Differentiating Conceptual Modelling from Data Modelling, Knowledge Modelling and Ontology Modelling and a Notation for Ontology Modelling

Tharam Dillon, Elizabeth Chang, Maja Hadzic, Pornpit Wongthongtham

Digital Ecosystems and Business Intelligence Institute
School of Information Systems
Curtin University of Technology
GPO Box U1987, Perth 6845, Western Australia

(t.dillon, e.chang, m.hadzic, p.wongthongtham)@curtin.edu.au

Abstract

This paper considers conceptual modelling for three purposes namely data modelling, knowledge modelling and ontology modelling. It differentiates between the nature of the conceptual models for these three. It then proposes a representation suitable for ontology modelling.

Keywords: Conceptual Modelling, Data Modelling, Knowledge Modelling and Ontology Modelling

1 Introduction

Conceptual modelling has been previously used for many purposes. Dillon and Tan (Dillon & Tan 1993) discuss conceptual modelling for traditional software, data modelling and knowledge modelling. Dillon, Chang, Rahayu and Dillon (Dillon *et al.* 2008) extend these ideas to conceptual modelling for XML Documents and Web Services.

A great deal of discussion has arisen recently over the relationship between conceptual models for data modelling, knowledge modelling and ontology modelling with each group working in a given area sometimes taking an almost ideological view of their work in trying to distinguish it from the other two. It is, therefore, useful to take a step back and ascertain what the conceptual models in each of these three areas are trying to achieve and to differentiate between them. Our group is uniquely placed to this having worked solidly in all three areas.

A related issue is the question of a suitable notation for representation of the ontology models. Note here that while considerable work has been done on representations for:

1. Implementation of ontologies.
2. Formal semantics based on description logics.

Less work has been directed towards a representation that is in a form suitable for domain experts and non computer scientists to understand and critique such an ontology model. In this paper we propose such a notation.

2 Ontology Definition

Ontology can be viewed as a shared conceptualization of a domain that is commonly agreed to by all parties. It is defined as 'a specification of a conceptualization' (Gruber 1993). 'Conceptualization' refers to the understanding of the concepts and relationships between the concepts that can exist or do exist in a specific domain or a community. A representation of the shared knowledge in a specific domain that has been commonly agreed to refers to the 'specification' of a conceptualization.

From this definition, the key constructs are 'shared' and 'commonly agreed' knowledge. There is a common misunderstanding about ontologies. Some people may think ontology is only a set of words, definitions or concepts, while others think ontology is similar to an object-oriented technology or simply a knowledge base.

However, this is not true and these are not ontologies because the ideas contained in the signifiers 'shared' and 'commonly agreed' must be considered when talking about ontologies. It is important to note that:

- If some concepts or knowledge is shared but the meaning is not commonly agreed or vice versa, they are not ontologies.
- If a set of knowledge is only referred to as ontology, but nobody uses it or shares it, then it is also not ontology.
- If a set of concepts is used to describe objects, it can be shared knowledge within a single application or a single object-oriented system; however, it is not ontology, because ontology is for a specific domain and not for a specific application.

Ontology is the agreed understanding of the 'being' of knowledge. In other words, there is consensus regarding the interpretation of the concepts and the conceptual understanding of a domain. Every domain has a commonly used set of concepts or words, and a set of possible closely related meanings to select from. If these concepts are attributed the same meaning by the community, business parties or domain of service operators, then computers can easily be used to establish communication and carry out tasks automatically. If, within a domain, different parties were to use their own terms and definitions, and there were no translators, then tasks could become very time consuming. It would

become virtually impossible for computers to do the tasks automatically using the Internet, especially in the service-oriented network or virtual collaborative environment.

A very simple example of an ontological viewpoint of a concept is that a 'Published Article' should include author name(s), article title, publisher of journal or conference name, or newspaper name, place of publication, date of publication and ISBN number, or volume number or page number. This single concept can be used cross-culturally and internationally, from the public sector to the private, from groups to individuals.

3 Ontology Design Versus Data Modelling

Ontologies and data models both represent domain knowledge but are showing some fundamental differences. Useful points of reference when making the comparison are (Spyns *et al.* 2002):

1. Application dependencies.
2. Knowledge coverage.
3. Expressive power.
4. Operation levels.

Ontology and data models differ from each other in their dependencies on the application(s) they are designed for. Data models are designed to fulfil specific needs and therefore depend on the specific tasks that need to be performed. Users, goals, purposes and intended use of the model influence the modelling process and the level of detail described by this model. In contrast, ontologies are as generic and task-independent as possible. Minimum of application requirements are being considered during the ontology design. For example, an important property of a gene is the fact that it provides a code for a protein. If the application does not require this knowledge, the data model will be designed without it. On the other hand, ontologies represent agreed and shared knowledge. So, the fact that gene codes for a protein will need to be included in the gene ontology. Ontology consists of relatively generic knowledge that can be reused by different kinds of applications across a community of users.

An ontology and data models differ from each other in the amount and kind of knowledge they cover. Data models focus on the establishing correspondence between organization of the data in databases and concepts for which the data is being stored. On the contrary, ontologies are concerned with the understanding of the knowledge by community members. Ontologies are a more complete representation of concepts and relationships. Ontology knowledge is considered to be the hard coded part of a computer system (Greiner *et al.* 2001). Knowledge understood to be true within a certain domain does not change relatively to the way this knowledge is organized and stored within a computer system. The data represent a dynamic part of a computer system which may change even during run-time (Greiner et al. 2001).

