Initiative for developing eProcurement Ontology

eProcurement UML conceptual model conventions

Deliverable WP 1.2

Eugeniu Costetchi

17 April 2020

Disclaimer

Project acronym	ePO
Project title	Initiative for developing eProcurement Ontology
Document reference	eProcurement UML conceptual model conventions
Author(s)	Eugeniu Costetchi
Editor(s)	Eugeniu Costetchi
Contractor	Infeurope S.A.
Framework contract	10688/35368
Actual delivery date	17 April 2020
Delivery nature	Report (R)
Dissemination licence	Creative Commons Attribution 4.0 International
Filename	wp1-3-uml-conventions
Suggested readers	project partners, future users, practitioners, software architects

Abstract

TBD

Contents

1 Introduction 4
  1.1 Key words for Requirement Statements 4
  1.2 Context 4
2 Definitions 4
  2.1 Conceptual Model 4
  2.2 Unified Modelling Language (UML) 5
  2.3 Formal ontology 6
3 Requirements 8
4 Conventional constraints 8
  4.1 What is in a name? 8
  4.2 Case sensitivity 9
  4.3 Delimitation 9
  4.4 Name uniqueness 10
  4.5 Suffix and prefix 10
  4.6 Classes 11
  4.7 Relations 11
  4.8 Relations reusability 12
  4.9 Attributes 13
  4.10 Controlled lists 14
  4.11 Descriptions 14
5 Technical constraints 15
  5.1 Charset 15
  5.2 Namespaces 15
6 Final word 15

1 Introduction

The conceptual model is represented as a UML class diagram. This diagram comprises primarily packages, classes, data types, enumerations, enumeration items, class attributes, association relation and dependency relation. There are additional elements which will be addressed contextually below.

The UML Conceptual Model of the eProcurement domain represents a valuable resource for generation of the formal eProcurement ontology. It is, in principle, possible to generate automatically the formal ontology from the UML model, provided that a set of clear transformation rules are established, and that a set of modelling conventions is respected.

This document aims at establishing the conventions for and assumptions about the UML (v.2.5) model such that it can be further automatically transformed from its XMI (v.2.5.1) serialisation into a formal ontology in RDF format [ref rdf]. This process is described elsewhere [ref] and the main concern here is defining constraints and recommendations for the UML conceptual model.

1.1 Key words for Requirement Statements

The key words “MUST”, “MUST NOT”, “REQUIRED”, “SHALL”, “SHALL NOT”, “SHOULD”, “SHOULD NOT”, “RECOMMENDED”, “MAY”, and “OPTIONAL” in this document are to be interpreted as described in RFC 2119 [3].

The key words “MUST (BUT WE KNOW YOU WON’T)”, “SHOULD CONSIDER”, “REALLY SHOULD NOT”, “OUGHT TO”, “WOULD PROBABLY”, “MAY WISH TO”, “COULD”, “POSSIBLE”, and “MIGHT” in this document are to be interpreted as described in RFC 6919 [15].

1.2 Context

2 Definitions

2.1 Conceptual Model

A *conceptual model* is a representation of a system that uses concepts to form said representation. A *concept* can be viewed as an idea or notion; a unit of thought [9]. However, what constitutes a unit of thought is subjective, and this definition is meant to be suggestive, rather than restrictive. That is why each concept needs to be well named by providing preferred and alternative labels, and a clear and precise definition supported by examples and explanatory notes.

The conceptual model comprises representations of concepts, their qualities or attributes and relationships to other concepts. Most commonly these are association and generalisation relations. In addition behaviour can be represented, ranging from the concept level up to the level of the system as a whole. Behavioural aspects, however, fall out of the scope in the current specification, which addresses mainly at the structural elements.

2.2 Unified Modelling Language (UML)

The *Unified Modelling Language (UML)* is a general-purpose, developmental, modelling language in the field of software engineering that is intended to provide a standard way to visualize the design of a system [2].

This set of specifications is based on the assumption that the conceptual models are represented with UML. Moreover, for the purposes of this convention only the structural elements of UML are considered, and in particular those comprising a class diagram. Next, the most important structural elements are introduced.

