Exploiting unified modelling language (UML) as a preliminary design tool for Common Logic-based ontologies in manufacturing

Exploiting UML as a Preliminary Design Tool for Common Logic-based Ontologies in Manufacturing

C. Palmer[a], A.G. Gunendran[b] , R.I.M. Young*[a], N. Chungoora[a], Z. Usman[a], K. Case[a] and J.A. Harding [a]

[a]*Wolfson School of Mechanical and Manufacturing Engineering, Loughborough University, Loughborough, UK*

[b]*Emerson Control Techniques, Newtown, Powys, UK*

Claire Palmer
Wolfson School of Mechanical and Manufacturing Engineering, Loughborough University, Loughborough, LE11 3TU, UK
Tel: +44 (0)1509 227684
Email:C.Palmer3@lboro.ac.uk

George Gunendran
Emerson Control Techniques, Newtown, Powys, UK
Email: AGeorgeG@gmail.com

Robert Young
Wolfson School of Mechanical and Manufacturing Engineering, Loughborough University, Loughborough, LE11 3TU, UK
Tel: +44 (0)1509 227662
Fax: +44(0)1509227648
Email: R.I.Young@lboro.ac.uk

Nitishal Chungoora
Wolfson School of Mechanical and Manufacturing Engineering, Loughborough University, Loughborough, LE11 3TU, UK
Tel: +44(0)7871226566
Email: N.Chungoora@lboro.ac.uk

Zahid Usman
Wolfson School of Mechanical and Manufacturing Engineering, Loughborough University,

Loughborough, LE11 3TU, UK
Tel: +44 (0)1509 227634
Email: Z.Usman@lboro.ac.uk

Keith Case
Wolfson School of Mechanical and Manufacturing Engineering, Loughborough University, Loughborough, LE11 3TU, UK
Tel: +44(0)1509227654
Fax:+44(0)1509227648
Email: K.Case@lboro.ac.uk

Jenny Harding
Wolfson School of Mechanical and Manufacturing Engineering, Loughborough University, Loughborough, LE11 3TU, UK
Tel: +44(0)1509227662
Fax:+44(0)1509227648
Email: J.A.Harding@lboro.ac.uk

*corresponding author. Email: R.I.Young@lboro.ac.uk

Abstract

This paper proposes a particular method which utilizes the Unified Modeling Language (UML) as a design visualization tool for modelling ontologies based on the Common Logic knowledge representation language. The use of this method will enable Common Logic ontological concepts to be more readily accessible to general engineers and provide a valuable ontology design aid. The method proposed is explored using the Knowledge Frame Language (KFL) which provides constructs to facilitate ontology building and is built on Common Logic. The major constructs of KFL are briefly defined and a description of how each construct may be represented in UML is given. Examples are presented showing how the constructs may be modeled in UML and a Common Logic implementation based on a UML design is illustrated and discussed. The manufacturing domain is utilized as an experimental basis for demonstrating the proposed method.

Keywords:

Ontology design tools, Ontology visualization methods

1. Introduction

Common Logic is an ISO standard language based on first order logic (ISO/IEC 24707, 2007). Common Logic Interchange Format (CLIF) provides a syntax for Common Logic. Common Logic provides a rich structural representation method for ontologies. The advantages of Common Logic compared to popular knowledge representations such as RDF (Brickley and Guha, 2004) and OWL (Smith et al., 2004) are that it is more expressive and provides more powerful deductive reasoning capabilities.

The sharing of manufacturing business knowledge aids effective decision making and hence provides better, faster and cheaper products and services. Ontologies to facilitate knowledge sharing are starting to be defined in the manufacturing sector (Cheung et al., 2006; Chungoora and Young, 2008). The use of Common Logic enables the utilization of predefined formal process semantics from the Process Specification language PSL (ISO 18629) thus providing a base set of concepts for a manufacturing ontology. In addition to the relations which can be modeled by OWL, Common Logic enables ternary relations to be captured, allowing for example, the fact to be specified that one process occurs between two other processes within a process sequence.

Design methods exist that define the process of building an ontology. Methodologies have been developed for building ontologies from scratch and for reusing existing ontologies. Pinto and Martins (2004) and Dean et al. (1998) survey ontology building methodologies. Corcho et al. (2003) compare the main methodologies, tools and languages for building ontologies. Ontological design tools facilitate the ontology development process and enable communication by presenting knowledge in a user-friendly way. Katifori et al. (2007) categorize the characteristics and features of ontology visualization methods. However, there are not many tools available which are sufficiently expressive to represent the concepts of Common Logic.

Whilst there are no intrinsic difficulties in relating Common Logic with design tools there are only two tools that we have identified that are compatible with Common Logic. This lack of design tools is possibly due to the recent development of Common Logic. The compatible design tools are: conceptual graphs (Sowa, 2000) or an equivalent propositional semantic network (Sowa, 1992) and the IDEF5 schematic language (Knowledge Based Systems Inc., 1994). A propositional semantic network is a graphical notation for representing knowledge in the abstract (as a proposition). A conceptual graph is a bipartite graph with two kinds of nodes, called concepts (depicted as boxes) and conceptual relations (depicted as circles). Conceptual graphs have an expressive power equal to that of a first order logic language.

The IDEF5 schematic language is one of two ontology languages supporting the IDEF5 Ontology Description Capture Method, the other being the IDEF5 elaboration language. The IDEF5 schematic language is a graphical language which enables domain experts to model the most common forms of ontological information. Users are able to develop ontologies from the beginning and to revise existing ontologies. The IDEF5 schematic language models the basic elements that exist in a domain, their distinguishing properties and salient relations. More detailed characterization of the elements in the ontology is captured by the IDEF5 elaboration language, a structured textual language. The IDEF5 Method Report argues that a graphical language is needed to form an intuitive interface for entering basic ontological information, however for entering more complex information graphical representations are considerably more cumbersome than their standard linear counterparts. This is the reason the IDEF5 ontology development process does not utilize conceptual graphs but defines two ontology languages, a graphical one and a textual one. A problem with the IDEF5 schematic language it that it has limited graphical visualization tools.

UML (Unified Modeling Language) (OMG, 2009) is a visual modeling language used to describe object-oriented designs. UML is widely used and has numerous support tools, e.g. Rational Rose (IBM Software), Enterprise Architect (Sparx Systems), ArgoUML (Tigris.org), etc. As UML is the standard way to represent software designs this paper will consider how it can be used to represent ontologies based on Common Logic. The use of UML will enable ontological concepts to be more readily accessible to general software engineers. For engineers skilled in object-oriented concepts UML will provide an intuitive design tool and facilitate the ontology construction process.

One of the first people to suggest UML to model ontologies was Cranefield (2001). Cranefield mapped UML concepts to RDF(S). RDF(S) extends RDF with frame-based primitives. Gasevic et al. (2009) describe and review the limitations of existing efforts in using UML to visualize ontological representation systems. These approaches utilize outdated UML specifications. Other related work is that of Cali et al. (2002), Berardi et al. (2005) and Fillottrani et al. (2006) who describe a framework based on description logic for reasoning on UML class diagrams. None of the approaches consider how UML may be used to represent Common Logic.

The aim of this paper is to describe how a Common Logic-based ontology may be depicted in UML diagrams. An explanation is provided as to why Common Logic is needed to represent a manufacturing ontology. Implementation methods are considered, examples of ontological constructs are given and a description of how the constructs may be represented in UML is provided. The reason for considering UML as a design tool for ontologies is to utilize it as an aid to represent core manufacturing concepts. To support the development of integrated interoperable systems it is important that easy to interpret design support tools are available. A UML design describing a Common Logic ontology for a manufacturing domain

is presented to demonstrate the applicability of UML as a preliminary design tool for Common Logic-based ontologies.

2. Why Common Logic is Needed

Possible alternative heavyweight ontological representations to Common Logic are:

- Frames plus a First Order Logic constraint language (Gómez-Pérez et al, 2004), implementable as Protégé Frames with the Protégé Axiom Language (PAL) as the heavyweight ontological formalism in the Protégé version 3.4 ontology environment (Protégé Website, 2011).
- Description Logics (Gómez-Pérez et al, 2004) implementable as the Web Ontology Language (OWL 2; W3C, 2009) with the rule language SWRL (Semantic Web Rule Language; Horrocks et al., 2004).