Ontologies have more expressive power compared to the data models. Data modelling languages use typical language constructs. In contrast, ontology languages are more expressive languages as they include constructs that express other kinds of meaningful constraints such as taxonomy or inferencing. Ontology languages make the domain conceptualization more correct and precise.

Ontology and data models operate on different levels. Ontologies are generic, task and implementation-independent and as such operate on a higher level of abstraction. In contrast, data models are at the lower level of abstraction. Thus a 'new car database' would model a car using attributes such as model and make of car, pricing, colour, engine size, accessories. In contrast, a 'repair database' might model spare parts associated with a particular model and make of a car, their price, the price for the labour to fit in the spare part, result ranges for certain tests, etc. Note each of these data modelling have a narrow view of the concepts of a car. As the ontology is the knowledge shared by different applications across a community, we can design a Car Ontology which will represent the knowledge common and shared by both 'new car database' and 'repair database'. This knowledge will operate on a higher level of abstraction and will be designed independently from the applications running on each of these databases.

Differences between the ontologies and data models, the characteristics of application dependencies, knowledge coverage, expressive power and operation levels, can be used by data modellers and ontology designers as points of reference. These guidelines will help them stay focused on their own goal.

4 Ontology Versus Knowledge Base

The difference between an ontology and a knowledge base can be seen in their different objectives (Maedche 2003). An ontology captures and represents the conceptual structure of a domain in a form that is shared by the community of users in that domain. In contrast, a knowledge base (Dillon *et al.* 2008; Dillon & Tan 1993) seeks to model knowledge in that domain in a form suitable for a particular Knowledge-Based System (KBS) to carry out problem solving. Knowledge base models the problem solver's (expert's) approach to problem solving for the particular set of problems being addressed within this domain. An ontology has terminology and conceptualization agreed by the community to be correct and to be consistently used across the community. Whilst in the KBS, the terminology can be specific to the particular system and consistency is understood to mean logical consistency within the narrow focus of the problems being solved. Knowledge base enables an inference engine or agent to reason to solve the particular set of problems that the KBS is meant to address.

The main differences between ontology and knowledge bases are summarized in Table 1.

The purpose of an ontology is to describe facts assumed to be always true by the community of users. It aims to capture the conceptual structures of a domain. In contrast, knowledge base aims to specify the portion of the domain useful for solving a particular set of problems. It may also describe facts related to a particular state of affairs.

For an agent, a shared ontology describes a vocabulary for communicating about a domain. In contrast, a knowledge base contains the knowledge needed to solve

problems or answer queries about such a domain. Domain knowledge is described by ontologies while operational knowledge is described in the knowledge base. An ontology specifies knowledge about a certain domain while a knowledge base specifies knowledge used by the agent and/or an inference engine to perform its actions.

	Ontology	**Knowledge base**
objectives	conceptual structures of a domain	particular states of a domain
consistency	facts always true	facts true for a particular state of affairs
actions	communication	problem solving
knowledge	domain knowledge	operational knowledge & restricted domain knowledge
applicability	shared by community	specific to system

Table 1: Ontology versus Knowledge base

5 Separate, yet in Unity

No discussion is needed to decide which modelling approach is the best as they are all uniquely different from each other. There is a potential behind each modelling approach to be the best choice for a specific need. For example, we know that ontologies have far more expressive power than glossaries. But extensive and complex knowledge described in an ontology may not be needed for a specific task and the designers will choose to use glossaries over ontologies.

But we need to be careful here to name the modelling approach for what it is. For example, we should not name glossaries as ontologies. By misusing the names we can only dilute and hide the real importance of a specific modelling approach. Each modelling approach needs to stand independently from the others for it to show its true and full significance. We need to embrace the true meaning of each modelling approach, know its purpose and reason of its existence.

When we clearly understand true meaning and potential of each modelling approach, we can then freely move towards using these models in unity.

Because ontologies and data model operate on different levels but are oriented in the same direction, it is possible to:

1. use ontologies to support data model design.
2. use data models to elicit knowledge that could be useful for ontology design.
3. integrate ontologies with data models.
4. use ontologies as means of working with and querying databases with different data models in the same domain (see Figure 1).

Ontologies can be used to build, support and clarify data models. An ontology can play an important role during the data model design, especially in the requirements analysis. A wide range of ontologies is available via ontological libraries and can be used to assist in the modelling of a specific application domain.

Also, data models may support the ontology design process. Data models may help to provide knowledge that will help designers make decisions regarding organization and structuring of the concepts within an ontology. The ontologies designed through support of data models are usually specialized ontologies and not general ontologies. The number of applications for which these specialized ontologies can be used is limited.

The integration of ontologies with data models into a unified meta-model can result in an effective combination of data, documents and formal knowledge (Kuhn *et al.* 1997). The main goal of this integration is to enrich the data models and increase expressiveness of the domain under analysis. The application designer needs to identify ontologies from the ontological libraries suitable to be used for that portion of the domain which needs to be formally described.

