**Converting an Informal Ontology into Ontolingua: Some Experiences**

Mike Uschold

AIAI-TR-192

March 1996

A slightly abridged version of this paper appears in the Proceedings of the Workshop on Ontological Engineering held in conjunction with ECAI 96 in Budapest.

*Mike Uschold*
Artificial Intelligence Applications Institute
(AIAI); The University of Edinburgh
80 South Bridge
Edinburgh EH1 1HN
Scotland
Tel: +44 (0)131 650-2732
Fax: +44 (0)131 650-6513
Email: `m.uschold@ed.ac.uk`

**Abstract**

We report our experiences of converting a carefully defined informal ontology expressed in natural language into the formal language: Ontolingua. The objectives of this paper are 1) to explore some of the nitty gritty details of formalising ontology definitions and 2) to serve as a basis for clarifying the relationship between this and other approaches to ontology construction (*e.g.* using competency questions), for the eventual aim of producing a comprehensive methodology.

We first discuss concepts in the meta-ontology, including entities, classes, instances, relationships, roles, sets and states of affairs. With respect to roles, we define a special meta-class to classify objects whose existence necessarily depends on their being in a relationship with some other entity (e.g a customer). We describe a mechanism for classifying states of affairs which can be used to restrict what can be in certain relationships (e.g pre-condition).

We then note some general issues that arise when producing formal definitions of the main terms; *e.g.* representing terms from a difference perspective, and identifying when and how new terms must be introduced. The need for new terms arises not only to fill gaps, but also to make explicit facts and logical dependencies that were only implied by the text definitions.

# 1 Introduction

In this paper, we report our experiences of the process of converting the informal version of the Enterprise Ontology [3], expressed in natural language, into the formal language, Ontolingua (OL).

As described in [6, 7], we recommend separating out the informal and formal phases of ontology production. In the case of the Enterprise Ontology, we used the informal natural language version as a specification for the formal version. In this paper, we refer to the informal version as the Specification and to its formal encoding in Ontolingua as the Code.

Our experiences of developing the Specification are described in detail in [6, 7]. Also described in [7] is a more formal approach to developing ontologies based on the idea of competency questions; this was developed as part of the TOVE project [1, 2].

One objective of this paper is to facilitate the comparison and eventual merging of this [Enterprise] approach with the competency question [TOVE] approach. This, in turn, should enable further steps to be taken towards the development of a comprehensive and cohesive methodology for building ontologies than, for example is described in [7].

Another key objective of this paper is to examine some of the nitty gritty details of the coding process. We describe some of the difficulties we had and some of our approaches to overcoming them. To our knowledge, these experiences are described here at a much finer level of granularity, than has been reported elsewhere.

This paper also serves to clarify the relationship between the Specification and the Code. Further details are available in the natural language version of the Enterprise Ontology available from the Enterprise Project [3] web pages.

On occasion, we refer to technical details regarding Ontolingua. However, they may be safely ignored by readers unfamiliar with the language, as other material does not depend on these details.

The Enterprise Ontology was was developed as part of the Enterprise Project, a collaborative effort to provide a method and a computer toolset for enterprise modelling. Both the informal and formal versions of the Enterprise Ontology are available on the world-wide web. Readers should obtain and refer to these to obtain full benefit of this paper.

**The Role of the Code** The role of the formal representation of the Enterprise Ontology is to provide a more precise specification of the meaning of the ontology than is possible in natural language.

Coding began with version 1.0 of the Specification. The analysis performed during coding identified a number of opportunities for improving the Ontology. These are reflected in the Code. It also resulted in a small number of changes, some significant, being made to the Specification resulting in the current version (1.1).

The Code is not claimed to be totally rigorous or complete. In particular, we make no claims about how or whether the axioms will be used directly by any theorem prover or automatic language translation software. Users of the Code may add further axioms for greater rigour or completeness depending on their requirements.

**Fidelity** Overall, we believe that we were successful in accurately representing the intended meaning of the terms described in the Specification. Below we discuss some of the details of the coding process and the relationship of the Code to the Specification. Differences include simple name changes, removing some terms, adding new terms and shifts in perspective for a particular concept.

**Choice of Language** The existence of the Ontolingua language (based on KIF) in general and the Ontology Editor/Server in particular has greatly facilitated the process of converting the natural language specification of the ontology into a formal language. The choice of Ontolingua as a representation language has proved highly suitable from the point of view of representational adequacy.

