFrom and validThrough |
| **Min transaction amount** | eligibleTransaction Volume references an instance of PriceSpecification | eligibleTransaction Volume references an instance of PriceSpecification | eligibleTransaction Volume references an instance of PriceSpecification |

| | Simplified Schema | Schema | GoodRelations |
|---|---|---|---|
| **Volume amount** | eligibleQuantity references an instance of QuantitativeValue which has properties, value or min/maxValue and unitCode (UN/CEFACT Common Codes) | eligibleQuantity references an instance of QuantitativeValue which has properties, value or min/maxValue and unitCode (UN/CEFACT Common Codes) | hasEligibleQuantity references an instance of QuantitativeValueInteger which has properties, hasValue or hasMin/MaxValue and hasUnitOfMeasurement |

**Table 1. Comparison of Product Offer Properties between Schema and GoodRelations**

Lastly, a retailer with brick-and-mortar locations would want to document the store locations, especially when searching for a store that has a particular item in stock. This information is defined using the *schema:hasPOS* / *gr:hasPOS* [8]property. The range of the property is a *schema:Place* / *gr:Location* class with this additional detail:

- Address (*schema:address*, specifically a *schema:PostalAddress* and its properties)
- Direct contact information (*schema:telephone, schema:faxNumber*)
- Open/closed hours and days of week (*schema:openingHoursSpecification* / *gr:hasOpeningHoursSpecification* with its properties)

## *4.5. Personal Inventory of Purchased Items*

A collection of gear purchased by a person or organization is defined similarly to a Retailer Inventory, in that it contains instances of actual, physical products. The collection is maintained by the retailer, and reflects the items purchased either online or from a brick-and-mortar store. These items are instantiated as instances of *gr:Individual* / *schema:Produc*t class.

*gr:Individual* instances in the collection are defined only once and will not include properties such as *gr:hasInventoryLevel* or be part of *gr:Offerings*. Multiples of given models are differentiated by incremental numbers in their URIs, as seen in *Figure 8*, although any approach that assigns unique URIs is valid.

The collection shown in *Figure 8* contains four Black Diamond camming units (used as placed protection in rock, which will catch falls), two each of different cam sizes. In the ROC ontologies, we adopted the convention of distinguishing instances by including the size in the name ("Cam1" for a size 1 cam versus "CamPoint75" for a cam with a .75 size) and then suffixes that are incremented as items are added. Any naming convention

[8] POS is an acronym for "points of sale".

can be used (including the use of arbitrary identifiers) as long as the resulting identifiers are unique.

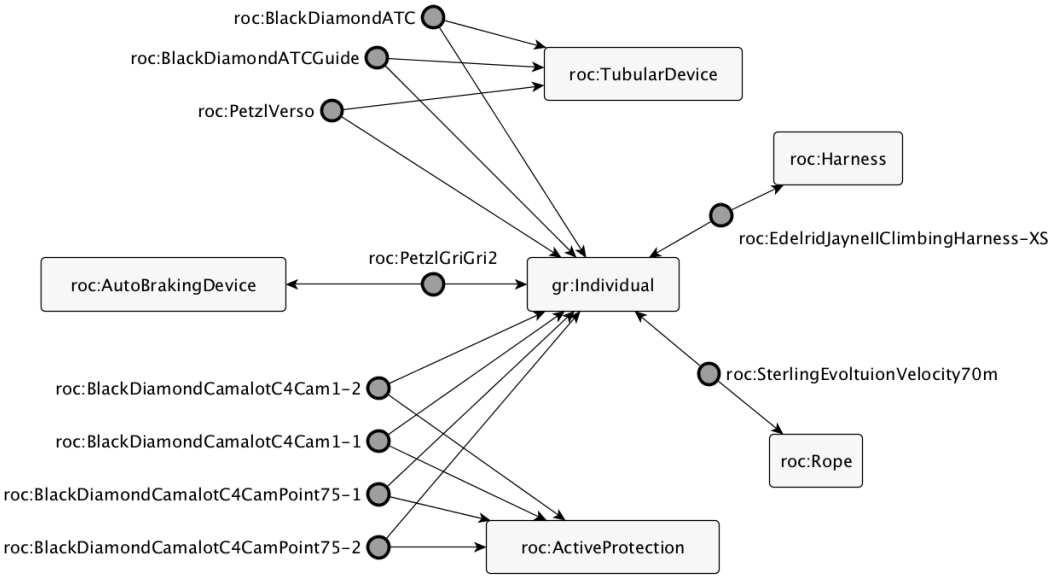


**Figure 8. Examples of Instances in a Personal Inventory**

For each individual, the date and time of purchase is captured using the *roc:purchased* property. One use of this property is to notify the owner when the expected lifetime of the product is reached.