Recently, the use of ontologies over knowledge bases for the agent's operational knowledge has been proposed. This would result in two different kinds of ontologies: domain ontology and operational ontology (which replaces the knowledge base). Let's take an example to clarify the difference between domain and operational knowledge. We may be assigned a task to deliver some books to a certain address. We may have the domain knowledge about the street locations in the city. Still, we may not be able to complete this task if we lack the operational knowledge (i.e. if we do not know how to perform the action of locating the address and delivering the books). Domain and operational knowledge are both needed by an agent to function optimally.

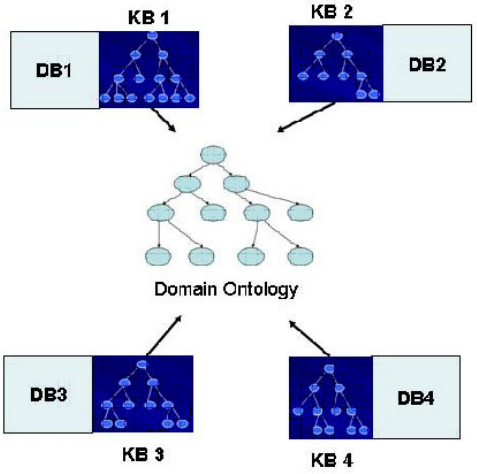


Figure 1: Different databases share same ontology

In Figure 1, we show an example where each database (DB) of the four different databases has specific applications associated with it and uses its own knowledge base (KB) to perform the specific actions. The knowledge base of one database is independent from the knowledge base of another database. A new application is proposed where the four different databases need to collaborate with each other. A domain ontology needs to be designed for this purpose and each of the four databases needs to commit to this common ontology. This ontology would enable effective and efficient cooperation

and collaboration of the four databases. The databases would be able to communicate with each other and share the information, tasks and results efficiently.

6 Ontology Representation

A domain of interest can be represented by generic ontologies and specific ontologies, which are also known as upper ontologies and sub-ontologies (lower ontologies). The specific ontologies or sub-ontologies are also known as 'ontology commitments' (Spyns *et al.* 2002). They commit to use all the upper ontology concepts and specifications. As an example, a generic concept may be called human relationship and a specific 'human relationship' concept can be 'boy and girl relationship' or 'business and customer relationship' or 'teacher and student relationship', and so on. These are specific relationship concepts and they are all committed to the specification of the generic concept 'human relationship'. The specification of the generic concept 'human relationship' can be defined as having at least two parties involved. If one party is missing, the relationship would not exist. Therefore, all the specific ontologies or sub-ontologies should commit to this specification.

To represent ontology, processes should include a set of tasks as following:

1. Define concepts and concept hierarchy in generic ontology e.g. define human relationship concept.
2. Define ad-hoc relationships e.g. define relationship of human entities.
3. Define constraints of each relationship e.g. define as having at least two entities involved.
4. Define concepts and concept hierarchy in specific ontology e.g. define business relationship concept specific to human relationship concept.
5. Define instances of ontological concepts in the specific ontology e.g. Alice can be defined as one of the human entities having a relationship with another human entity i.e. Bob. Alice and Bob are both instances of human entities.

The above tasks are not linear even though some order will be followed to ensure consistency and completeness of representation.

7 Ontology Modelling

Representing domain knowledge can be very complicated. The life cycle spans everything from modelling to evolution built up of semantic entities is a challenge. Modelling is an important aspect. It is insufficient and ineffective to use traditional software engineering modelling languages because it is not precise enough to capture knowledge.

There are various formalisms for modelling ontologies notably Knowledge Interchange Format (KIF) and knowledge representation languages descended from KL-ONE. However, those representations have had little success outside Artificial Intelligent (AI) research laboratories (Farquhar *et al.* 1996; MacGregor 1991) and require a steep learning curve. KIF provides a Lisp-like syntax to express sentences of first order predicate logic, and descendants of KL-ONE include description logics or terminological logics that provide a formal characterisation of the representation (Genesereth & Fikes 1992). Traditionally, AI knowledge representation has a linear syntax but no standard graphical representation.

In order to understand the modelling concept, it is necessary to understand it's fundamental. Modelling mechanism and notations will then be presented. We have borrowed some UML figures to define the abstract syntax, semantics, and notations as an alternative formalism. The notations further structure into metamodels / diagrams which are logically structured. The main aim is not only to make it simple and to make it easier to understand, but importantly this model should be able to capture the semantic richness of the defined ontology modelling.

The ontology modelling is a semi-formal graphical representation for specifying and modelling ontology in terms of concepts drawn from Ontology concepts and principles. We design the modelling to encompass a broad set of ontology theories and approaches inclusively.

8 Notation for Ontology Modelling

In this section, modelling notations are presented to facilitate the ontology design process. Some concepts or terms represented by the notation have multiple presentations.

Ontologies consist of instances, properties, and classes. Ontology instances represent concrete information specified as an instance of one or several ontology classes. Links between ontology instances represent their interactions. Ontology properties represent relationhips held among ontology classes/ontology instances. Ontology classes represent the concepts defined as the specification of a group of instances that belong together because they share some properties.