A *class* represents a discrete concept within the domain being modelled. It is a description of a set of individual objects that share the same attributes, behaviour, relationships. Graphically, a class is rendered as a rectangle [2].

An *instance* or *individual object* is a discrete (run-time) entity with identity, state and invocable behaviour and which can be distinguished from other (run-time) entities. We say that an individual object instantiates a class and represents a concrete (run-time) manifestation of that class. Conversely, a class represents the abstract concept by which to understand and describe the multitude of instantiated individual objects.

A *property* is a structural feature which represents some named part of the structure of a class and characterises it in a particular fashion. It can be an attribute of a classifier or a member end of a relation.

An attribute is a named property of a class that describes the type and a range of values that instances of the property may hold. An attribute may be conceptualised as a slot shared by all objects of that class that is filled by values through instantiation [2].

When building abstractions very few classes stand alone. Instead most of them are connected to each other in a number of different ways. In UML, there are three kinds of *relationships* that are important in this specification: *dependencies*, which represent using relationships among classes (including refinement, trace, and bind relationships); *generalizations*, which link generalized classes to their specializations; and *associations*, which represent structural relationships among objects. Each of these relationships provides a different way of combining your abstractions [2].

When a class participates in an association, it has a specific role that it plays in that relationship. A *role* is the face the class at the near end of the association presents to the class at the other end of the association. It is possible to explicitly name the role a class plays in an association [2].

An association represents a structural relationship among objects. In many modelling situations, it's important for you to state how many objects may be connected across an instance of an association. This "how many" is called the *multiplicity* of an association's role, and is written as an expression that evaluates to a range of values or an explicit value. It is possible to show a multiplicity of exactly one [1], zero or one [0..1], many [0..*], one or more [1..*], or even an exact number (for example, 3) [2]. Multiplicity applies not only to associations, but to dependency relation as well, and also to class attributes.

A *stereotype* represents an extensibility mechanism that is foreseen in UML. It allows for the possibility to create new domain specific kinds of elements that are derived from the existing standard ones. In the simplest form, they act as annotations on the UML building blocks, but can redefine entirely the visual representation of the UML element. For example some elements may be considered, optional, recommended or required in the context of information exchange. This is possible by creating the three stereotypes and applying them accordingly [2].

2.3 Formal ontology

Much has been discussed about what an ontology is and is not [7]. In computational context, an ontology encompasses a representation, formal naming and definition of the categories, properties and relations between the concepts, data and entities that substantiate one, many or all domains of discourse.

Here we adopt Studer et al. [17] definition that *"an ontology is a formal, explicit specification of a shared conceptualization"*. In this specification we adopt *Web Ontology Language (OWL 2)* [5, 11, 12] to specify the formal ontologies. OWL 2 is a knowledge representation language, with formally defined meaning, designed to formulate, exchange and reason with knowledge about a domain of interest.

OWL 2 ontologies can be used along with information written in *Resource Description Framework (RDF)* [18]. RDF is a standard model for data interchange on the Web. And OWL 2 ontologies themselves are primarily exchanged as RDF documents.

An RDF document is composed of RDF statements. The *RDF statement*, or *triple*, is a three-slotted structure of the form $< subject - predicate - object >$. The RDF statement asserts that some relationship, indicated by the predicate, holds between the resources denoted by the subject and object [18]. The subject is always a resource identified by an URI, while the object may be either a URI resource or a literal value. Next, the relevant OWL 2 concepts are introduced.

Classes provide an abstraction mechanism for grouping resources with similar characteristics. Classes can be understood as sets of individuals, called the class extension. The individuals in the class extension are called the instances of the class [5].

Individuals in OWL 2 represent actual objects from the domain. There can be named individuals, which are given an explicit name to refer to the same object; and anonymous individuals, which do not have an explicit name and are used locally.

Datatypes are entities that refer to sets of data values. Thus, datatypes are analogous to classes, the main difference being that the former contain data values such as strings and numbers, rather than individuals [11].

Literals represent data values such as particular strings or integers. They can also be understood as individuals denoting data values. Literals can be either plain (no datatype) or typed [11].