Beyond the date of purchase, the condition of the gear could be captured using the *roc:hasCondition* property. This is important if the retailer resells used equipment. However, the retailer could also allow a user to maintain the condition of the equipment in their personal inventory. A motivation for labeling products in poor or damaged condition would be to receive notification when there are sales on replacement gear.

Another use for maintaining lists of purchased items is to notify customers in case of safety issues or recalls.

## *4.6. Purchase Lists*

In ROC, purchase lists are assumed to be generated in one of two ways. First, a customer can create a list of items to purchase while shopping online or in-person at a brick-and-mortar store. Second, a list can be automatically generated from a personal inventory based on both age and gear condition, or based on upgrading items already in the collection. The pre-generated lists could be supplemented and/or edited as a customer decides what gear to purchase. An application using the ROC ontologies can also make suggestions for other gear to buy based on items that are currently in the purchase list, based on the retailer's sales history of gear that is commonly used together.

Regarding generating a purchase list based on a personal inventory, it would include any items where the product's *roc:condition* is either *roc:Poor* or *roc:Damaged*. These are items that should be replaced for safety reasons. An application using the ROC ontologies could provide additional value by comparing the needed items with a retailer's inventory, or search online for details on sales, and then return a purchase list. Given sales history, the application might even recommend discounts for the customer. For a retailer, another application could execute online searches to compare prices, or if the retailer is out of stock, find other sellers with the gear or find alternate products from other manufacturers.

Unfortunately, when an inexperienced customer is creating a purchase list, they may select incompatible gear. The ROC ontologies can be used to issue a warning in these cases. The retailer can then ensure that the customer understands that some of the selected items should not be used together. This demonstrates the power of inferring incompatibilities, which improves customer satisfaction and reduces the need for staff to be proficient in all aspects of the domain.

# 5. Constraint and Inference Processing Using ROC

Inference is enabled when concepts and individuals are defined in ontologies. In this section, we discuss using the Stardog graph database ("Stardog", n.d.) and its integrity constraint and reasoning functionality to address some of the application areas mentioned above[9].

## *5.1. Constraints Applicable to a Retailer's Inventory*

Two types of constraints were discussed in Section 4.4 related to a retailer's inventory. The first was to validate that the ROC climbing ontology class was the same for an individual product in the inventory as was defined for its backing product model. The second was that the individual product's color was one of the colors listed in its related product model. The SPARQL declarations for both of these constraints are shown in *Figure 9* and *Figure 10*.

[9] Inferencing rules in Stardog are defined using SPARQL and a Semantic Web Rule Language (SWRL) ("SWRL: A Semantic Web Rule Language Combining OWL and RuleML", n.d.). Constraints are a validation of instances modeled as OWL axioms (expressed in SPARQL), based on a Closed World Assumption and a weak variant of the Unique Name Assumption (Perez-Urbina, Sirin & Clark, n.d.).

```
PREFIX gr: <http://purl.org/goodrelations/v1#>
SELECT ?individual
       ?indClimbingGear
       ?prodClimbingGear
       ('The individual identified as ?individual is a different type of gear than its
referenced product. The individual is identified as a ?indClimbingGear, but the product
is a ?prodClimbingGear.' as ?violation)

WHERE { ?individual a gr:SomeItems ;
                    a ?indClimbingGear ;
                    gr:hasMakeAndModel ?product .
        ?product a ?prodClimbingGear .

        FILTER regex(str(?indClimbingGear), 'roc#') .
        FILTER regex(str(?prodClimbingGear), 'roc#') .
        FILTER (?indClimbingGear != ?prodClimbingGear) . }
```

**Figure 9. ROC ClimbingGear Type Constraint**

*Figure 9* demonstrates how constraints can be defined in SPARQL and include complete information on the error/warning[10]. The error/warning message is the value of the ?violation variable, where we also use the query variable names as substitution strings. The English translation of the SPARQL constraint is:

- Find an instance that is defined as a *gr:SomeItems* and also get its ROC climbing ontology class (the ?indClimbingGear variable, filtered by the namespace "roc")
- Follow the *gr:hasMakeAndModel* property to the backing product model and retrieve its ROC climbing ontology class
- Only return instances where the ROC types of the individual and the backing product do not match

*Figure 10* is a similar constraint which is interpreted as:

- Find an instance that is defined as a *gr:SomeItems* and also get its color (the ?color variable)
- Follow the *gr:hasMakeAndModel* property to the backing product model
- Only return instances where the product's colors do not include the individual's color