Ontology properties are binary relations on ontology classes / ontology instances. For example, the ontology property linkAB links the ontology class A to the ontology class B, and links the ontology instance a (an ontology instance of ontology class A) to the ontology instance b (an ontology instance of ontology class B). Ontology properties can have inverses. Ontology properties can be limited to having a single value, which is to being functional. They can also be either transitive or symmetric. Each ontology property is used to create restrictions that are used to restrict the ontology instances that belong to an ontology class. These ontology property characteristics and restrictions are explained in more detail later in the sections. Ontology properties are also known as roles in Description Logic and are roughly equivalent to relationships in the usual conceptual model used in software engineering. Ontology properties are also quite close to attributes in object-oriented modelling. Ontology classes are organised into a superclass-subclass hierarchy. Subclasses are subsumed by their superclasses.

The ontology classes are organised in a hierarchy so each main ontology class has its subclasses. The subclasses can either be disjoint or decomposition or partition.

Disjoint ontology classes: their ontology instance cannot be an ontology instance of more than one of these disjoint ontology classes. This means that an ontology instance that has been asserted to be a member of one of the ontology classes in the group cannot be a member of any other ontology classes in that group.

Decomposition ontology classes: the ontology classes are overlapped. An ontology instance can be a member of a combination of ontology classes in the group.

Partition ontology classes: the ontology classes form the covering. An ontology instance of an ontology class must be either an ontology instance of any one of the ontology classes in the group.

We clearly understand that the semantics of "Aggregation" or "part–of" is very different from "Generalisation" relationship. Some Ontology languages such as OWL do not clearly separate Generalisation and Aggregation. However, organisation of Partition ontology classes is more or less like aggregation. When it comes to implementation, different implementation tools have different constraints to implement the entire whole conceptual model. Therefore, one needs to carry out the transformation from the Model to a particular computing language. In this book, we model the ontology with generalisation, partition (or aggregation), decomposition, disjoint, and association relationships. We implement the ontology in OWL, therefore the part-of relationship is implemented as partition. Therefore, we carry out the transformation on this from the modelling to implementation.

An ontology class can be a superclass of more than one ontology class called multiple inheritances. Ontology allows a collection of subclasses to be declared to unify a superclass; that is to say, every ontology property of the superclass is a property of the subclasses as well.

An ontology class can take the form of complex descriptions. The complex descriptions can be built up using simpler ontology class descriptions that are bound together using logical operation i.e. OR (U) and AND (∩).

A union ontology class is created by combining two or more ontology classes using the OR operator (U). For example, consider the union of ontology classes A, B and C. This describes an ontology instance of union ontology class that belongs to either the ontology class A or the ontology class B or the ontology class C, or all.

An intersection ontology class is created by intersecting two or more ontology classes using the AND operator (∩). For example, consider the union of ontology classes A, B, and C and the intersection with ontology class C. This describes an ontology instance of intersection ontology class that is equivalent to ontology class C.

There are two conditions to describe ontology class. First is a "necessary condition" meaning that, if something is an ontology instance of an ontology class, it is necessary in order to fulfil a condition(s). An ontology class that has only necessary conditions is known as a primitive ontology class. Second is a "necessary and sufficient condition" meaning that not only the conditions are necessary for being an ontology instance of the ontology class, but also any ontology instance that satisfies conditions must be an ontology instance of the ontology class. An ontology class that has at least one set of necessary and sufficient conditions is known as a defined ontology class.

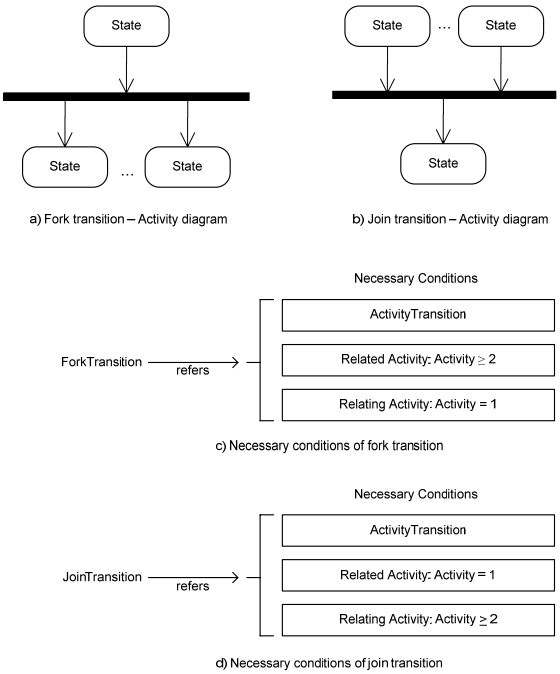


Figure 2: Fork transition in domain of activity diagram

For example, in the domain of activity transition in an activity diagram, there is the fork transition that is the transition from one activity split to at least two activities, and the join transition that is the transition from at least two activities joined to one activity shown in Figure 2 (a) and Figure 2 (b) respectively. The description of ontology class ForkTransition can state that if something is an ontology instance of the ontology class ForkTransition, it is necessary for it to be an instance of the ontology class ActivityTransition and it is necessary for it to have at least two relations of Related_Activity from the ontology class Activity and one relation of Relating_Activity from the ontology class Activity. Likewise, for the ontology class JoinTransition, it is necessary for its ontology instance to be an ontology instance of the ontology class ActivityTransition and it is necessary for it to have one relation of Related_Activity from the ontology class Activity, and at least two relations of Relating_Activity from the ontology class Activity. In this example, necessary condition is used to describe ontology class ForkTransition and JoinTransition shown in Figure 2 (c) and Figure 2 (d) respectively.