In OWL 2 *properties* are defined as that which can take the *predicate* role in an RDF statement, and are distinguished as object properties and datatype properties. *Object properties* represent relationships between pairs of individuals. *Data properties* represent relationships between an individual and a literal. In some knowledge representation systems, functional data properties are called attributes [11].

3 Requirements

4 Conventional constraints

Defining naming and structural conventions for concepts in an ontology and then strictly adhering to these conventions doesn't only makes the ontology easier to understand, but also helps avoid some common modelling mistakes.

UML is a language without formal semantics. Moreover it is quite flexible and permissive with ways in which a concept can be expressed. Also, there are many alternatives in naming concepts. Often there is no particular reason to choose one or another alternative. However, we need to define a set of naming conventions for classes, relations, attributes, controlled lists and adhere to it [10].

In theory any name can be assigned to a concept, relationship or property. In practice, there are two types of constraints on the kind of names that should be used: *technical* and *conventional*. This sections deals with conventional constraint, while the technical constraints are addressed in Section 5.

4.1 What is in a name?

The naming conventions apply to the *element names* in the conceptual model. These names are intended for further use as human-readable denominations, called *labels*; and machine-readable denominations, called *identifiers*. The identifiers serve as a basis for generating URIs [1] to ensure unambiguous reference to a formal construct; while the labels are meant to ease the comprehension by human-readers. For this reason we will consider that mostly the conventional recommendations provided here apply to them and none of the technical constraints.

In [14] a simple convention is proposed: that the identifier of a conceptual model element is the name of the element, where spaces have been removed. For example, the identifier of the "Legal Entity" class is "LegalEntity". Note that the casing is important and is addressed in Section 4.2.

It is recommended that the names and descriptions for classes and properties are expressed in British English [6]. In addition, a mechanism for providing a multilingual labelling system should be adopted.

It is recommended to avoid abbreviations in concept names. Also the words employed in the meta-model such as “class”, “property”, “attribute”, “connector” etc. should be avoided as well. Names which are nonsensical, unpronounceable, hard to read, or easily confusable with other names should not be employed [4].

4.2 Case sensitivity

We can greatly improve the readability of an ontology if we use consistent capitalization for concept names. For example, it is common to capitalize class names and use lower case for property names. And so, the names of classes, data-types and enumerations must begin with a capital letter while the names of class attributes, enumeration items, association and dependency relations, including their source and target roles, must begin with a lower case character.

All the names are case sensitive. This means that the class “Winery” and the attribute “winery” are two different names. Nonetheless such similarities are strongly discouraged and more elaborated names are highly encouraged. For example, a simple elaboration is to use suffixes or prefixes.

4.3 Delimitation

In UML, the spaces in names are allowed and using them may be the most intuitive solution for many ontology developers. It is however, important to consider other systems with which your system may interact. If those systems do not use spaces or if your presentation medium does not handle spaces well, it can be useful to use another method [10].

It is recommended that the element names should avoid using spaces and instead follow a camel-case convention. *CamelCasing* is the practice of writing phrases such that the word or abbreviation in the middle of the phrase begins with a capital case.

Exceptionally, if the conceptual model authors must maintain high readability of the UML diagrams, spaces may be tolerated and handled by the conversion script. In the conversion process, spaces are trimmed and phrases turned into camel-case form. For example “ Pre-award catalogue request ” is transformed into “Pre-AwardCatalogueRequest”.

4.4 Name uniqueness

In the formal ontology, each class, property or individual in the formal ontology must be uniquely identifiable in the formal ontology. Therefore the elements of the conceptual model, classes, attributes, connectors, instance, should have unique names.

This means that there cannot exist a class and an attribute with the same name (such as a class “winery” and a property “winery”). Neither there can exist a class and an instance or an instance and a relation with the same name.

Names that reduce to the same identifier are considered unique. For example “Legal Entity” and “LegalEntity” are different labels but they reduce to the same identifier “LegalEntity”. In such cases the names are considered equal and the UML elements replicated.

Nevertheless, name uniqueness is a recommendation but sometimes it is useful to replicate an UML element. In such cases, when the names are reused, the assumption is that the two UML elements represent manifestations of the same meaning. This is especially important for relations and is explained in Section 4.8.