Within the Ontology development part of the Enterprise Project, suitability from other vantage points, (*e.g.* the software) remains untested.

**Outline** In this paper, we first describe the details of how the Meta-Ontology presented in the Specification was manifest in the Code. Of particular importance is how Roles and States of Affairs were handled. After this, we identify some of the main issues that arose during the coding process.

**A note on terminology** In general, terms defined in the Specification will be presented in upper case (*e.g.* ACTIVITY) and terms defined in the Code will be presented in italics (*e.g. Strategic-Purpose*).

# 2 Meta-Ontology

KIF, on which Ontolingua (OL) is based, gives the full expressive power of first-order logic. As such, it comes with a standard meta-ontology, namely: objects, relations, and functions. For the most part, OL provided adequate primitives to cover what was required to represent the Enterprise Meta-Ontology. There was little to be gained by formally defining things like 'ENTITY' and 'RELATIONSHIP' as described in the Specification. However, for clarity, we point out precisely what these correspond to in the Code.

## 2.1 Entities, Classes and Instances

In the Specification, to conform to common natural language usage, we intentionally blurred the distinction between a *type* of entity, and a *particular* entity of a certain type. The majority of terms defined in the Specification correspond to *types* of entities, which, in OL are unary relations called *Classes – e.g.* Person, Activity, Purpose. Particular entities of a certain type are called *Instances*, in OL.

Formally, 'ENTITY' in the Specification, (taken as a type of thing rather than a particular thing of a certain type) is equivalent to the union of the OL Frame-Ontology classes: Set and Thing.

## 2.2 Relationships, Roles and Role Classes

**Relationship** 'RELATIONSHIP', in the Specification was also deliberately ambiguous, reflecting common usage of the term in natural language. In particular, it referred both to the set of tuples constituting a relation and a single tuple. If we restrict usage to refer to the set of tuples (*i.e.* the mathematical relation), then 'RELATIONSHIP' is equivalent to a subclass of Relation@Frame-Ontology which excludes unary-relations. We found no need to define this class explicitly in OL.

**Attribute** 'ATTRIBUTE' in the Specification is roughly equivalent to a Function in OL. However, in the main, what was said to be an ATTRIBUTE in the Specification is modelled in OL as a slot on some class whose slot-cardinality is set to 1.[1]

**Role** While it seemed useful in the Specification to introduce various terms defined specifically as ROLEs, the concept of a ROLE is not directly and explicitly represented in the Code. Instead, a ROLE is implicitly represented as the semantics of an argument in a relation.

For example, a particularly important ROLE is RESOURCE, defined as the Role of an ENTITY in a RELATIONSHIP with an ACTIVITY whereby the ENTITY is or can be used or consumed during the ACTIVITY.

It is not obvious how or whether one might usefully represent this ROLE, per se, in Ontolingua. However corresponding to every ROLE, is the set of all ENTITIES that play that ROLE. For RESOURCE and other important ROLES, we formally represent this set as a special kind of class called a *Role-Class*.

We represent the RELATIONSHIP referred to in the definition of RESOURCE as a binary

[1] There is a subtle distinction here. A slot with slot-cardinality set to 1 may not explicitly be a Function in OL; rather it *corresponds* to what has the defining property of a function. In particular, it corresponds to a sub-relation (i.e. a subset of tuples) of the [independently defined] Binary-Relation used in the slot. That Binary-Relation need not be a Function.

relation called *Can-Use-Resource*, where the first argument refers to the activity, and the second to the entity. The unary relation *Resource*, represents the class of all entities (i.e. instances) that participate in this Relationship with some activity. It is defined as follows:

$$\forall E.(Resource(E) \quad \leftrightarrow \quad \exists A.(Activity(A) \wedge Can\_Use\_Resource(A, E)))$$

So, the concept of a ROLE is adequately represented in OL, but from a different perspective from that in the Specification. Rather than formalise the way an Entity participates in a Relationship, instead we formalise the set of all Entities that participates in a Relationship in that certain way.

As a matter of convenience, and formal precision, we defined *Role-Class* in OL as a meta-class, *i.e.* the class of all classes which are defined in terms of roles. Its instances are classes defined to be the set of all Entities playing a particular Role in some Relation. This idea is analogous to the notion of secondness described in [4].

A particular role class, such as *Resource*, is an instance of the [meta-]class *Role-Class*. To the extent that updates may occur which change the particular set of tuples comprising a relation, being an instance of such a class is dynamically determined. For example, an Entity may, in principle, be a Resource at one time, but not at another.