The notation of ontology class is represented as a rectangle with two compartments. The top compartment is for labelling the class and the second compartment is used for presenting properties related to the class or to an XML schema data type value. It is mandatory to specify the word '<<Concept>>' above the class label in the top compartment.

Ontology class Thing is the special class that represents the set containing all instances in the ontology. All ontology classes are subclasses of class Thing. Its notation description is the same as an ontology class notation, but the top compartment contains the word '<<Concept>>' and 'Thing' as its label and the second compartment is empty.

The generalisation symbol appears as a line with one end empty and the other with a hollow triangle arrowhead. The empty end is always connected to the class being subsumed, whereas the hollow arrowhead connects to the class that subsumes. This symbol is also used to indicate generalisation of ontology properties.

Ontology properties represent relations between two ontology classes or two ontology instances. Note that ontology properties are not necessarily tied to ontology classes. By default, an ontology property is a binary relation between thing and thing. The relations among ontology classes or ontology instances represented by data type property, annotation property, and object property come from two different sources in ontology.

Data type property associates ontology classes or ontology instances to an XML schema data type value or an RDF literal. Ontology also has same type of data type property called annotation property.

Annotation property can be used to add information called metadata that explains data about data to ontology classes, ontology instances and object or data type property.

Object property associates an ontology class to an ontology class, or an ontology instance to an ontology instance.

For example, ontology class A has a relation with ontology class B called x that is represented as an object property. Ontology class A and ontology class B have a relation with a type of xml:string called y and z respectively and both y and z are represented as a data type property. The object property x links the ontology instance a (an instance of class A) to the instance b (an instance of class B). The data type property y links the ontology instance a to the data literal c (a type of an xml:string). Another data type property z links the instance b to the data literal d (a type of an xml:string).

Ontology classes have no owned ontology properties. Ontology classes and ontology properties are independent of each other. Additionally, ontology properties may have subproperties so that it is possible to form hierarchies of ontology properties. Subclasses specialise their superclasses in the same way that subproperties specialise their super properties.

Ontology properties are binary and have distinctive beginnings and ends. Ontology properties link ontology classes from the domain to ontology classes from the range. Ontology class has its own ontology instance(s) hence specifying the domain and the range of an ontology property as an ontology class, it is eventually specifying the domain and the range of the ontology property as the ontology instances of ontology classes. For example, the object property x link ontology instance belonging to the ontology class A to ontology instance belonging to the ontology class B. In this case, the domain is ontology class A and the range is ontology class B.

It is possible to specify multiple ontology classes as the domain or the range for an ontology property. If multiple ontology classes are specified, the domain or the range of the ontology property is understood to be the union of the ontology classes. For example, if the range of an ontology property has the ontology classes A, B, and C, the range of the ontology property is then interpreted as A U B U C.

Ontology supports a fixed, defined extent for an ontology class and fixed, defined data values of data range for a data type property so called oneOf or enumeration. The former is used to define a class description of the enumeration kind. The specified list of instances is the instances of the class that enable class to be described by entirely enumerating its instances. The latter specifies the set of data values of the data range.

Ontology enriches the meaning of properties through the use of property characteristics. The first characteristic is functional properties, also known as single valued properties. For a given instance, if a property is functional, then there can be at most one instance that is related to the instance through the property. In other words, a functional property has a maximum cardinality of 1 on its range. Another property characteristic is inverse functional properties. For a given instance, if a property is inverse functional, then it means that the inverse property is functional; there can be at most one instance related to that instance through the property. In other words, an inverse functional property has a maximum cardinality of 1 on its domain. Ontology allows properties to be declared symmetric or transitive. It is important that in both cases, the domain and range must be type compatible. Ontology properties are used to create restrictions which are used to restrict the instances that belong to a class. There are three main categories of restrictions: quantifier, cardinality, and hasValue restrictions.

Quantifier restrictions are of two types: someValueFrom which can be read as "at least one" or "some" and allValueFrom which can be read as "only". Ontology property can have its range restricted when the property is applied to the domain class, either that the range is limited to a class only (allValueFrom) or that the range is one part of a class (someValueFrom). Notice that in allValueFrom restrictions, the range would not have been related with other classes apart from a specified class.

An ontology property when applied to a class can be constrained by cardinality restrictions on the domain giving the minimum (minCardinality), maximum (maxCardinality), or exact (cardinality) specified number of instances which can participate in the relation. Maximum cardinality restrictions denoted by the "less than or equal to" specify the maximum number of relationships that an instance can participate in for a given property. Minimum cardinality restrictions denoted by the "greater than or equal to" specify the minimum number of relationships that an instance must participate in for a given property. Cardinality restrictions denoted by the "equal" specify the exact number of relations that

an instance must participate in for a given property. A hasValue restriction describes the set of instances that have at least one relation along a specified property to a specific instance.

The notation of ontology property can be represented as a rectangle. In the rectangle, it is mandatory to specify the word either '<<Object property>>' or '<<Data type property>>' for the object property or the data type property respectively above the property label. Just as class has subclasses, property can have sub properties as well. Symbols of generalisation are the same for both class and property. Because they are applied to different entities, they will not create any confusion.

Data type property of ontology class can be expressed in the bottle compartment of class notation. This is an alternative design for data type properties. That means the top compartment is called its domain. In the bottle compartment, notation formats as in the order of data type property name, its characteristic, its type (e.g. String, Integer) and its restriction. The type of data type property is considered as a range. The characteristics of data type property can be functional, inverse functional, non-functional, transitive, and symmetric. An oneOf in ontology is represented in curly brackets ({}).