Extended CLIF (ECLIF) (ECLIF Reference, 2010), the flavor of CLIF used to structure ontologies in this paper, is able to express manufacturing process semantics more directly and is more efficient in execution than the alternative rule languages above (more details are available in Chungoora, 2010). PAL and ECLIF enable integrity constraints to be specified but SWRL does not. Integrity constraints place restrictions on the constructs present in an ontology and the ways in which the knowledge base associated to the ontology can be populated, aiding in the prevention of modelling mistakes. Integrity constraints are arguably more intuitive for IT professionals with Software Engineering and Database Systems backgrounds than the value restriction approach applied by OWL which is used to derive additional information about OWL property values (de Bruijn et al, 2005). To define process sequences in manufacturing requires formal methods of placing constraints on the sequence order (Young et al, 2007). Integrity constraints are also required to enable

computational comparisons of the meaning of terms, facilitating process interoperability. This heavyweight approach provides confidence that the real meaning behind terms coming from different systems is the same (Das et al, 2007).

The advantages of Common Logic are that are it is good at modelling complex domains and is supported by an environment that can scale to very large data sets and support proof structures. The above alternative representations have limited modelling capabilities. Neither representation is able to capture ternary relations or ontological functions (Chungoora, 2010). The need for these constructs within the manufacturing domain is demonstrated by the example manufacturing ontology (section 5 "Applying the Method to a Manufacturing Ontology"). A more powerful representation is required to address the formal semantics of the manufacturing domain. Description Logic approaches are limited to a hierarchical data model in order to achieve favourable computational properties (Orsi and Tanca, 2010)

The Common Logic environment utilised by this research provides efficient reasoning over complex models and large data sets, being capable of storing up to 2,000,000 facts (Highfleet.com). Ontology languages based on Description Logics suffer from scalability problems when answering queries (de Bruijn et al, 2005). Optimising approaches focus on reducing the expressivity of the ontology language in order to obtain formal tractability guarantees (Horrocks et al, 2010).

The Common Logic environment is able to provide a "proof view" which displays a list of the facts used to infer a query result and the chain of reasoning used to derive the result (Highfleet Tools User Manual, 2010). The reasoning chain is obtained from a combination of the facts used to infer the result together with the underlying ontology structure and axioms. The use of a proof structure will increase user confidence in the results obtained from the knowledge base. The reasoner Pellet indicates how Description Logic

inferences are computed (Clark&Parksia). However, to achieve a concise, intuitive proof from an ontology represented in Description Logics which is based on the open world assumption (i.e. the presumption that what is not currently known to be true, is undefined) would be difficult.

Whilst there is a broad range of research groups working with Description Logics indicating this approach is of value, the view of this research is that the manufacturing domain needs the greater capabilities of Common Logic to model its inherent complexity, as indicated by the arguments above. Whatever approach to represent an ontology is taken, design tools are needed to facilitate ontology development and this is the primary purpose of this paper.

3. Implementation and Scope

ECLIF is designed to support database engineering applications which generate deductive databases. The Knowledge Frame Language (KFL) (KFL Reference, 2010) provides constructs to facilitate ontology building and is based on ECLIF and frame-based primitives. KFL rationalizes ECLIF from a parenthesis-heavy syntax which is confusing for some users to a more structured set of concepts with 90% of the functionality of ECLIF. The use of the common KFL representation enables interoperability at the semantic level. KFL constructs are more readable than ECLIF and facilitate code reuse, thus enabling more efficient use of a programmer's time and allowing smaller, consistent and more manageable ontologies to be built. However, to capture specialized ontological relationships requires the use of ECLIF. As KFL is used to implement a query answering system it is based on a closed world assumption (personal communication, HIGHFLEET Inc.), i.e. the presumption that what is not currently known to be true, is false. KFL makes the unique name assumption which states that different names refer to different concepts.

UML is not sufficiently expressive to represent ECLIF axioms, equating with the IDEF5 schematic language in terms of expressive power and, like the IDEF5 schematic language, is useful for the construction of first-cut ontologies. UML has its own constraint language, OCL (Object Constraint Language) (OMG, 2010), but this is unable to provide any advantages with regards to axiom capture and design. It is therefore proposed that axiom constructs be defined within ECLIF but that the lightweight ontology design is produced in UML to a specification which maps directly to the KFL constructs. Lightweight ontologies classify concepts and define relationships between the concepts. Heavyweight ontologies add axioms and constraints to lightweight ontologies.

How best to use UML to describe the major KFL constructs is discussed and particular methods proposed – for reasons of brevity detailed directives are not considered. UML version 2.2 will be referenced throughout this paper, unless indicated otherwise. Example constructs from the UML design describing a Common Logic-based ontology for a manufacturing domain are implemented in KFL.

4. Methods for Mapping KFL Constructs to UML Concepts

UML provides a high level design for the overall ontology, enabling the capture of key KFL constructs. The major constructs of KFL are:

- Properties
- MetaProperties
- Relations
- Functions

Properties are the components used to structure a KFL taxonomy. KFL properties equate to classes in standard ontological terminology (Gruber, 1993). A MetaProperty is a

special type of property which can only be instantiated as a property. Relations link instances of properties together, allowing information to be created about the concepts the properties represent. KFL allows instances to be related to each other and to primitive data values. The knowledge representation language OWL (W3C, 2009) uses the term "property" to describe this construct. Functions provide an abbreviated structure for specifying information for property instances.

KFL constructs take the form of directives which are expressed as a colon at the beginning of the line, followed by a keyword and one or more arguments referring to elements of the ontology, e.g. ":Prop Vehicle" where "Prop" is a keyword defining a property construct and "Vehicle" refers to the property ontology element.

The next sub-section considers the use of common UML symbols to describe KFL constructs. Constructs requiring the use of more advanced UML notations are described in the subsequent sub-sections.

4.1 Mapping KFL Constructs to UML Symbols in Common Use

This sub-section briefly considers how the property construct and some simple relation constructs can be related to well known UML symbols. An enhancement to the project construct (sub-section 4.2 Disjoint Properties) and more types of relations are described later (sub-sections "4.4 Ternary Relations", "4.5 Super-relations", "4.6 Second Order Relations"). Properties may be mapped to UML classes as both are defined as categories with a Boolean property (HIGHFLEET System v4.0 Browse File, OMG Unified Modeling Language Superstructure Version 2.2, Figure 7.9 - Classifiers diagram of the Kernel package and Figure 7.12 - Classes diagram of the Kernel package).

Examples of two properties are given below.

```
;;; Examples of Properties
```

```
:Prop Vehicle
:Inst Type
:sup Object

:Prop Car
:Inst Type
:sup Vehicle
```

In KFL a semi-colon indicates a comment. Properties must contain the following directives: ":Prop", ":Inst" and ":sup". ":Prop" defines the property identifier, e.g. "Vehicle". ":Inst" describes what the property instantiates. In the above examples the properties are instances of Type. A "Type" instance exists permanently, for example, an instance of `Vehicle` is always a `Vehicle`, it will not metamorphose. ":sup" describes how properties are organized by subsumption, e.g. `Vehicle` subsumes `Car`. The property "Object" is a KFL base entity.

Figure 1 a. shows how the two example properties may be displayed in UML.

(Fig. 1. about here)

UML classes are depicted as rectangles. A line with a hollow arrowhead represents generalization, with the arrow pointing to the more general class, e.g. `Car` is a subtype of `Vehicle`.

KFL contains a range of types of relations, of which Unary and Binary will be considered in this sub-section. Unary relations have one argument position. Unary relations can provide additional information about a property without the need to create an additional property. A unary relation declaration in KFL is as follows:

```
;;; Unary Relation Example

;;; Related Property
:Prop Feature
:Inst Type
:sup Object

;;; Relation
```

```
:Rel compoundFeature
:Inst UnaryRel
:Sig Feature
```

Like properties, relations have three required fields in a declaration. Relations must contain the following directives: ":Rel", ":Inst" and ":Sig". ":Rel" defines the identifier of the relation. ":Inst" describes the kind of relation instantiated, in this case it's unary. The ":Sig" directive defines the signature of the relation by stating a property for each argument position, so here there is only one property given ("Feature"). A unary relation may be described in UML as a Boolean attribute (see figure 1b).

Attributes are specified within a separate compartment of the class rectangle. The class `Feature` contains the Boolean `compoundFeature`.

Binary relations relate two property instances and are the most common kind of relation. A simple binary relation example is given below. As the relation is binary two properties are specified for the ":Sig" directive.

```
;;; Simple Binary Relation Example

;;; Related Property

:Prop Person
:Inst Type
:sup Object

;;; Relation

:Rel cousinOf
:Inst BinaryRel
:Sig Person Person
```

Binary relations may be represented by UML associations, as in figure 1 c.

Associations represent the relationships between instances (of classes) and are represented as a line between classes. The line can be named to describe the association. Fig.

1 c. includes the alternative method of drawing a UML diagram when a relationship exists between instances of the same class.