There are many other important ROLES in the Specification that give rise to a *Role-Class* in OL; a few are noted below:

*Assumption*: The *State-Of-Affairs* in an *Assumed* relationship with some *Actor*;

*Stake-Holder*: An *Actor* that *Holds-Stake-In* some *Organisational-Unit*;

*Purpose*: a *State-Of-Affairs* that is either

- in a *Hold-Purpose* relationship with some *Actor*, or
- the *Intended-Purpose* of some *Plan*.

Note that *Purpose* is an interesting example which is logically the union of two *Role Classes*. Note that strictly, *Purpose* is not a simple Role-Class as we have defined it, rather is is defined as the union of two simple role classes. We have not attempted to distinguish between simple and complex roles classes, nor therefore, have we attempted to clarify or define what other kinds of complex role classes might arise.

See appendix A for formal definitions in Ontolingua.

### 2.3 Set Classes

A situation which commonly arises is the need to represent certain sets which are not themselves naturally viewed as classes. Consider a MARKET SEGMENT; it is a subdivision or component of a market. Every market segment can itself be viewed as a market, but is distinguished by being defined explicitly as being limited to certain products, vendors, and or customers. It is reasonable, then, to represent a *Market-Segment* as a sub-class of *Market*. Its attributes are *Product-Range*, *Vendor-Range* and *Customer-Range*. What is the type/class of values of these attributes? A product range, is a set of products, a customer range is a set of customers, etc. So, we create three new classes called: *Set-of-Products*, *Set-of-Vendors* and *Set-of-Customers* and appropriately restrict the type of entity that can fill the range slots.

However, how do we represent these latter things? One way is to invent new classes (*e.g.* *Set-of-Products*) independently from the underlying class (*e.g.* *Product*). Instead, we chose to capture the fact that these are special kinds of classes; they are special in that every instance of such a class is itself a set, and furthermore, every member of such an instance set is restricted to be of a single class. For example, *Set-of-Products*, is defined as follows:

$$\begin{aligned}\forall Ps.(Set\_of\_Products(Ps) \quad \leftrightarrow \quad & set(Ps) \wedge \\ & \forall x.member(x, Ps) \rightarrow Instance\_of(x, Product))\end{aligned}$$

We define all classes defined in this manner to be instances of a meta-class *Set-Class* which is the class of all such classes. *Set-of-Products* is one of its instances.

See appendix B for formal definitions in Ontolingua. See [5] for a detailed motivation for set classes and alternative formalisation in a higher order logic.

## 2.4 State of Affairs

STATE OF AFFAIRS is defined informally as a situation. It is something that can be thought of as holding, or being true (or conversely, as not holding, or as being false). Thus, in first-order logic, any state of affairs can be represented by a syntactically valid sentence, or formula. Note that while it may be convenient to think of a state of affairs as a set of sentences (*e.g.* $\{S_1, S_2, S_3\}$), this is equivalent to a single sentence using explicit conjunction (*i.e.* $S_1 \wedge S_2 \wedge S_3$). Strictly speaking, then, to formally represent a state of affairs, is to formally specify the syntax of a first-order logic sentence. Fortunately, this and other meta-level things are already formalised in KIF, so there was no need to re-define this from scratch.

From a practical standpoint, the reason for having *State-Of-Affairs* in the Ontology is to clarify the meaning of certain terms (*e.g.* *Help-Achieve*, *Intended-Purpose*, *Pre-Condition* and *Effect*). In the Code, this is done by restricting the argument types in certain relations. However, to be *any* sentence at all is a very loose, ineffective restriction. For example, Pre-Conditions and Effects relate to activities in the domain being modelled, thus we should like to further restrict the state of affairs to be only those sentences which refer to world state conditions. For example, *Home-City(John, Edinburgh)* should be allowed, but *Relconst('Intended-Purpose')* which refers to the representation language itself, should be prohibited[2].

So, the class *State-Of-Affairs* is too general because it allows sentences to be constructed referring to *any* relation at all. We require a way to define sub-classes of *State-Of-Affairs* by restricting the set of relations that can be referred to when constructing sentences representing states of affairs.

To do this, we define a meta-level binary relation: *Restricted-Sentence* whose first argument is a sentence, and whose second argument is a set of relational constants. The relation holds if and only if:

1. the first argument is a syntactically valid first-order logic sentence;
2. all relational constants referred to in the first argument are in the set comprising the second argument.