4.5 Suffix and prefix

Some ontology engineering methodologies suggest using prefix and suffix conventions in the names to distinguish between classes and attributes. Two common practices are to add a “has-” or a suffix “-of” to attribute names. Thus, our attributes become “hasMaker” and ”hasWinery” if we chose the “has-” convention. The attributes become “makerOf” and “wineryOf” if we chose the “of-” convention. This approach allows anyone looking at a term to determine immediately if the term is a class or an attribute. However, the term names become slightly longer [10].

Here it is recommended that the names of class attributes employ the “has-” suffix.

Other common suffixes are the prepositions “-by” and “-to”. The organisation ontology [16] exemplifies their usage in cases such as “embodiedBy” and “conformsTo”. However, if the preposition can be avoided, then do so [13].

It is recommended to use prepositions in the ontology terms only if necessary.

Optionally common and descriptive prefixes and suffixes for related properties or classes may be used. While they are just labels and their names have no inherent semantic meaning, it is still a useful way for humans to cluster and understand the vocabulary. For example, properties about languages or tools might contain suffixes such as “Language” (e.g. “displayLanguage”) or “Tool” (e.g. “validationTool”) for all related properties [6].

4.6 Classes

When choosing class names it is conventional to use simple nouns or noun phrases. In case the class refers to actions, states, relations or qualities, which are usually expressed in natural language by verbs or adjectives then they must be nominalised. We often form nouns from other parts of speech, most commonly from a verb or an adjective. We can then use the noun phrase instead of the verb or adjective to create a more formal style. This process is called nominalisation.

A class name represents a collection of objects. For example, a class “Language” actually represents all languages. Therefore, it could be more natural for some model designers to call the class “Languages” rather than “Language”. In practice, however, the singular is used more often for class names, while the plural for sets and collections [10]. Therefore, it is required that the class names must always use the singular lexical form.

When building the class hierarchy, names of direct subclasses of a class should consistently either all include or not include the name of the superclass. For example, if we are creating two subclasses of the “Wine” class to represent red and white wines, the two subclass names should be either “Red Wine” and “White Wine” or “Red” and “White”, but not “Red Wine” and “White” [10].

Class specialisations with a single child must be avoided. This means that there should be at least two sibling subclasses specified in the model. By default the classes are not disjunctive, however, if required, the transformation script may generate disjunctive classes by default.

Circular inheritance must be avoided. This means that if there is a B that specialises a class A then A may not specialise B or any of the sub-classes of B.

4.7 Relations

When establishing relations between concepts it is conventional to use verbs of action, state, process or relation such as such as “includes”, “replaces”, “manages”.

It is required to use a verb or a verb phrase for relationship terms. It should be in *lowerCamelCase* such that $< subject - predicate - object >$ triples may actually be read as natural language clauses, e.g. “EconomicOperator offers ProcuredItem” [6].

The verb phrase must be in present tense, if needed inflected as third person singular. If an additional level of specificity is needed a qualifying nominal phrase may be appended.

Relationships are usually bi-directional and the inverse one should be provided where it makes sense. Adjust the verb phrases in the predicates as appropriate, usually, by employing the *active and passive voice* in the term formulation brings the desired result. For example “uses/isUsedBy” and “refersTo/isReferredToBy” or “offers/isOfferedBy” [6].

The name of the inverse relation should not be semantically inverted verb, such as in case of “buys/sells” , “open/closes”. The semantically inverted dichotomies must be modelled in separate connectors because they represent different relations. For example the dichotomy “buys/sells” should be modelled as two pairs: “buys/isBoughtBy” and “sells/isSoldBy”.

When the relation is of different nature, more like an attribute, then prefixing and suffixing techniques can be employed. For example, in the Organisation Ontology [16], the concepts of an “Organisation” and a “Site” are defined along with two relationships that are the inverse of each other: “Organisation hasSite Site” and “Site siteOf Organisation” [13].

It is recommended that each relationship includes a definition of its inverse.