Object property of ontology class can be expressed in the bottle compartment of class notation like data type properties. This is an alternative design for object property. In this manner, the top compartment is still called its domain. Notation format in the bottle compartment is the same as the order of the object property's name, its characteristics, class name (its range), and its restriction. Class name expression as a range can assert the complex class description e.g. union, intersection. In addition, an object property can be expressed as an arrow with an open arrowhead and with text label of the object property name. This is another alternative design for object properties. The arrow points from the domain of property to the range of property. Its restrictions can be expressed in the bracket after its name. The characteristics of object property can be functional, inverse functional, non-functional, symmetric and transitive.

Characteristics of the software engineering ontology property are classified into five categories which are functional, inverse functional, non-functional, symmetric and transitive properties. A functional property notation is represented as an open arrowhead with text label of property name and symbol number 1 at near the arrowhead. A non-functional property notation is represented simply as an open arrowhead with the text label of the property name, which is similar to a functional property notation. The difference is that there is no symbol number 1 near the arrowhead. However, at the arrow head the cardinality restriction can be presented near the arrowhead. An inverse functional property notation is represented as an open arrowhead with the arrowhead pointing in the opposite direction to its inverse property notation and a dotted line placed in the middle links the inverse functional property symbol and its inverse. As usual, the text label represents the property name and there is symbol number 1 near the arrowhead. A symmetric property notation is represented simply as a solid line with a text label of the property name. Lastly, a transitive property notation is represented as a dotted arrow with an open arrowhead and with a text label of its property name.

In the case of property being expressed in the bottle compartment of class notation, a functional property is represented by the word 'Single' whereas a non-functional property is represented by the word 'Multiple'. An inverse functional property and a symmetric property are represented by the words 'Inverse' and 'Symmetric' respectively.

An upside down A symbol ∀ represents restriction allValueFrom. A backwards facing E symbol ∃ represents restriction someValueFrom. A symbol denoted by ∋ represents restriction hasValue. For the cardinality restriction, symbols equal (=), greater than and equal (>) and less than and equal (<) represent respectively cardinality specifying the exact number, minimum cardinality specifying the minimum number and maximum cardinality specifying the maximum number. Cardinality constraints can be placed with the format of x..y where 'x' is the value of minimum cardinality and 'y' is the value of maximum cardinality. The asteroid (*) is used as part of the specification to indicate the unlimited upper bound. For example, specifying 'a > 2' is equivalent to '2..*', specifying 'a < 2' is equivalent to '0..2', specifying 'a = 2' is equivalent to '2..2' or just '2'.

Table 2 summarises the potential indicators matching with ontology property characteristics or restriction.

Association class can be used to participate in further associations for property in ontology. To specify association classes, a dotted line is used to attach the property notation to the ontology class notation.

Ontology allows to define instances and to assert properties about them. Ontology instances do not by default satisfy the unique name assumption. Having said that, the same instance always refers to the same object but a given object may be referred to by several different instances. Accordingly, counting a set of instances does not warrant the inference that the set refers to that number of objects.

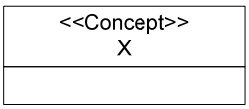
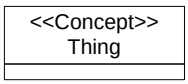
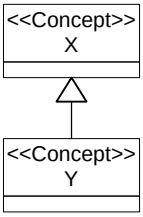
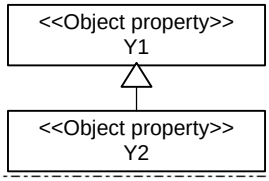
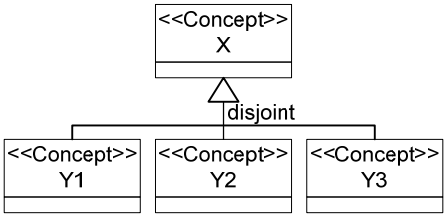
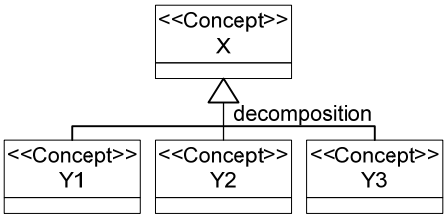
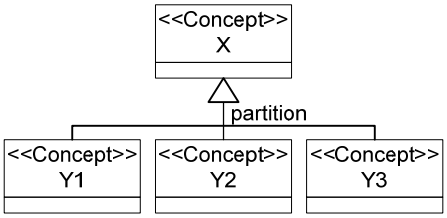
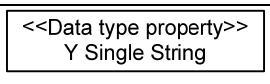
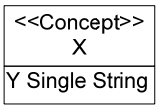
An instance notation is represented as an ellipse with a dotted line attached to its class or property. If it is an instance of property, then in the ellipse it contains the property name followed by a colon and then the instance name. Unlike an instance of class, in the ellipse there is only the instance name. To make it easy to read, a dotted line is attached to most of the class name or property name of its class instance or property instance.