Here, the most general case is the degenerate one, where the second restriction has no effect. Formally, $S$ is a *State-Of-Affairs* if and only if *Restricted-Sentence(S, AllRelconsts)* is true; where AllRelconsts is the set of all relational constants. Formally,

$$\forall S.(State\_Of\_Affairs(S) \quad \leftrightarrow \quad Restricted\_Sentence(S, setofall(?r, relconst(?r))))$$

The more useful cases arise when one wishes to define sub-classes of State-Of-Affairs, such as *WS-Condition*, or *Authority-Condition*. Because there are likely to be a wide variety of world state relations, it would be awkward to have to explicitly list them. It is more convenient to create a separate class of world state relation constants, (*WS_Relconst*) and use the *setofall* function. Formally,

$$\forall S.(WS\_Condition(S) \quad \leftrightarrow \quad Restricted\_Sentence(S, setofall(?r, WS\_Relconst(?r))))$$

Where, for example, *WS_Relconst('Home-City')* would be true and thus in the restricted set of relational constants.

In other cases, the restriction may be to a very small number, or a single relational constant; then it is simpler to list them directly. For example,

$$\forall S.(Authority\_Condition(S) \quad \leftrightarrow \quad Restricted\_Sentence(S, setof('Hold\_Authority')))$$

**Final remarks.** Strictly, to do a comprehensive job of formally defining State-Of-Affairs, we would have to essentially repeat what is defined in the KIF-Meta ontology, re-structuring it slightly to suit our purposes. We have chosen not to do this at this time.

[2] In KIF, *Relconst* is a unary relation representing relational constants; it is used in a bootstrapping fashion to define KIF syntax.

# 3 Producing Formal Definitions

The Meta-Ontology as described above, is the formal foundation on which the definitions of all other terms is based. In producing formal definitions of the terms in the Meta-Ontology and of all other terms, a number of issues arose giving rise to the need to change things somewhat from how they were described in the Specification. Most of the important changes fell into the following categories, which we will address in turn:

- A number of terms were not coded at all;
- Some terms were defined from a different perspective;
- Many new terms were introduced.

For example, in the Meta-Ontology, ACHIEVE, ENTITY and RELATIONSHIP fall in the first category; ROLE is defined from a different perspective (*i.e. Role-Class*); and POTENTIAL ACTOR is a new term (not found in version 1.0 of the Specification). Below we elaborate on these issues and give further examples from the main sections of the Ontology.

## 3.1 Terms not Defined

In some cases, a term referred to a concept which there was no obvious need to define, or there was no obvious way to do so in a useful manner.

For example, ACTIVITY-DECOMPOSITION is manifest in the details of how SUB-ACTIVITIES are inter-related, and other constraints that comprise an ACTIVITY SPECIFICATION. Defining something formally corresponding to an ACTIVITY DECOMPOSITION did not seem useful.

A MANAGEMENT LINK is defined to be a specific relationship between two particular ORGANISATIONAL UNITS. In the Code, we instead define the *Manages* relation which formally represents all such links as a set of tuples. Formally, MANAGEMENT LINK refers to the class of all tuples that are in the *Manages* relation; there was no need to formally define such a class.

Similarly, in version 1.0, ORGANISATIONAL STRUCTURE was defined to be "the MANAGEMENT LINKS relating a set of OUs" which strictly speaking, can be interpreted to be identical to the set of tuples comprising the *Manages* relation, and thus is also unnecessary to define.

## 3.2 Terms Viewed from a New Perspective

In some cases, the perspective from which an entirely clear and natural definition was given in the Specification, was awkward to base the formal definition on. ROLE is one such example, we have already considered. Another is AUTHORITY, which is defined as "the right of an Actor to EXECUTE an ACTIVITY SPECIFICATION". However, it was simpler to model this as a binary relation (*Hold-Authority*) denoting the fact that an ACTOR has the right to EXECUTE an ACTIVITY SPECIFICATION. There is no essential change in meaning, just of perspective. It would be possible to model the 'right' explicitly to retain the original perspective, but this was not deemed useful.

## 3.3 New Terms

There are rather more terms in the Code than in the Specification. There are three main reasons for this.

1. to fill gaps, *i.e.* things were missing in the Specification;
2. to make explicit much that which was only implied in the Specification which required teasing out;
3. to formalise logical connections that were clearly evident, but not precisely characterised in the Specification.