Specialized kinds of binary relations exist of which symmetric and asymmetric binary have been considered. Symmetric binary relations specify the deduction if Rel(a,b) then Rel(b,a). Specialised binary relations are declared with the ":Inst" directive. The simple binary relation example can be more specifically defined as:

```
;;; Symmetric Binary Relation Example

:Rel cousinOf
:Inst SymmetricBR
:Sig Person Person
```

For example, if Jane Smith is `cousinOf` John Smith, then John Smith is `cousinOf` Jane Smith.

Asymmetric binary relations state that if Rel(a,b) then not Rel(b,a).

```
;;; Asymmetric Binary Relation Example

:Rel parentOf
:Inst AsymmetricBR
:Sig Person Person
```

For example, if Mary Brown is the `parentOf` Jack Brown, Jack Brown cannot be the `parentOf` Mary Brown. Figure 1 d. represents the asymmetric binary relation example in UML.

A UML note (the dog-eared rectangle) is used to express specialized binary relations. A UML note symbol is used to render constraints or comments. As the relation is asymmetric it is only navigable in one direction. In UML the open arrowhead indicates the association is unidirectional.

4.2 Disjoint Properties

By default in KFL two properties may share a common instance. The optional directive

“disjointWith” is used to indicate whether two properties are disjoint (have no instances in common). A simple ontology example including two disjoint properties is shown below.

```
;;; Disjoint Property Example

:Prop MalePerson
:Inst Type
:sup Person

:Prop FemalePerson
:Inst Type
:sup Person
:disjointWith MalePerson

:Prop Employee
:Inst MaterialRole
:sup Person

:Prop Manager
:Inst MaterialRole
:sup Person
```

Applying this example ontology, an instance of a `Person` can be classified as a `MalePerson` or a `FemalePerson`, an `Employee` and a `Manager`. An instance cannot be both a `MalePerson` and a `FemalePerson` as these two properties are disjoint. A“MaterialRole” instance is transient and may be associated with a time interval. For example, a `Person` can be an `Employee` for a certain period, become unemployed and then become an `Employee` again.

UML 2 clarifies the representation of disjoint properties. In UML 2 the constraint “disjoint” is added to a generalization arrow to indicate classes which have no common instances. The generalization arrow is labeled with the constraint “overlapping” to indicate classes which share common instances. UML utilises a different default to KFL - in UML by default classes are disjoint. Figure 2. shows how the Disjoint Property example may be represented in UML.

(Fig.2. about here)

UML constraints are denoted by text enclosed in braces ({}). To avoid confusion and enable direct mapping of UML to KFL the default disjoint constraint is labeled.

4.3 MetaProperties

MetaProperties are properties whose instances may only be properties, enabling subsets of property instances to be defined. Subsets of property instances are useful in defining specialized relation types. As car models are a widely understood domain these will be used to provide a MetaProperty example.

```
;;; MetaProperty Example

:Prop FordCarModel
:Inst MetaProperty
:sup Type

;;; Required Property

:Prop FordCar
:Inst Type
:sup Object

;;; Instances of the FordCarModel MetaProperty

:Prop Focus
:Inst FordCarModel
:sup FordCar

:Prop Fusion
:Inst FordCarModel
:sup FordCar

:Prop Fiesta
:Inst FordCarModel
:sup FordCar
```

The concept of MetaProperties is illustrated in figure 3.

(Fig.3. about here)

`MyCar` is an instance of the property `FordCar` and would be defined in a knowledge base

as an ECLIF assertion (fact), e.g. `(FordCar MyCar)`. The property `Fusion` is subsumed by `FordCar`, but `Fusion` is also an instance of the MetaProperty `FordCarModel`. A grey ellipse is used in the illustration to denote a concept that is required to be simultaneously modeled as a property and an instance (notation of Henderson-Sellers and Gonzalez-Perez, 2005). `FordCar`, classified as a `Fusion`, has an instance `MyCar`.

UML employs a plain line to indicate a relationship. The line ends may be annotated with a multiplicity value which indicates how many instances may participate in the relationship. '*' means "zero or more" (see figure 3.). How multiplicities are modeled in KFL will not be considered in this paper.

Representing metaproperties in UML requires the detailed power type description given in UML 2. In UML a power type is a class whose instances are subclasses of another class. Power types are metaclasses whose instances can also be subclasses. As KFL properties may be mapped to UML classes MetaProperties may be represented by power types, as demonstrated in figure 4.

(Fig.4. about here)

A power type is specified by placing an indicator colon together with the name of the power type next to the set of classes that are instances of the power type. Fig. 4. shows that each `FordCar` can be subclassed as either a `Focus` or a `Fusion` or a `Fiesta`. Furthermore, `Focus`, `Fusion` and `Fiesta` are instances of `FordCarModel`, thus `Focus`, `Fusion` and `Fiesta` are both classes and instances (of the power type). This can be seen to relate to the MetaProperty example given above.

4.4 Ternary Relations

Ternary relations have three argument positions, as shown in the following example.

```
;;; Ternary Relation Example
```

```
;;; Related Properties

:Prop Company
:Inst Type
:sup Object

:Prop Part
:Inst Type
:sup Object

;;; Relation

:Rel supplies
:Inst TernaryRel
:Sig Company Part Company
```

For example, `Company` Rolls Royce `supplies Part` engine to `Company` BAE Systems. Ternary relations can be depicted in UML by n-ary associations (see figure 5.). N-ary associations exist between three or more classes.

(Fig.5. about here)

N-ary associations are described in UML by the diamond notation. Association ends can be labelled with a role name which provides some semantics about the nature of the association end. The KFL argument positions are represented by role names (e.g. arg1). Each argument is given the prefix of "arg" to avoid confusion with UML multiplicities.

4.5 Super-relations

Super-relations define specializations of relations and can be used to form a hierarchy of relations. The relationship between a relation and its corresponding super-relation is defined with the additional directive ":supRel" (see below).

```
;;; Super-relation Example

;;; Super-relation

:Rel relatedTo
:Inst SymmetricBR
:Sig Person Person
```

```
;;; Relation

:Rel cousinOf
:Inst SymmetricBR
:Sig Person Person
:supRel relatedTo
```

For example, if Jane Smith is `cousinOf` John Smith, then it can be deduced that Jane Smith is `relatedTo` John Smith. In UML generalization may be applied to associations as well as to classes, hence the super- relation example can be represented as shown in figure 6.

(Fig.6. about here)

To avoid the confusion of connecting lines the notation used represents each association as an association class and draws the generalization arrow between the rectangles for the association classes (UML 1.4 notation, neither specified nor deprecated in UML 2.2). Association classes enable more information to be provided about associations, in this case hierarchical information.

4.6 Second Order Relations

The term “second order relation” is defined in the IDEF5 Method Report (Knowledge Based Systems Inc., 1994). A second-order relation is a relation that connects two properties or a property and an instance. For example, the relation “has-more-instances-than” is a relation that holds between two properties. The “instance-of” relation holds between a property and an instance. The “instance-of” relation is of such importance in modelling that it is represented explicitly in both KFL and UML. In KFL it is captured by the “:Inst” directive. UML represents the “instance-of” relation as its inverse, indicated by a dashed arrow labelled with the keyword “<<instantiate>>” as in figure 3.

A real world example of a second order relation is illustrated in figure 7. A relation between an instance and a property is shown as this is the most complex type of second order relation considered in this paper.

(Fig.7. about here)

Ford (an instance of the property Company) `manufactures` `Fiesta`(s) (a property). The relation specifies Fiestas in the abstract, it does not describe which specific Fiesta is manufactured.

As noted previously, relations link instances of properties. To model second order relations in KFL requires the use of metaProperties to express connections to properties (instances of metaProperties). The real world second order relation can be modeled in KFL as:

```
;;; Second Order Relation Example

;;; Related Properties
:Prop Fiesta
:Inst FordCarModel
:sup FordCar

:Prop Company
:Inst Type
:sup Object

;;; Relation (connects a property and an instance)

:Rel manufactures
:Inst AsymmetricBR
:Sig Company FordCarModel
```

`FordCarModel` and `FordCar` are defined in the MetaProperty example (see previous). To instantiate the `manufactures` relation the following ECLIF assertions need to be specified in a knowledge base.

```
(Company Ford)
(manufactures Ford Fiesta)
```

UML 2.2 takes the view that instances of classifiers (metamodels) can be classifiers. Fig. 8.

represents the second order relation example in UML.

(Fig.8. about here)

As shown in figure 4. `FordCarModel` is a power type. A note has been added to figure 8. to clarify that `FordCarModel` is a power type and which classes are instances of the power type. To provide more explanation the second order relation example is instantiated in figure 9.