[10] At this time, we are only statically reporting errors in English, although it would be possible to instead reference an instance of an ErrorMessage class (also defined in the ontology) with various text properties that differ by language.

```
PREFIX gr: <http://purl.org/goodrelations/v1#>
SELECT ?individual
       ?color
       ?product
       ('The individual identified as ?individual has color (?color) that is not valid
for the referenced product, ?product.' as ?violation)

WHERE { ?individual a gr:SomeItems ;
                    gr:color ?color ;
                    gr:hasMakeAndModel ?product .

        FILTER NOT EXISTS { ?product gr:color ?color } . }
```

**Figure 10. Product's Color Constraint (Restricted to Product Model's Colors)**

In order to determine if there are constraint violations in Stardog, an application simply loads the SPARL declarations into a database and then requests that the database be validated.

Other constraints have also been defined:

- Beyond color, the other properties of an item in a retail or personal inventory (such as size, usage, etc.) must be one of the values of the related product model
- Items in a personal inventory should have a condition
  - A constraint for all *gr:Individual*s
- Retail inventory items should have an inventory level
  - A constraint for all *gr:SomeItems*
- All individuals in a retail or personal inventory should have only one color option, one size, one height, one width, …
  - A constraint for all *gr:Individual*s and *gr:SomeItems*

By individually adding or removing constraints, a retailer or user can utilize the concepts and individuals in ROC to meet their specific requirements.

## *5.2. Constraints Applicable to a Purchase List*

One constraint was discussed in Section 4.6 related to a purchase list. It dealt with alerting a customer that they were buying incompatible gear. One example is where a static rope is purchased at the same time as a belay device. This could be a problem, as static ropes are designed for raising or lowering loads, and for use with ascenders. Dynamic ropes are designed to stretch and absorb the impact of a fall. A constraint that alerts someone to the incompatibility is shown in *Figure 11*.

```
PREFIX roc: <http://purl.org/ninepts/roc#>
SELECT ?individual
       ('A static rope is being purchased with a belay device. Please verify that
this is desired since static ropes should be used for raising/lowering loads or
with an ascender and not to protect in case of a fall.' as ?violation)

WHERE { ?individual a roc:Rope ;
                    roc:dynamic false .
        ?individual2 a roc:BelayDevice. }
```

**Figure 11. Product Incompatibility Constraint Example**

This particular constraint is run with reasoning (inference) turned on in Stardog so that we do not have to declare that an instance of *roc:AutoBrakingDevice* or *roc:TubularDevice* is also a type of *roc:BelayDevice*. Because the ontology defines the former as subclasses of *roc:BelayDevice*, running the query with reasoning will take the hierarchy into account and correctly process all the instances. Without reasoning, we would have to replace the check for ?individual2 being a type of *roc:BelayDevice* with a union of checks for *roc:AutoBrakingDevice* and *roc:TubularDevice*.

## *5.3. Inferences Applicable to a Personal Inventory or Purchase List*

ROC was designed to support various, specific inferences. One example is an automated discount system. When there is overstock of an item, a retailer may want to expedite sales and therefore discount the product. To support this, a rule similar to that shown in *Figure 12* could be written. It would suggest updated prices to a retailer if the inventory level is over a certain, predefined amount. If the suggested price is approved, an application would automatically change the *gr:PriceSpecification* of a *gr:Offering* for the product. The price in the inference in *Figure 12* is a percentage discount that can be set by the retailer and can vary between products. Once the product falls below the overstock level, an application could again reset the price to its original. Another use of this sort of automated system could be for reorder levels.

```
PREFIX rule: <tag:stardog:api:rule:>

[] a rule:SPARQLRule ;
   rule:content """
        PREFIX gr: <http://purl.org/goodrelations/v1#>
        PREFIX roc: <http://purl.org/ninepts/roc#>
        IF { ?individual gr:hasInventoryLevel ?inventory .
             ?inventory gr:hasValue ?value .
             FILTER (xsd:integer(?value) > 10) .

             ?offering gr:includes ?individual ;
                       gr:hasPriceSpecification ?ps .
             ?ps a gr:UnitPriceSpecification ;
                 gr:hasCurrencyValue ?price ;
                 gr:hasCurrency ?currency . }

        THEN { BIND (?price * 0.75 AS ?newPrice) .
               BIND (CONCAT(str(?newPrice), ?currency) AS ?suggestedPrice) .
               ?ps roc:hasSuggestedCurrencyValue ?suggestedPrice . }
    """.
```