### 3.3.1 Filling Gaps

Examples of the first situation are SALE OFFER and ACTIVITY SPECIFICATION. The latter is a particularly important concept which was deemed to require explicit definition, so as to distinguish a set of instructions for doing something from the doing of the thing itself (*i.e.* ACTIVITY). The underlying concept was clearly evident in the original definition of PLAN (in version 1.0): "a specification of one or more ACTIVITIES for some PURPOSE". With the addition of ACTIVITY SPECIFICATION, this was changed to "an ACTIVITY SPECIFICATION with an INTENDED PURPOSE".

### 3.3.2 Making Things Explicit

An example of the second situation arises where something is defined in the Specification as 'a Role in a Relationship between an X and a Y whereby ...'. For example, Assumption is defined to be "a Role of a State Of Affairs in a Relationship with an Actor whereby the Actor takes the State Of Affairs to be true without knowing whether it is true or not". In the Specification, it is only noted that the Relationship exists but it is neither named nor defined. These Relationships are formalised as [usually binary] relations. In this case, the *Assumed* relation was defined and *Assumption* is a *Role-Class* formally defined in terms of this relation.

### 3.3.3 Formalising Logical Connections

As an example of the last situation, consider the following definitions from version 1.0 of the Specification:

**PLANNING:** an ACTIVITY whose major EFFECT is to produce a PLAN;

**STRATEGY:** a PLAN to ACHIEVE a high-level PURPOSE;

**STRATEGIC PLANNING:** an ACTIVITY whose PURPOSE is to produce a STRATEGY.

Problems with these definitions are:

- the idea of a 'major EFFECT' is undefined;
- 'high-level PURPOSE' has no meaning, though it appears to be a special kind of PURPOSE;
- STRATEGIC PLANNING is not defined in terms of PLANNING;
- the phrase 'to produce' is used in the definitions of STRATEGIC PLANNING and PLANNING, but is undefined.

To address this, we made the following alterations:

- We introduced a new term: *Strategic-Purpose* which is formally defined as a type of *Purpose*;
- *Strategic Planning* is formally defined as a type of *Planning*;
- 'to produce' is defined as a Relationship called *Actual-Output* between an *Activity* and an Entity where by the Entity is an output produced by the *Activity*;
- the idea of a 'major EFFECT' is formalised using *Intended-Purpose* which is linked with *Actual-Output* in the formal definition of *Planning*.

Most of these changes are reflected in version 1.1 of the Specification, the major exception being *Actual-Output*, which is defined only in the Code. The following definitions are as given in the Code:

*Planning*: An *Activity* whose *Intended-Purpose* is to produce a *Plan*.

*Strategic-Purpose*: A *Purpose* held by an *Actor* that is declared to be of 'strategic' importance.

*Strategy*: a *Plan* whose *Intended-Purpose* is a *Strategic-Purpose*

*Strategic-Planning*: a *Planning Activity* whose *Intended-Purpose* is to produce [an *Actual-Output* which is] a *Strategy*

Although we avoiding the use of the term 'high-level', the resulting definition of *Strategic-Purpose* has a circular aspect. The fact is, whether something is 'strategic' or not, is a fairly arbitrary declaration. It is up to users to use this is a sensible manner.

Summarising this example, by introducing two new terms: *Strategic-Purpose* and *Actual-Output* we have been able to make our definitions more precise, making various implicit connections explicit.

# 4 Summary and Conclusion

In the paper, we have reported our experiences in converting the informal version of the Enterprise Ontology expressed in natural language, into the formal language: Ontolingua. We have attempted to do so in a general way.

First, we described proposed solutions to what may be general problems occurring in the development of a wide range of ontologies. This includes how to represent a state of affairs, role concepts and sets which are not themselves classes.

Then we characterised in general terms the sorts of issues that will be faced when converting an informal ontology into a formal one. This included:

- representing terms from a difference perspective; *e.g.* roles
- when and how to introduce new terms; in particular:
  - when an important concept is missing, so as to fill a gap;
  - to make explicit that which was clearly evident but only implied in the informal ontology;
  - formalising logical connections between terms that are related where such relationships were not initially obvious from the informal ontology.

**Future Work**

Currently, the Enterprise Ontology is untested in a genuine application. Only when such tests are performed will it become clear which of our decisions were important and the extent to which they may be regarded as 'right'.