(Fig.9. about here)

Instances are specified in UML by an underlined construct consisting of the instance name, a colon and the class name. Figure 9. shows an instance of `Company`, Ford, that `manufactures` an instance of `FordCarModel`, `Fiesta`. However, because `FordCarModel` is a power type, `Fiesta` is also a class. Therefore a second order relation exists between an instance, Ford, and a class, `Fiesta`.

4.7 Functions

Functions use arguments to refer to a property and return an instance of the property. A function term is an expression consisting of the function name and argument sequence. The function term denotes the property instance corresponding to the value of the function. For example, the function term (kg 25.0) could be used to refer to a particular instance of the property "MassQuantity". The function term (2D_Coordinates 12.10 13.50) could be used to refer to a particular instance of the "Location" property. Some function examples are given below.

```
;;; Examples of Functions

;;; Referred Properties

:Prop MassQuantity
:Inst Type
:sup Object
```

```
:Prop GeoLocation
:Inst Type
:sup Object

:Prop SpatialLocation
:Inst Type
:sup Object


;;; Functions
:Fun kg
:Inst UnaryFun
:Sig RealNumber -> MassQuantity

:Fun 2D_Coordinates
:Inst BinaryFun
:Sig RealNumber RealNumber -> GeoLocation

:Fun 3D_Coordinates
:Inst TernaryFun
:Sig RealNumber RealNumber RealNumber -> SpatialLocation
```

Like properties and relations functions have three required fields. Functions must contain the following directives: ":Fun", ":Inst" and ":Sig". ":Fun" provides an identifier for the function, e.g. "3D_Coordinates". ":Inst" classifies the function by the number of parameters that need to be specified for the function. Examples are provided above of a unary, binary and ternary function. ":Sig" specifies types of property for the function arguments. The ":Sig" directive consists of two parts: values on the left of the arrow indicate the function argument type; the value on the right of the arrow is the property instantiated by the entire function term. So in the case of the `3D-Coordinates` function all the parameters must be real numbers and the whole term, for example (3D_Coordinates 4 45 67), is an instance of `SpatialLocation`.

There is no direct equivalent to a function in UML as functions are a KFL implementation shorthand technique. For example, the information described by the `kg` function could also be modeled as:

```
;;; Alternative Method of Modeling Function Information
```

```
;;; Required Properties

:Prop MassQuantity
:Inst Type
:sup Object

:Prop Unit
:Inst Type
:sup Object

;;; Required Relations

:Rel hasAmount
:Inst BinaryRel
:Sig  MassQuantity RealNumber

:Rel hasUnit
:Inst BinaryRel
:Sig  MassQuantity Unit
```

The following ECLIF assertions would be needed to define an instance of “MassQuantity”.

```
(MassQuantity myMass)
(Unit kg)
(hasAmount myMass 25.0)
(hasUnit myMass kg)
```

It can be seen that this method is much more complex, requiring an additional property `Unit` and two relations to be specified.

KFL functions can be represented in UML by the use of stereotypes. A stereotype extends a UML model, enabling the use of platform or domain specific terminology. Figure 10. shows how the examples of KFL functions may be depicted in UML.

(Fig.10. about here)

In UML a stereotype keyword (indicated by guillemets) can be used to list groups of elements in a class (UML 1.4 notation, neither specified nor deprecated in UML 2.2). The stereotype keyword is used to identify the KFL function name and type (e.g. unary function). The function name is partitioned from the type by a colon. The attributes listed beneath the stereotype form the parameters of the function.

5 Applying the Method to a Manufacturing Ontology

Core concepts are being developed for a manufacturing ontology. A sub-set of these concepts is presented in figure 11 to illustrate how UML may be utilized as a design tool for a Common Logic-based ontology. The example ontology is sufficiently complex to contain all of the KFL constructs discussed above. Examples of the constructs are pointed out within the UML design and implementation in KFL is shown.

(Fig.11. about here)

The example ontology contains several properties and a complex metaProperty example demonstrating generalization of metaProperties. Figure 11 shows that `MachineToolType` subsumes `MillingMachineType` and `LatheMachineType` and `MachineTool` subsumes `MillingMachine and LatheMachine`. The ":machineToolType" power type annotation indicates that `MillingMachine and LatheMachine` are instances of `MachineToolType.` `LatheMachine` subsumes `ManualLatheMachine and AutoLatheMachine` which are shown by the power type annotation to be instances of `LatheMachineType.` Thus a `ManualLatheMachine` is an instance of a `LatheMachineType` and a sub-class of a `LatheMachine` which is an instance of a `MachineToolType` and a sub-class of a `MachineTool`. None of the tools and tool types share common instances. The metaProperty section described of the UML design for the manufacturing ontology is implemented in KFL as follows:

```
;;; MetaProperty Section of Manufacturing Ontology
;;;MetaProperties

:Prop MachineToolType
:Inst MetaProperty
:sup Type
```

```
:Prop LatheMachineType
:Inst MetaProperty
:sup MachineToolType

:Prop MillingMachineType
:Inst MetaProperty
:sup MachineToolType
:disjointWith LatheMachineType

;;; Required Properties

:Prop Resource
:Inst Type
:sup Object

:Prop MachineTool
:Inst Type
:sup Resource
;;;disjoint directives not shown for brevity

;;; Instances of the MachineToolType MetaProperty

:Prop LatheMachine
:Inst MachineToolType
:sup MachineTool

:Prop MillingMachine
:Inst MachineToolType
:sup MachineTool
:disjointWith LatheMachine

;;; Instances of the LatheMachineType MetaProperty

:Prop ManualLatheMachine
:Inst LatheMachineType
:sup LatheMachine

:Prop AutoLatheMachine
:Inst LatheMachineType
:sup LatheMachine
:disjointWith ManualLatheMachine
```

The example manufacturing ontology includes the following examples of relations concerning arity. The manufacturing ontology contains a unary relation, described by the Boolean attribute `machineOperator` contained by class `Human,` and several binary relations relating properties to primitive data values. UML describes relationships between

classes and simple data values as attributes, e.g. the class `MachineTool` contains the attribute `number_of_axes:int`. As previously discussed, UML represents relationships between classes as associations. The example ontology contains an asymmetric binary relation, `isLocatedIn,` and a ternary relation, `participatesIn`. The relations are modeled in KFL as:

```
;;; Examples of Arity Relations in the Manufacturing Ontology
;;; Related Properties

:Prop Human
:Inst Type
:sup Resource
;;;disjoint directives not shown for brevity

:Prop Facility
:Inst Type
:sup Object

:Prop MachiningOperation
:Inst Type
:sup Object

;;; Unary Relation Example

:Rel machineOperator
:Inst UnaryRel
:Sig Human

;;; Example of Binary Relation between a Property and a
;;; primitive data value

:Rel number_of_axes

:Inst BinaryRel
:Sig MachineTool IntegerNumber
;;;MachineTool property defined in MetaProperty Section above

;;; Asymmetric Binary Relation Example

:Rel isLocatedIn
:Inst AsymmetricBR
:Sig Resource Facility

;;; Ternary Relation Example

:Rel participatesIn
```

```
:Inst TernaryRel
:Sig Resource MachiningOperation TimeInstant
;;; TimeInstant is  KFL base entity
```

The example manufacturing ontology includes special relations. Fig. 11. shows a super-relation hierarchy linking `Resource` to `MachiningOperation`. The super-relation `usedIn` subsumes `isConsumedIn` and `isReusedIn`. A second order relation, `containsToolType` connects `Facility` to `MachineToolType`. Both the super-relation and the second order relation are asymmetric binary relations. The special relations are implemented in KFL below.

```
;;; Examples of Special Relations in the Manufacturing
;;; Ontology

;;; Super-relation Example

:Rel usedIn
:Inst AsymmetricBR
:Sig Resource MachiningOperation
;;;Properties Resource MachiningOperation and MachineTool
;;;defined in Arity Relations example above

;;; Relations

:Rel isConsumedIn
:Inst AsymmetricBR
:Sig Resource MachiningOperation
:supRel usedIn

:Rel isReusedIn
:Inst AsymmetricBR
:Sig Resource MachiningOperation
:supRel usedIn

;;; Second Order Relation Example
:Rel containsToolType
:Inst AsymmetricBR
:Sig Facility MachineToolType
;;;Facility property defined in Arity Relations example above
;;;MachineTool property defined in MetaProperty Section above
```

The second order relation `containsToolType` links an instance of a property to a

property. An example instance of the `containsToolType` relation is specified by the ECLIF assertions:

```
(Facility Workshop_1)
(containsToolType Workshop_1 MillingMachine)
```

It should be noted that the example manufacturing ontology includes a similar relation, `containsTool`, which is a ordinary asymmetric binary relation connecting two instances of properties together (see figure 11.). An example specification of the `containsTool` relation is:

```
(Facility Workshop_20)
(ManualLatheMachine MLMachine_2)
(containsTool Workshop_20 MLMachine_2)
```

An example of a function is provided by the `SpindleSpeed` class which contains the function `revs_per_min`, modeled in KFL as:

```
;;; Manufacturing Ontology Function Example
:Fun revs_per_min
:Inst UnaryFun
:Sig RealNumber -> SpindleSpeed
```

6 Conclusions

A demonstration has been provided showing how KFL constructs can be captured through a considered application of UML. To fully represent all the major constructs of KFL requires the use of UML version 2. By utilizing some of the less well known modeling concepts of UML, such as n-ary associations and power types, it is possible to use UML as a design visualization tool for KFL ontologies. As UML is widely understood and has many support tools, the ability to use it as a design tool will facilitate ontology development.