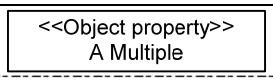
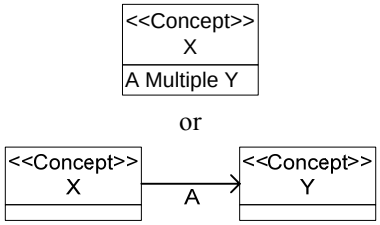
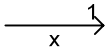
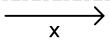
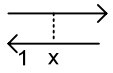
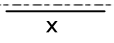
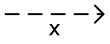
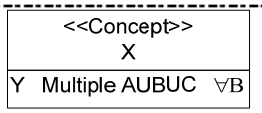
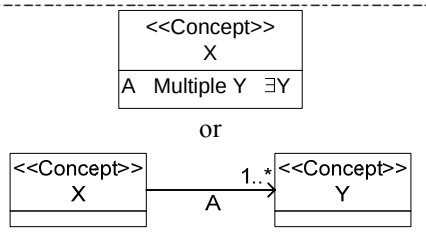
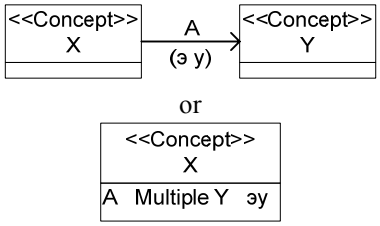
Table 3 shows a list of notations and their potential use.

Indicator	Property Characteristic/Restriction	Meaning
1	Functional property	One only
0..*	Non-functional property	Zero or more
1..*	someValueFrom restriction	At least one

n	Cardinality restriction	Only n where n > 1
0..n	Maximum cardinality restriction	At most n where n > 1
n..*	Minimum cardinality restriction	At least n where n > 1

Table 2: A list of indicators

Concept/Term	Notation	Semantics
Class		Ontology class *X*.
owl-Thing Class		All classes in ontology are subclasses of class *owl:Thing*.
Generalisation		To express either class-subclass or property-sub property.
		Class *Y* is a subclass of class *X*.
		Object property *Y2* is sub property of object property *Y1*.
		Disjoint classes *Y1*, *Y2* and *Y3*.
		Decomposed classes *Y1*, *Y2* and *Y3*.
		Partition classes *Y1*, *Y2* and *Y3*.
Data type property		Data type property *Y* which is functional and its type is string.
		Class *X* has data type property *Y* which is functional and its type is string.

Object property		Object property *A* which is non-functional.
		Class *X* has relations with class *Y*. The relation considers as an object property named *A* which is non-functional property.
Property characteristic		
- Functional property		A functional property *X*.
- Non-functional property		A non-functional property *X*.
- Inverse functional property		An inverse functional property *X*.
- Symmetric property		A symmetric property *X*.
- Transitive property		A transitive property *X*.
Property restriction		
- allValueFrom	∀	All value from
		Class *X* has relations with the union of class *A*, *B* and *C*. The relation considers as an object property named *Y* which is non-functional property. The allValueFrom of property *Y* defines that those relations relate to instances from the class *B* only.
- someValueFrom	∃	Some value from
		Class *X* has relations with class *Y*. The relation considers as an object property named *A* which is non-functional property. The someValueFrom of property *A* defines at least one relation related to an instance of class *Y*.
- hasValue	∋	Has value
		Class *X* has relations with class *Y*. The relation considers as an object property named *A* which is non-functional property. hasValue restriction is used to the property *A* implied that instances of class *X* have at least one relation to specific instance y of class *Y*.
- Cardinality	=	To specify the exact number

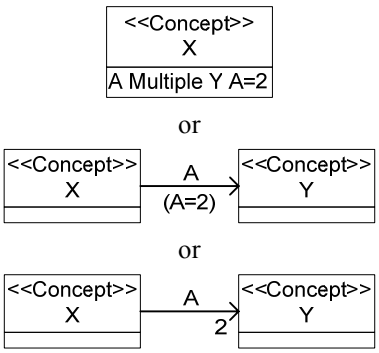
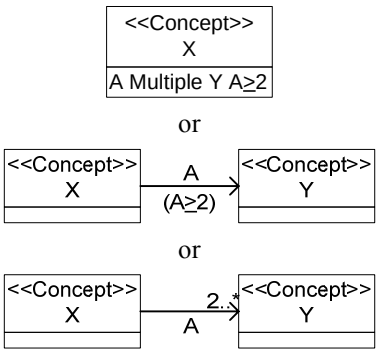
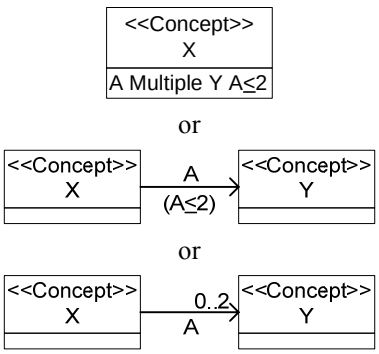
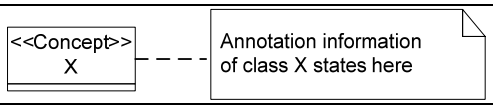
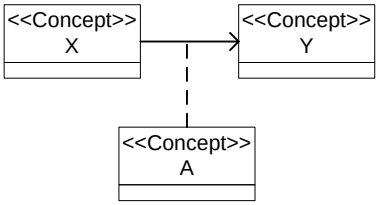
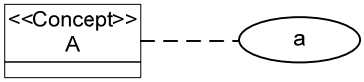
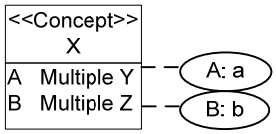
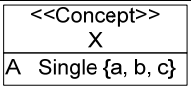
		Class *X* has relations with class *Y*. The relation considers as an object property named *A* which is non-functional property. Property *A* restricts that there are exactly two relations from instances of class *X* related to instances of class *Y*.
- Minimum cardinality	≥	To specify the minimum number
		Class *X* has relations with class *Y*. The relation considers as an object property named *A* which is non-functional property. Property *A* restricts that there are from at least two of relations from instances of class *X* related to instances of class *Y*.
- Maximum cardinality	≤	To specify the maximum number
		Class *X* has relations with class *Y*. The relation considers as an object property named *A* which is non-functional property. Property *A* restricts that there are from at most two of relations from instances of class *X* related to instances of class *Y*.
Annotation property		Class *X* has annotation property.
Association class		Class *A* is association class of object property related class *X* to class *Y*.
Class instance		*a* is instance of class *A*.
Property instance		*a* is instance of property *A* and *b* is an instance of property *B*.
oneOf		Functional data type property *A* relates to a set of data value of ‘a’, ‘b’ and ‘c’.