Also, we wish to clarify the relationship between the approaches to ontology development used in the Enterprise project and that used in TOVE. Both start with an informal stage and proceed to a formal one; but the Enterprise approach does not use competency questions, whereas the TOVE does. We wish to explore when and whether each approach is more or less appropriate and the extent to which they may fruitfully be merged.

**Acknowledgements**

We extend thanks to James Rice in particular and the team at Knowledge Systems Lab (KSL) Services in general for their help during the coding effort. They showed unusual commitment to our being satisfied users. Details of KSL are available on the World-Wide Web at *http://www-ksl.stanford.edu/*.

Persons who provided useful comments and feedback on various versions of the Enterprise Ontology Code include: Florence Fillion, Nicola Guarino, Ian Harrison, Martin King, Chris Menzel and Austin Tate.

# A Role Classes

```
;;; Role-Class

(define-frame Role-Class
  :own-slots
  ((Documentation "Role-Class is a meta-class.  Its instances are classes
defined to be the set of all Entities playing a particular Role in some
Relation.

<p> To the extent that updates may occur which change the particular set of
tuples comprising a relation, being an instance of such a class is dynamically
determined.")
   (Subclass-Of Class))
  :axioms
  (<=> (Role-Class ?rc)
       (Exists (?r ?n)
       (and (relation ?r)
    (natural ?n)
    (forall (?z) (<=> (instance-of ?z ?rc)
      (exists (?args)
      (and (list ?args)
   (holds ?r ?args)
   (= (nth ?args ?n) ?z))))))))
  :issues
  ("Strictly, this definition is not quite right.  It does not cover the case
where a Role-Class may be defined in terms of more than one Role.
An entity is an instance of such a Role-Class if it is in one of two or more
Roles in one or more Relations.
<p>
It also does not cover the case when"
   "An odd case is when a Class is defined as the union of a Role-Class and
a non Role-Class; is this a Role-Class or not?"
   "?"
   "Why not have Role-Class a super-class of various Role-Classes such as
Purpose and Resource?
```

```
<p> I can think of no good reasons for it to be one way or the other, thus
choice was mainly arbitrary.  The other way would entail use of
a different name for the superclass so
as to suggest the right meaning (e.g. Role-Player, or Role-Playing-Entity)."
   (:Example "Given Classes Person and Document
 are defined, we show below how a binary
relation Read-Document and the Role-Class Reader might be defined.
<pre>
;;; Read-Document

(define-relation Read-Document (?person ?document)
  \"A Relationship between an Person and a Document whereby the Person
reads the Document\"
  :def
  (and (Person ?person) (Document ?document)) )

;;; Reader

(define-frame Reader
  :own-slots
  ((Documentation \"a Person in an Read-Document Relationship with
 some Document\")
  (Instance-Of Role-Class) (Subclass-Of Qua-Entity))
  :axioms
  (<=> (Reader ?person)
       (and (Person ?person)
    (exists (?document) (Read-Document ?person ?document)))) )

</pre>

Note that the axiom above is equivalent to how it might be expressed using
the general form given for defining a Role-Class:

<pre>

(<=> (Reader ?person)
     (exists (?args)
     (and (list ?args)
  (holds ?Read-Document ?args)
  (= (nth ?args 1) ?person))))

</pre>

Note also that the wording used to define role-classes is: <br>
'The Person in a R relationship.'  rather than <br>
'The Role of a Person that ... in a R relationship.' which was used in
the natural language version.  This reflects that fact that we do not
explicitly define what are called roles in the NL version; rather we
define Role-Classes in terms of such roles which are [only] implicit in
the definition of relations.")) )
```

```
;;; Qua-Entity

 (define-frame Qua-Entity
   :own-slots
   ((Documentation
    "An EO-Entity that is defined in terms of the role it plays in one or
more Relationships.
<UL>
<li> Qua-Entity is the most general Role-Class
<LI> Every instance of Role-Class is a subclass of Qua-Entity.
</UL>")
   (Instance-Of Role-Class) (SubClass-Of EO-Entity))
   :axioms
   (<=> (Qua-Entity ?x)
        (Exists (?rc)
                (and (Instance-Of ?rc Role-Class)
                     (Instance-Of ?x ?rc))))
   :issues
   ("This is an abstract class provided mainly for convenience, so it is easy
to see what all the Role-Classes are."
    "It is up to Ontology developers, users and maintainers to make
sure each Role-Class is declared to be a subclass of Qua-Entity or of
Actor, which is itself a subclass of Qua-Entity."
   ) )
```