Models should define such inverse pairs for relationships although this does not extend to attributes. For example, Dublin Core[8] includes a property of “dateAccepted”, there is no inverse property that would link a given date, which is expressed as a simple value, to all the documents accepted for publication on that date.

4.8 Relations reusability

The relation names should be chosen so that there is a balance of accuracy and precision on one hand and the relation reusability on the other hand. These two dimensions are inversely correlated: the higher the reuse the lower the accuracy and vice versa.

On one hand; if we choose more generic predicates such as “isSpecifiedIn” then this tends towards maximising relation reusability across the model. Yet the risk of overloading the relation meaning is also increasing.

On the other hand, the above risk could be mitigated by simply appending the range class to the relation name: such “isSpecifiedInContract” and such “isSpecifiedInProcedure” following the following naming pattern: $verbPhrase{+}Qualifier[RangeClassName]$. This ensures predicate uniqueness and maximum level of specificity at the cost of reusability across and beyond the model. The latter can be achieved through inference, but an additional predicate inheritance structure must be defined. Another risk is that a change or evolution of the name of the class has a direct impact on all incoming predicates, and thus raising the chances of errors. This in turn may decrease the model agility and elasticity.

Optionally, the transformation process from the conceptual model to the formal ontology, may contain a mechanism of appending the name of the range class to the predicate name in order to automatically produce a predicate with higher specificity, shall this be required.

4.9 Attributes

When creating attribute names, it is conventional to use simple nouns such as “name”, “weight”, “colour”. Attributes are a special type of relations that describe an entity in terms of its qualities. And so, to be consistent with the above convention and in order to increase the clarity, it is recommended to employ the prefix “has-” for each attribute even if it does not add much to the term’s meaning. So it is preferred to use terms such as “hasName”, “hasWeight” and “hasColour”.

It is recommended to use simple nouns for attribute names prepended with the verb “has-”.

To avoid laborious mechanical work of adding the prefix, it is possible to rely on the convention that the attribute names starting with a capital letter must be read as having the “has-” prefix. It means that the transformation script will prepend the “has-” prefix to all attributes starting with a capital letter.

By default, the attribute multiplicity is “1”. This should be read as any number which is expressed as “0..*”. In special cases, a list of custom default multiplicities is defined for the transformation script. That means that some data types or classes that are used as attribute types are paired with a default multiplicity, for example "1..1", "0..1", "2..2".

4.10 Controlled lists

The controlled list is a carefully selected list of words and phrases and is often employed in the modelling practices. The controlled list has a name and a set of terms. For example the list of grammatical genders can be named "Gender" and comprise the terms "masculine", "feminine", "neuter" and "utrum".

It is required that the controlled lists are named using nouns or nominal phrases starting with a capital letter. The enumeration items must start with a lower case.

As a rule of thumb, but not always, the relationship between the controlled list as a whole and its comprising elements can be informally conceptualised as a class-instance, class-subclass, set-item, or part-whole.

4.11 Descriptions

Large emphasis is set on the naming conventions. Nonetheless, most often, a good name is insufficient for an accurate or easy comprehension by human-readers. To mitigate this and increase the conceptual richness, practitioners may wish to provide human readable definitions, notes, examples and comments grasping the underlying assumptions, usage examples, additional explanations and other types of information.

It is recommended that each element is defined by a crisp, one-line definition. The definition starts with a capital letter and ends with a period.

A description may provide complementary information concerning the usage of the element or its relation to relevant standards. For example, a description may contain recommendations about which controlled vocabularies to use, describe the underlying assumptions and additional explanations for reducing ambiguity. Descriptions may contain multiple paragraphs separated by blank lines. The descriptions should not paraphrase the definitions.

In case the model editor provides concrete examples of possible element values or instances then they shall be provided as a comma-separated list. Each example value is enclosed in quotes and is optionally followed by a short explanation enclosed in parentheses [14].

5 Technical constraints

5.1 Charset

The ontology shall be encoded in at least RDF [18] and XML[4] and shall conform to both format specifications. Both languages effectively require that terms begin with an upper or lower case letter from the ASCII character set, or an underscore (_). This tight restriction means that, for example, terms may not begin with a number, hyphen or accented character [13]. A formal definition of these restrictions is given in the XML specification document [4].