Table 3: A list of modelling notations

9 Illustration Examples

This section provides examples of ontology modelling for representing a part of trust ontologies. Figure 3 shows ontology representation of human relationships. Basically a human relationship is a relationship between human entities. The human relationship can be specific to the relationship between provider and customer and we call this particular relationship a business relationship. Ontologically, the business relationship is sub-ontology of a human relationship so it has to commit to the upper ontology's specification. Properties of trusted entities, trusting entities, and trust value in upper ontologies are inherited by sub-ontologies. Properties of trusted entities and trusting entities in the business relationship can be made specific to the human entities of provider and customer respectively.

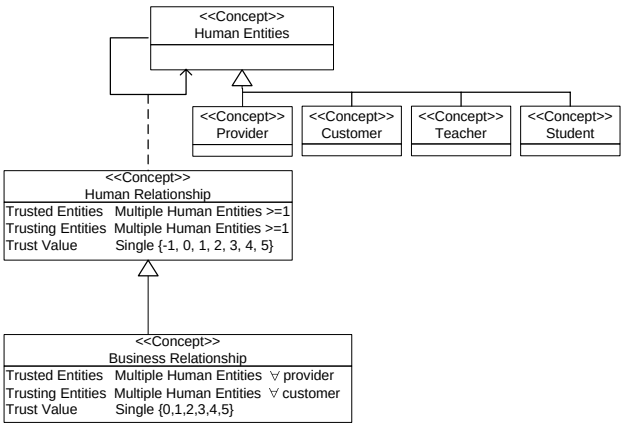


Figure 3: A part of trust ontologies

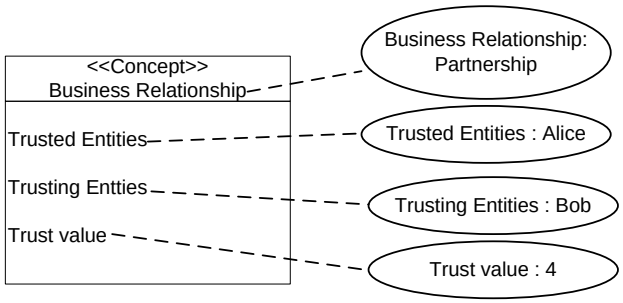


Figure 4: Ontology instances representation

Figure 4 shows ontology instances of the business relationship. Ontologically we can say that partnership between Alice and Bob is a kind of business relationship and trust value of this particular relationship is worth 4.

10 Conclusion

This paper differentiates between the conceptual models used for data modelling, knowledge modelling, and ontology modelling. It next examines the problem of developing a conceptual model for an ontology. It provides a notation that can be used for representing the conceptual models.

11 References

Dillon, TS, Chang, E & W., R 2008, *Object Oriented and Component based Conceptual Modeling and Design*.

Dillon, TS & Tan, PL 1993, *Object-Oriented Conceptual Modeling*, Prentice Hall, Australia.

Farquhar, A., Fikes, R. and Rice, J. (1996): 'The Ontolingua Server: A Tool for Collaborative Ontology Construction', *10th Knowledge Acquisition for Knowledge-Based Systems Workshop*, Banff, Canada.

Genesereth, M.R. and Fikes, R.E. (1992): *Knowledge Interchange Format Version 3 Reference Manual*, Stanford University Logic Group.

Greiner, R., Darken, C. and Santoso, N. I. (2001): Efficient reasoning. *ACM Computing Surveys* **33**(1):1–30.

Gruber, T.R. (1993): A translation approach to portable ontology specification, *Knowledge Acquisition*: pp. 199-220.

Kuhn, O. and Abecker, A. (1997): Corporate Memories for Knowledge Management in Industrial Practice: Prospects and Challenges. *Journal of Universal Computer Science* **3**(8):929-954.

MacGregor, R. (1991): Inside the LOOM classifier, *SIGART bulletin* **2**(3): 70-76.

Maedche, A.D. (2003): *Ontology Learning for the Semantic Web*. Norwell, Massachusetts, Kluwer Academic Publishers.

Spyns, P., Meersman, R. and Jarrar, M. (2002): Data modelling versus Ontology engineering. *SIGMOD Record* **31**(4):7-12.

.