The authors recognize that KFL is a proprietary approach but believe that any attempt to organize Common Logic axioms would result in a similar set of constructs to those

provided by KFL. We consider that the method described in this paper could be applied to any Common Logic-based ontology.

UML is able to represent lightweight ontologies. However, as shown by the example manufacturing ontology, considerable functionality can be captured. No attempt has been made to capture axioms and constraints as these are awkward to represent graphically. OCL may be used to describe rules that apply to UML models, however as CLIF is an international standard which provides a common notation there are no advantages in defining Common Logic axioms in OCL. The proposed approach uses UML to model domain concepts and their relationships and ECLIF to define axiom constructs.

The Common Logic code examples shown have been manually generated, but if sufficient interest was available within the information modeling community a tool could be implemented that automatically generates ontologies based on Common Logic from UML designs.

An approach has been developed to utilize UML as a design tool for Common Logic and an example ontology design depicted in UML has been described. This approach will be further developed and exploited to design ontologies for manufacturing knowledge sharing. The authors believe that as the power of Common Logic is required to capture the semantics of the manufacturing domain the approach defined in this paper will form a key enabler in manufacturing knowledge sharing.

Acknowledgements

We wish to thank the EPSRC, who have funded the work behind this paper through two related digital manufacturing projects, project 253 and project 237, of the Loughborough University Innovative Manufacturing and Construction Research Centre.

References

Berardi, D., Calvanese, D., De Giacomo, G., 2005. Reasoning on UML Class Diagrams is EXPTIME-hard. *Artificial Intelligence,* 168(1-2), 70-118. ISSN: 0004-3702.

Brickley, D. and Guha, R.V., 2004. RDF Vocabulary Description Language 1.0: RDF Schema. [Online] Available from: http://www.w3.org/TR/2004/REC-rdf-schema-20040210. [Accessed 11 October 2011].

Cali A., Calvanese D., De Giacomo G. and Lenzerini M., 2002. A formal framework for reasoning on UML class diagrams. *Lecture Notes in Artificial Intelligence*, 2366, 503-513 .

Clark&Parsia, Pellet: OWL 2 Reasoner for Java (version 3) [Online] Available at: http://clarkparsia.com/pellet [Accessed 11 October 2011].

Cheung, W.M., Bramall D.G., Maropoulos P.G., Gao J.X., Aziz H., 2006. Organisational knowledge encapsulation and re-use in collaborative product development. *International Journal of Computer Integrated Manufacturing*, 19 (7), 736-750, ISSN 0951-192X.

Chungoora, N. and Young, R.I.M., 2008. Semantic Interoperability Requirements for Manufacturing Knowledge Sharing. *Enterprise Interoperability III - New Challenges and Industrial Approaches*. Mertins, K., Ruggaber, R., Popplewell, K., and Xu,X (eds.), Springer-Verlag London Limited, U.K., 411-422, ISBN 978-1-84800-220-3.

Chungoora, N., 2010. A Framework to Support Semantic Interoperability in Product Design and Manufacture. Ph.D. thesis, Loughborough University, U.K.

Cranefield, S., 2001. Networked Knowledge Representation and Exchange using UML and RDF. *Journal of Digital Information,* 1 (8). [Online] Available at: http://journals.tdl.org/jodi/article/view/30/31 [Accessed 11 October 2011].

Corcho, O., Fernandez-Lopez, M. Gomez-Perez, A., 2003. Methodologies, tools and languages for building ontologies. Where is their meeting point? *Data & Knowledge Engineering*, 46 (1), 41-64.

Das, B., Cutting-Decelle, A.F., Young, R.I.M., Case, K., Rahimifard, S., Anumba, C.J. and Bouchlaghem, N., 2007. Towards the understanding of the requirements of a communication language to support process interoperation in cross-disciplinary supply chains. *International Journal of Computer Integrated Manufacturing*. 20(4), 396-410.

de Bruijn, J., Polleres, A., Lara, R. and Fensel, D., 2005. OWL DL vs. OWL Flight: Conceptual modeling and reasoning for the semantic web. *Proceedings of the Ninth World Wide Web Conference*, Japan, May 2005.

ECLIF Reference, 2010. HIGHFLEET Inc., Baltimore, U.S.A.

Fillottrani, P., Franconi, E. and Tessaris, S., 2006. The New ICom. *Proceedings of the 2006 International Workshop on Description Logics (DL2006)*, Windermere, Lake District, UK, May 30 - June 1. [Online] Available at: http://www.inf.unibz.it/~franconi/icom/ [Accessed 11 October 2011].

Gasevic, D., Djuric, D. and Devedzic, V., 2009. *Model Driven Engineering and Ontology Development* (Chapter 6). 2nd ed. Springer-Verlag, Heidelberg, Berlin, Germany.

Gómez-Pérez, A., Fernández-López, M. and Corcho, O., 2004. *Ontological engineering: with examples from the areas of knowledge management, ecommerce and the semantic web*. Springer-Verlag, London, UK.

Gruber, T.R., 1993. A translation approach to portable ontology specifications, Knowledge Acquisition, 5(2), 199-220.

Henderson-Sellers, B. and Gonzalez-Perez, C., 2005. Connecting Powertypes and Stereotypes, *Journal of Object Technology*, 4 (7), 83-96. [Online] Available at: http://www.jot.fm/issues/issue_2005_09/article3 [Accessed 11 October 2011].

Highfleet.com. The XKS Family of Knowledge Servers. [Online, accessed 17th Jan 2011] Available at: http://www.highfleet.com/xks-servers.html

Highfleet Tools User Manual, 2010. HIGHFLEET Inc., Baltimore, U.S.A.

Horrocks, I. , Benedikt, M. Gottlob, G., Lukasiewicz, T. and Motik, B., 2010. ExODA: Integrating Description Logics and Database Technologies for Expressive Ontology-Based Data Access. EPSRC Reference EP/H051511/1. [Online] Available at: http://gow.epsrc.ac.uk/ViewGrant.aspx?GrantRef=EP/H051511/1 [Accessed 11 October 2011].

Horrocks, I., Patel-Schneider, P.F., Boley, H., Tabet, S., Grosof, B. and Dean, M., 2004. SWRL: A Semantic Web Rule Language Combining OWL and RuleML. [Online] Available at: http://www.w3.org/Submission/SWRL/ [Accessed 11 October 2011].

ISO/IEC 24707, 2007. Information technology — Common Logic (CL): a framework for a family of logic based languages.

ISO 18629 (2004) Process Specification language (PSL).

Jones, D., Bench-Capon, T. and Visser, P., 1998. Methodologies for ontology development. *Proceedings of IT&KNOWS (Information Technology and Knowledge Systems) of the 15th IFIP World Computer Congress*, Budapest, Hungary.

KFL Reference, 2010. HIGHFLEET Inc., Baltimore, U.S.A.

Knowledge Based Systems Inc., 1994. Information Integration for Concurrent Engineering (IICE): IDEF5 method report. Texas, USA. [Online] Available at: http://www.idef.com/pdf/Idef5.pdf [Accessed 11 October 2011].

Katifori, A., Halatsis, C., Lepouras, G., Vassilakis, C., and Giannopoulou, E., 2007. Ontology visualization methods—A survey. *ACM Computing. Surveys*. 39, 4, Article 10 (October 2007).

OMG, 2010. Object Constraint Language Version 2.2 [Online] Available at: http://www.omg.org/spec/OCL/2.2/ [Accessed 11 October 2011].

OMG, 2009. OMG Unified Modeling Language (OMG UML), Superstructure and Infrastructure Version 2.2. [Online] Available at: http://www.omg.org/spec/UML/2.2/ [Accessed 11 October 2011].