It is required that the names use words beginning with an upper or lower case letter (A–Z, a–z) or an underscore (_) for all terms in the model. Digits (0–9) are allowed in the subsequent character positions.

5.2 Namespaces

6 Final word

Bibliography

[1] T. Berners-Lee, R. T. Fielding, and L. M. Masinter. Uniform Resource Identifier (URI): Generic Syntax. RFC 3986, Jan. 2005. URL `https://rfc-editor.org/rfc/rfc3986.txt`.

[2] G. Booch, J. Rumbaugh, and I. Jacobson. *Unified Modeling Language User Guide, The (2nd Edition) (Addison-Wesley Object Technology Series)*. Addison-Wesley Professional, 2005. ISBN 0321267974.

[3] S. O. Bradner. Key words for use in RFCs to Indicate Requirement Levels. RFC 2119, Mar. 1997. URL `https://rfc-editor.org/rfc/rfc2119.txt`.

[4] T. Bray, M. Sperberg-McQueen, E. Maler, F. Yergeau, and J. Paoli. Extensible markup language (XML) 1.0 (fifth edition). W3C recommendation, W3C, Nov. 2008. http://www.w3.org/TR/2008/REC-xml-20081126/.

[5] M. Dean and G. Schreiber. OWL web ontology language reference. W3C recommendation, W3C, Feb. 2004. http://www.w3.org/TR/2004/REC-owl-ref-20040210/.

[6] M. Dekkers, E. Stani, B. Wyns, and F. Barthelemy. D02.01 - specification of the process and methodology to develop the eprocurement ontology with initial draft of the eprocurement ontology for 3 use cases. Deliverable SC378DI07171, Publications Office of the European Union, 2017.

[7] N. Guarino, D. Oberle, and S. Staab. What is an ontology? In *Handbook on ontologies*, pages 1–17. Springer, 2009.

[8] J. Kunze and T. Baker. The dublin core metadata element set. Technical report, RFC 5013, August, 2007.

[9] A. Miles and S. Bechhofer. SKOS simple knowledge organization system reference. W3C recommendation, W3C, Aug. 2009. http://www.w3.org/TR/2009/REC-skos-reference-20090818/.

[10] N. F. Noy, D. L. McGuinness, et al. Ontology development 101: A guide to creating your first ontology, 2001.

[11] B. Parsia, P. Patel-Schneider, and B. Motik. OWL 2 web ontology language structural specification and functional-style syntax (second edition). W3C recommendation, W3C, Dec. 2012. http://www.w3.org/TR/2012/REC-owl2-syntax-20121211/.

[12] P. Patel-Schneider, B. Parsia, and B. Motik. OWL 2 web ontology language structural specification and functional-style syntax. W3C recommendation, W3C, Oct. 2009. http://www.w3.org/TR/2009/REC-owl2-syntax-20091027/.

[13] PwC EU Services. D3.1 - process and methodology for core vocabularies. Deliverable, ISA programme of the European Comission, 2011.

[14] PwC EU Services. e-government core vocabularies handbook. Report, ISA programme of the European Comission, 2015. URL `https://ec.europa.eu/isa2/library/e-government-core-vocabularies-handbook_en`.

[15] E. Rescorla, R. Barnes, and S. Kent. Further Key Words for Use in RFCs to Indicate Requirement Levels. RFC 6919, Apr. 2013. URL `https://rfc-editor.org/rfc/rfc6919.txt`.

[16] D. Reynolds. The organization ontology. W3C recommendation, W3C, Jan. 2014. http://www.w3.org/TR/2014/REC-vocab-org-20140116/.

[17] R. Studer, V. R. Benjamins, and D. Fensel. Knowledge engineering: principles and methods. *Data & knowledge engineering*, 25(1-2):161–197, 1998.

[18] D. Wood, R. Cyganiak, and M. Lanthaler. RDF 1.1 concepts and abstract syntax. W3C recommendation, W3C, Feb. 2014. http://www.w3.org/TR/2014/REC-rdf11-concepts-20140225/.