# B Set Classes

```
;;; Set-Class

(define-frame Set-Class
  :own-slots
  ((Documentation "Set-Class is a meta-Class.  Its instances are special
kinds of classes,  all of whose instances
are themselves sets (not Classes) such that every member of such a set
is specified to be a member of a certain Class.")
   (Subclass-Of Class))
  :axioms
  (<=>
   (Set-Class ?set-of-things)
   (Exists (?thing)
   (and (Class ?thing)
(forall (?things)
(<=> (instance-of ?things ?set-of-things)
     (and (set ?things)
  (forall (?x)
  (=> (member ?x ?things)
      (instance-of ?x ?thing))))))))

  :issues
```

```
  ("The Class which forms the basis for what sets are instances of the
    Set-Class is called the `base class'."
   "The denotation of a Set-Class is the power set of the denotation of its
    base class."
   "In a higher order logic, the set classes may be formed by
a type constructor function, which take as input the base class and returns
the corresponding set class. <p>
Here, we use a naming convention to indicate this; the names of
all set classes are prefixed with the text `Set-of', as in Set-of-Customers"
  (:Example   "<pre>
(<=>   (Set-of-Customers ?customers)
       (and (set ?customers)
    (forall (?x)
    (=> (member ?x ?customers)
(instance-of ?x Customer)))))
</pre>")) )

;;; EO-Set

 (define-frame EO-Set
   :own-slots
   ((Documentation
    "The most general Set-Class in the Enterprise Ontology.
Every instance of Set-Class is a subclass of EO-Set.")
   (Instance-Of Set-Class) (SubClass-Of EO-Entity Set))
   :axioms
   (<=> (EO-Set ?x)
        (Exists (?sc)
                (and (Instance-Of ?sc Set-Class)
                     (Instance-Of ?x ?sc))))
   :issues
   ("This is an abstract class provided mainly for convenience, so it is
easy to see what all the Set-Classes are."
    "It is up to Ontology developers, users and maintainers to make
sure each instance of Set-Class is declared to be a subclass of EO-Set."))



;;; Set-of-Products

(define-class Set-of-Products
  (?products)
  "A Set-Class all of whose instances are sets whose members are all of
Class Product."
  :iff-def
  (and (EO-Set ?products)  ;;;  i.e. Set-of-Products is a subclass of EO-Set
       (and (Set ?products)
    (forall (?x)
    (=> (Member ?x ?products)
(Instance-Of ?x Product)))))
  :issues
```

```
("This is a special Set-Class") )
```

Note that *EO-Set* is the most general instance of the meta-class *Set-Class*. All sub-classes of EO-Set are thus, also instances of this meta-class.

# References


[1] M. Gruninger and M.S. Fox. The logic of enterprise modelling. In J. Brown and D. O'Sullivan, editors, *Reengineering the Enterprise*, pages 83–98. Chapman and Hall, 1995.

[2] M. Gruninger and M.S. Fox. Methodology for the design and evaluation of ontologies. In *Workshop on Basic Ontological Issues in Knowledge Sharing*. International Joint Conference on Artificial Intelligence, 1995.

[3] Fraser. J., A. Tate, and M. Uschold. The enterprise toolset - an open enterprise architecture. In *The Impact of Ontologies on Reuse, Interoperability and Distributed Processing*, pages 42–50. Unicom Seminars, London, 1995. Further information about the Enterprise Project and Ontology is available on the World Wide Web from: *http://www.aiai.ed.ac.uk/~entprise/enterprise/*.

[4] J. Sowa. Top-level ontological categories. *International Journal of Human-Computer Studies*, 43(5/6):669–686, 1995.

[5] M. Uschold. The use of the typed lambda calculus for guiding naive users in the representation and acquisition of part-whole knowledge. *Data and Knowledge Engineering*, 1996. to appear in special issue on the part-whole relationship.

[6] M. Uschold and M. King. Towards a methodology for building ontologies. In *Workshop on Basic Ontological Issues in Knowledge Sharing*. International Joint Conference on Artificial Intelligence, 1995. Also available as AIAI-TR-183 from AIAI, The University of Edinburgh.

[7] M. Uschold and Gruninger M. Ontologies: Principles, methods and applications. *Knowledge Engineering Review*, 11(2), 1996.