OMG, 2005. OMG Unified Modeling Language Specification Version 1.4.2. (ISO/IEC 19501) [Online] Available at: http://www.omg.org/cgi-bin/doc?formal/05-04-01 [Accessed 11 October 2011].

Orsi, G. and Tanca, L., 2010. Introduction to the TPLP special issue, logic programming in databases: From Datalog to semantic-web rules. *Theory and Practice of Logic Programming*, 10 (3), 243-250.

Pinto, H.S. and Martins, J.P., 2004. Ontologies: How can They be Built? *Knowledge and Information Systems,* 6, 441–464.

Smith, M.K., Welty, C. And McGuinness, D.L., 2004. OWL Web Ontology Language Guide. [Online] Available at: http://www.w3.org/TR/owl-guide/ [Accessed 11 October 2011].

Sowa, J.F., 1992. Semantic networks, in: S.C. Shapiro (Ed.), *Encyclopedia of Artificial Intelligence*, 2nd ed., Wiley, New York, 1493–1511.

Sowa, J.F., 2000. Knowledge Representation: Logical, Philosophical and Computational Foundations, Brooks Cole, Pacific Grove, CA, U.S.A.

W3C, 2009 OWL 2 Web Ontology Language Primer [Online] Available at: http://www.w3.org/TR/2009/REC-owl2-primer-20091027/ [Accessed 11 October 2011].

Young, R.I.M., Gunendran, A.G., Cutting-Decelle, A.F. and Gruninger, M., 2007. Manufacturing knowledge sharing in PLM: a progression towards the use of heavy weight ontologies. *International Journal of Production Research*, 45(7), 1505-1519.

Figure captions

Fig.1. Basic KFL constructs shown in UML Diagram

Fig.2. An Example of Disjoint and Non-disjoint Properties

Fig.3. FordCar MetaProperty Example (using notation of Henderson-Sellers and Gonzalez-Perez, 2005)

Fig.4. MetaProperty shown in a UML Diagram

Fig.5. Ternary Relation shown in a UML Diagram

Fig.6. Super-Relation shown in a UML Diagram

Fig.7. An Example of a Second Order Relation

Fig.8. The Second Order Relation Example depicted in UML.

Fig.9. An Instantiation of the Second Order Relation Example in UML.

Fig.10. Functions shows in a UML diagram

Fig.11. A Partial Manufacturing Ontology Design

CLIF	ECLIF	KFL

Axiom: Resources are objects
 	
(forall (?r)
  (if (Resource ?r)
  (Object ?r)))
	(Type Resource)
	(argProp Resource 1 Top)
	(arity Resource 1)
	(sup Resource Object)
	(=> (Resource ?__&&3) (Object ?__&&3))
	(=> (Resource @Args) (Object @Args))
	(=> (and (Resource @Args1)
	  (listProj (listof 1)(listof @Args1)
	  (listof @Args2)))
	  (Object @Args2))
		:Prop Resource
		:Inst Type
		:sup Object

Axiom: The machineOperator relation only holds for humans
 	
(forall (?mop)
  (if (machineOperator ?mop)
  (Human ?mop))
	(UnaryRel machineOperator)
	(argProp machineOperator 1 Human)
	(arity machineOperator 1)
		:Rel machineOperator
		:Inst UnaryRel
		:Sig Human

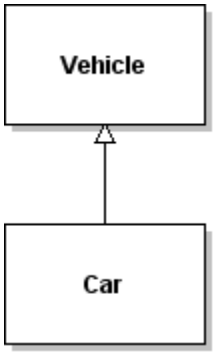
Axiom: The isLocatedIn relation only holds between resources and facilities
 	
(forall (?r ?f)
  (if (isLocatedIn ?r ?f)
  (and (Resource ?r)
  (Facility ?f))))
	(BinaryRel isLocatedIn)
	(argProp isLocatedIn 1 Resource)
	(argProp isLocatedIn 2 Facility)
	(arity isLocatedIn 2)
		:Rel isLocatedIn
		:Inst BinaryRel
		:Sig Resource Facility

Axiom: The participatesIn relation only holds between resources, machining operations and time instants, respectively
 	
(forall (?r ?m ?t)
  (if (particpatesIn ?r ?m ?t)
  (and (Resource ?r)
  (MachiningOperation ?m)
  (TimeInstant ?t))))
	(TernaryRel participatesIn)
	(argProp participatesIn 1 Resource)
	(argProp participatesIn 2 MachiningOperation)
	(argProp participatesIn 3 TimeInstant)
	(arity participatesIn 3)
		:Rel participatesIn
		:Inst TernaryRel
		:Sig Resource MachiningOperation TimeInstant

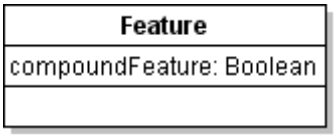
Axiom: A real number value of revolutions per minute is a spindle speed
 	
(forall (?n)
  (if (RealNumber ?n)
  (SpindleSpeed (revs_per_min ?n))))
	(UnaryFun revs_per_min)
	(argProp revs_per_min 1 RealNumber)
	(arity revs_per_min 1)
	(returnProp revs_per_min SpindleSpeed)
		:Fun revs_per_min
		:Inst UnaryFun
		:Sig RealNumber -> SpindleSpeed

TABLE

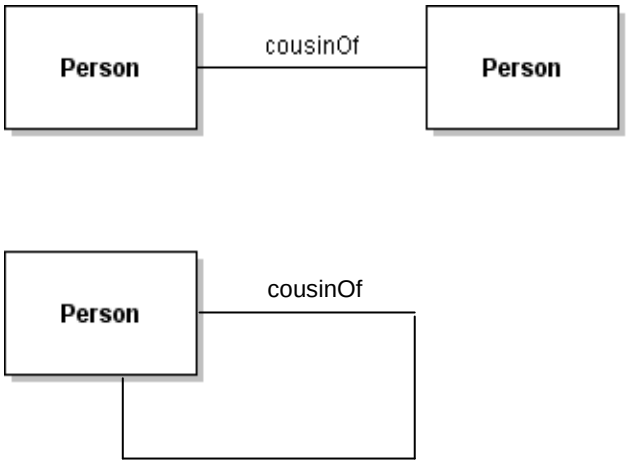
FIGURE 1



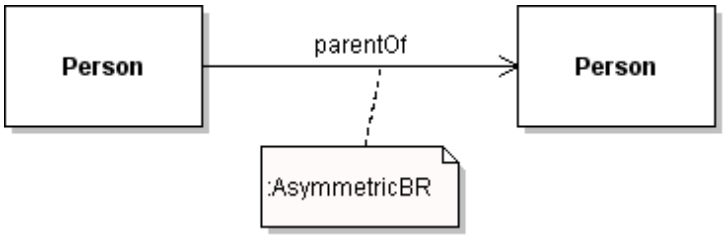
(a) CL properties shown in a UML Class



(b) Unary Relation shown in UML

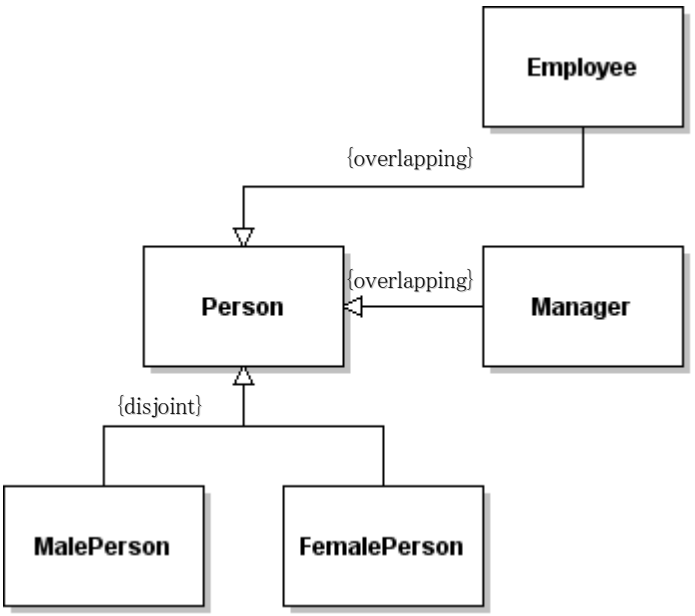


(c) Methods of Representing a Simple Binary relation in UML



(d) Asymmetric Binary Relation shown in UML

FIGURE 2


Employee
{overlapping}
Person
{overlapping}
Manager
{disjoint}
MalePerson
FemalePerson