**Figure 12. Rule to Infer Discounted Price Due to Overstock**

The rule can be explained as:

- Find any individual with an inventory level greater than 10 items
    - First, the instance of *gr:QuantitativeValueFloat* that is related as an inventory level is retrieved (following the gr:hasInventoryLevel property)
- Get the *gr:QuantitativeValueFloat*'s specific value (following the *gr:hasValue* property)
    - The value must be converted to an integer, as *gr:hasValue* has a range of literal
- Find any *gr:Offering*s that reference the individual and get their *gr:PriceSpecification*
- Only retrieve *gr:PriceSpecification*s that are kinds of *gr:UnitPriceSpecification*
- For these specifications, get the price and currency
    - There may be more than one *gr:UnitPriceSpecification* if the value is specified in multiple currencies
- Define the new price and return it when querying for any *roc:hasSuggestedCurrencyValue* recommendations

## 6. Future Work

Coupling schema.org and GoodRelations with a domain-specific ontology enables a broad set of applications (some examples of these applications were discussed above). ROC is our initial attempt to demonstrate this. At present, we are working with a local climbing chain, related to modeling their gym locations, available inventory and product and training offers. At the same time, we are mining some of the larger retailers' web sites for similar product and price comparison data.

Another extension of ROC's concepts supports advice and product recommendations for a retailer's customers. This advice would codify details such as is found on REI's web pages, "Expert Advice" ("Climbing Articles & Tips", n.d.).

A third area of work improves ROC's suggestions for replacement, upgraded and "consumed" gear in an individual collection. Several scenarios that combined both customer and retailer benefits were mentioned in Section 4.6.

## References

1. Climbing Articles & Tips. (n.d.). Learn at REI. Retrieved from http://www.rei.com/learn/expert-advice/climbing.html
2. Denale, R., & Weidenhamer, D. (2016, November 17). U.S. Census Bureau News, Quarterly Retail E-Commerce Sales, 3rd Quarter 2016. Retrieved from http://www.census.gov/retail/mrts/www/data/pdf/ec_current.pdf.
3. FOAF (2000-2015+). (n.d.). The FOAF Project. Retrieved from http://www.foaf-project.org/

4. Gruninger, M., & Obrst, L., et al. (2014). Ontology Summit 2014 Communiqué Semantic Web and Big Data Meets Applied Ontology. Retrieved from http://ontolog.cim3.net/file/work/OntologySummit2014/OntologySummit2014_Communique/OntologySummit2014_Communique_v1-0-0_20140429-1045.pdf
5. Guha, R. V. (2012, November 08). Good Relations and Schema.org. Retrieved from http://blog.schema.org/2012/11/good-relations-and-schemaorg.html
6. Guha, R. V., Brickley, D., & Macbeth, S. (2015, December 15). Schema.org: Evolution of Structured Data on the Web. Databases, 13(9).
7. Hepp, M. (2011, October 01). GoodRelations Language Reference. Retrieved from http://www.heppnetz.de/ontologies/goodrelations/v1.html
8. ItemCondition. (n.d.). schema.org. Retrieved October 12, 2016, from http://schema.org/itemCondition
9. OWL 2. (2012, December 11). OWL – Semantic Web Standards. Retrieved from http://www.w3.org/OWL/
10. Perez-Urbina, H., Sirin, E., & Clark, K. (n.d.). Validating RDF with OWL Integrity Constraints. Retrieved from http://docs.stardog.com/icv/icv-specification.html
11. Popular UN/CEFACT Common Codes for Units of Measurement. (n.d.). GoodRelations Wiki. Retrieved December 10, 2016, from http://wiki.goodrelations-vocabulary.org/Documentation/UN/CEFACT_Common_Codes
12. Schema.org analysis 2014. (2014, April 22). Searchmetrics. Retrieved from http://www.searchmetrics.com/news-and-events/schema-org-in-google-search-results/
13. Schemas. (n.d.). schema.org. Retrieved October 12, 2016, from http://schema.org/docs/schemas.html
14. SomeProducts. (n.d.). schema.org. Retrieved October 3, 2016, from http://schema.org/someProducts
15. SPARQL 1.1 Query Language. (2013, March 21). W3C Recommendation. Retrieved from http://www.w3.org/TR/sparql11-query/
16. Stardog. (n.d.). Stardog 4: The Manual. Retrieved from http://docs.stardog.com/
17. SWRL: A Semantic Web Rule Language Combining OWL and RuleML. (2004, May 21). W3C Member Submission. Retrieved from http://www.w3.org/Submission/SWRL/