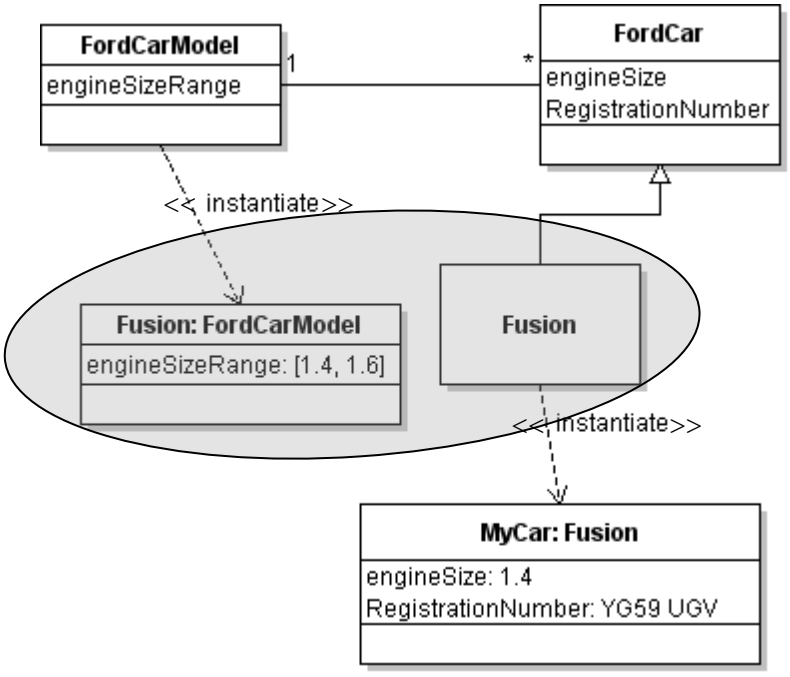
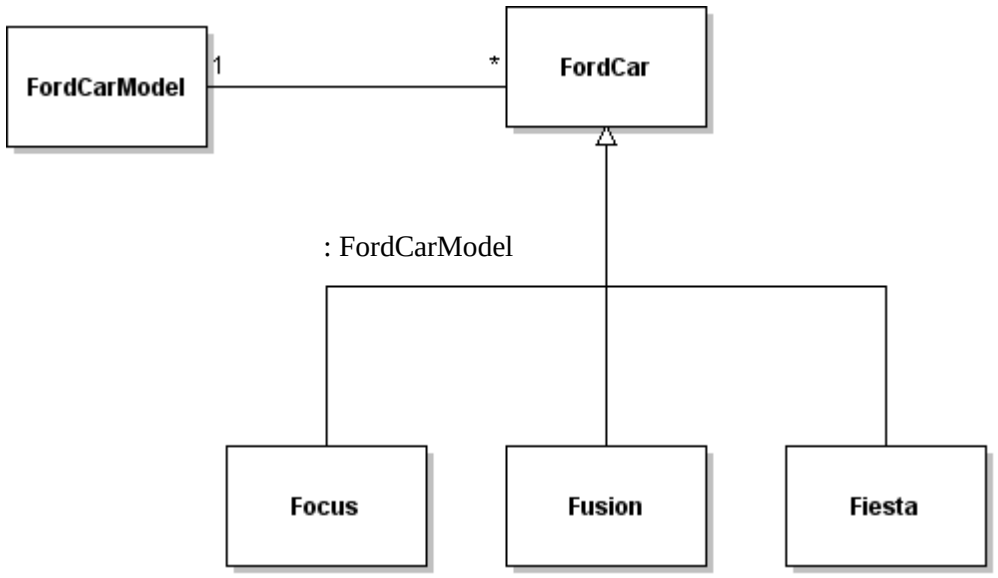

FordCarModel
engineSizeRange
1
*
FordCar
engineSize
RegistrationNumber
<< instantiate>>
Fusion: FordCarModel
engineSizeRange: [1.4, 1.6]
Fusion
<< instantiate>>
MyCar: Fusion
engineSize: 1.4
RegistrationNumber: YG59 UGV

FIGURE 3

FIGURE 4



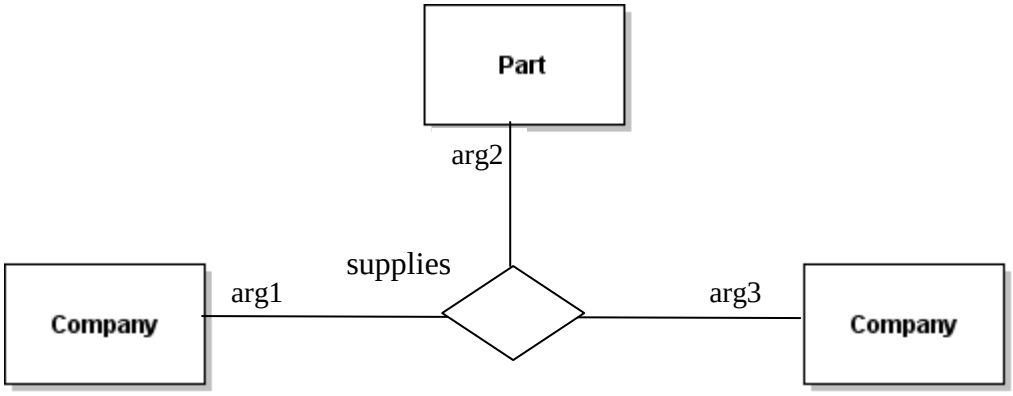
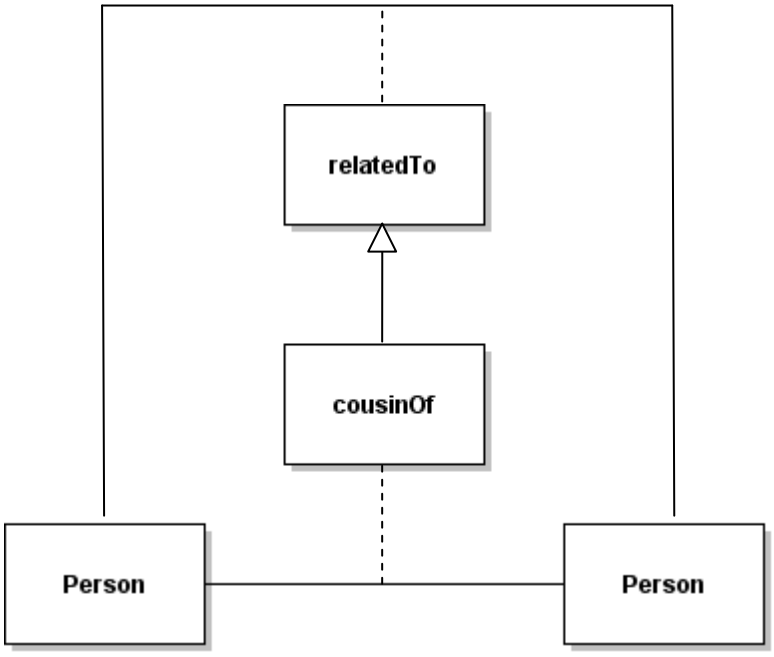


FIGURE 5

FIGURE 6



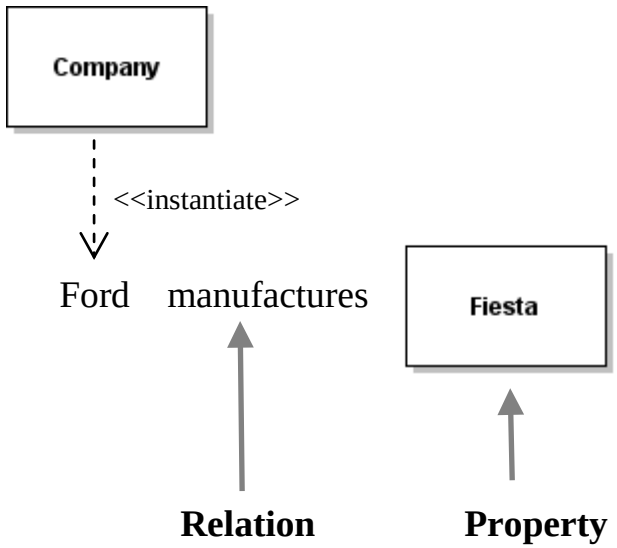

Company
<<instantiate>>
Ford
manufactures
Fiesta
Relation
Property

FIGURE 7

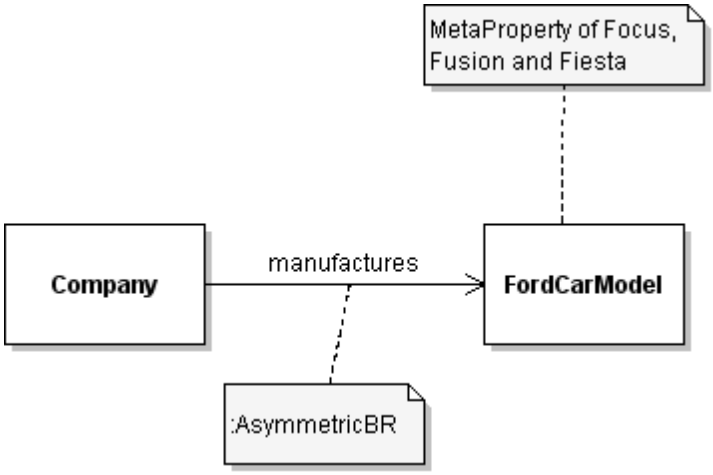

MetaProperty of Focus, Fusion and Fiesta
Company
manufactures
FordCarModel
:AsymmetricBR

FIGURE 8

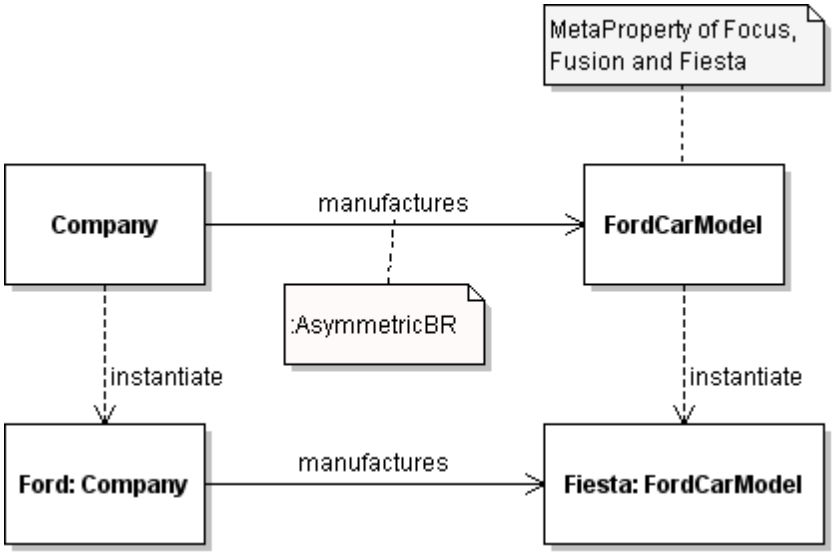

MetaProperty of Focus, Fusion and Fiesta
Company
manufactures
FordCarModel
:AsymmetricBR
instantiate
instantiate
Ford: Company
manufactures
Fiesta: FordCarModel

FIGURE 9

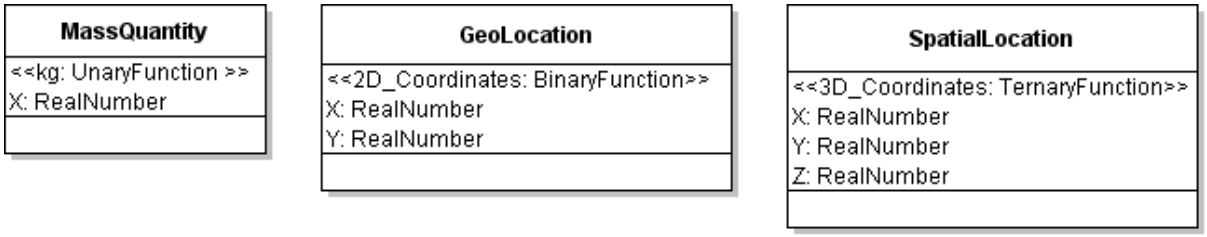

MassQuantity
<<kg: UnaryFunction >>
X: RealNumber
GeoLocation
<<2D_Coordinates: BinaryFunction>>
X: RealNumber
Y: RealNumber
SpatialLocation
<<3D_Coordinates: TernaryFunction>>
X: RealNumber
Y: RealNumber
Z: RealNumber

FIGURE 10

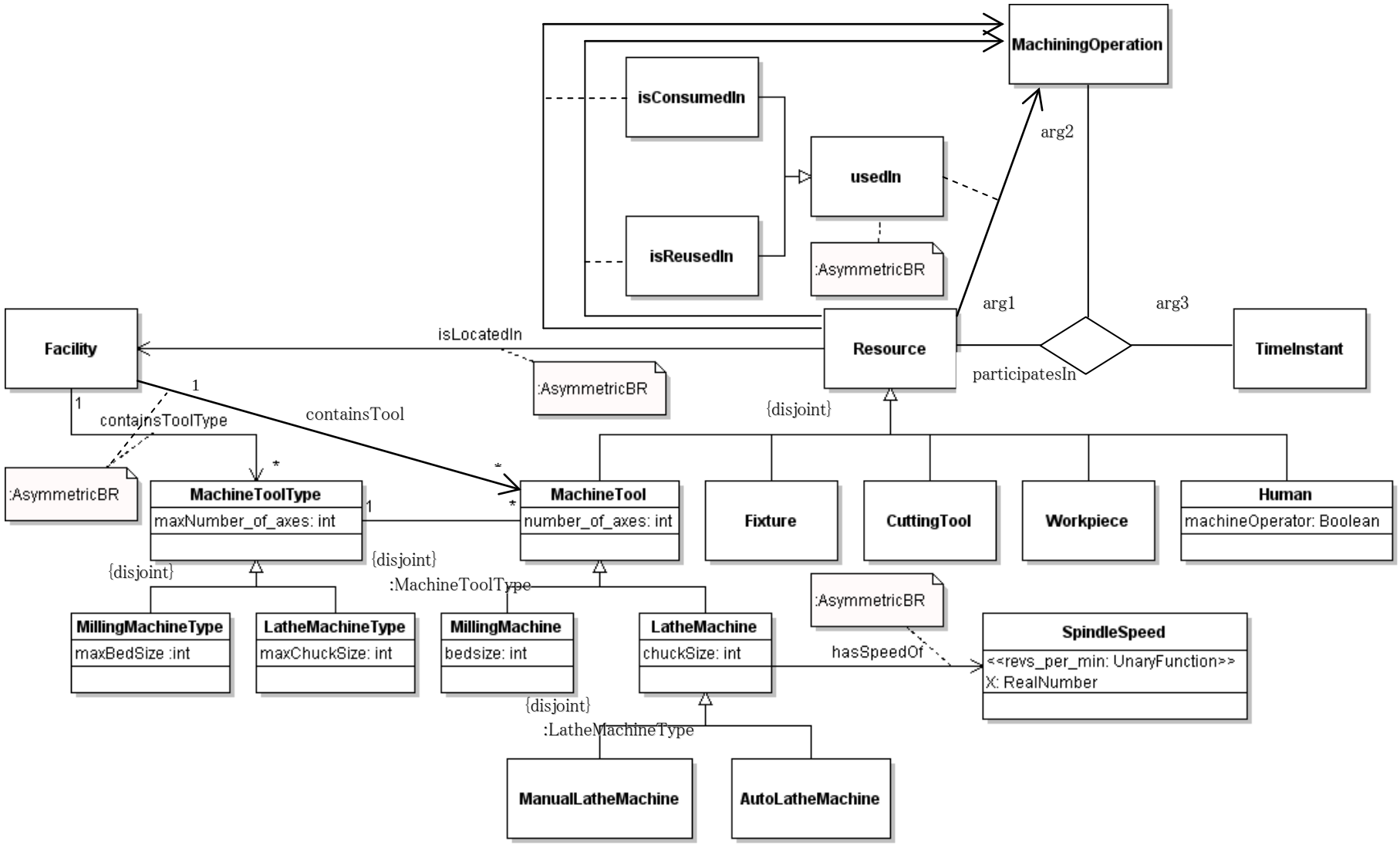

MachiningOperation
isConsumedIn
usedIn
isReusedIn
:AsymmetricBR
arg2
arg1
arg3
Facility
isLocatedIn
Resource
TimeInstant
participatesIn
:AsymmetricBR
1
containsToolType
containsTool
{disjoint}
:AsymmetricBR
MachineToolType
maxNumber_of_axes: int
MachineTool
number_of_axes: int
Fixture
CuttingTool
Workpiece
Human
machineOperator: Boolean
{disjoint}
{disjoint}
:MachineToolType
MillingMachineType
maxBedSize :int
LatheMachineType
maxChuckSize: int
MillingMachine
bedsize: int
LatheMachine
chuckSize: int
:AsymmetricBR
hasSpeedOf
SpindleSpeed
<<revs_per_min: UnaryFunction>>
X: RealNumber
{disjoint}
:LatheMachineType
ManualLatheMachine
AutoLatheMachine

FIGURE 11