CHAPTER 5
ONTOLOGY-BASED METADATA DICTIONARY COMPONENTS

This chapter furnishes some formal definitions of ontology-based metadata dictionary components that encompass the virtual concepts, relationships, physical source configurations, and construction rules. These formal definitions are represented by object-oriented and set theory that can be used as a basis for the construction of metadata dictionary for any real world implementation. These formal definitions will be used as a basis for designing the XML-based metadata dictionary in the next chapter to ensure the model realization.

5.1 Ontology-based Metadata Dictionary Components

A general notation for formal semantics of ontology-based metadata dictionary and its components are introduced based on object-oriented and set theory. The domain ontology, D, that represents the ontology-based metadata dictionary is defined as a quadruple

$$D = \langle C, R, S, \hat{C} \rangle, \text{ where}$$

- $C = \{vc_i \mid \forall\, I=1 \ldots n\}$ is a finite set of all unique virtual concepts in D;
- R denotes the relationships between concepts or instances in D encompassing associative, IS-A, IS-PART-OF, and instantiated relationships;
- S denotes physical source configurations; and
- $\hat{C}$ denotes the construction rules established to govern the operations over C and R, thus ensuring the correctness and consistency of the domain ontology.

Each domain ontology component is defined as follows:

5.1.1 Virtual Concept

A virtual concept $vc_k \in C$ is defined as a pair

$$vc_k = \langle n(vc_k), P(vc_k) \rangle, \text{ where}$$

- $n(vc_k)$ is the virtual concept name which is unique within D; and
- $P(vc_k) = \{vp_{k1}, vp_{k2}, \ldots, vp_{km}\}$ is a finite set of unique virtual properties of the virtual concept vc_k.

These virtual properties can be classified into three groups, namely, *object identifiers*, *object identifier references*, and *ordinary properties*, depending on their property values. The set of object identifiers or keys of vc_k is denoted as $OID(vc_k) \subseteq P(vc_k)$ in which each instance of vc_k cannot have the same value properties in $OID(vc_k)$. $REF(vc_k) \subseteq P(vc_k)$ denotes the set of object references or foreign keys of vc_k that establishes the relationships between instances of different concepts. $ORD(vc_k) \subseteq P(vc_k)$ denotes the set of ordinary properties whose values are atomic values (e.g., integer, string).

Given a virtual concept vc_k, the two sets $OID(vc_k)$ and $REF(vc_k)$ can be joint sets, that is, $OID(vc_k) \cap REF(vc_k) \neq \phi$ (e.g., the set $OID(vc_k) = \{vp_{k1}, vp_{k2}\}$ and the set $REF(vc_k) = \{vp_{k2}\}$), or disjoint sets, that is, $OID(vc_k) \cap REF(vc_k) = \phi$ (e.g., the set $OID(vc_k) = \{vp_{k1}\}$ and the set $REF(vc_k) = \{vp_{k2}\}$). The two sets $OID(vc_k)$ and $REF(vc_k)$ and the set $ORD(vc_k)$ are disjoint sets, that is, $(OID(vc_k) \cup REF(vc_k)) \cap ORD(vc_k) = \phi$ (e.g., the set $OID(vc_k) = \{vp_{k1}, vp_{k2}\}$, the set $REF(vc_k) = \{vp_{k2}\}$, and the set $ORD(vc_k) = \{vp_{k3}\}$).

To solve data type and scaling conflicts, the virtual property $vp_{kc} \in OID(vc_k)$ or $vp_{kd} \in ORD(vc_k)$ is designed as a class consisting of two domain properties, namely, predefined type and scaling domains. Denote PD and SD as the sets of values of the predefined type domain and scaling domain properties, respectively, that is, PD = {"integer", "string", "float", "decimal", "char"} and SD = {"NULL", "kilogram", "pound", "US$", "AUS$"}.

A virtual property $vp_{kc} \in P(vc_k)$ is defined as a pair

$$vp_{kc} = \langle n(vp_{kc}), d \rangle, \text{ where}$$

- $n(vp_{kc})$ is the virtual property name which is unique within vc_k; and
- d is the domain variable defined as follows:

$$d = \begin{cases} \text{NULL} & \text{, if } vp_{kc} \in REF(vc_k) \\ \langle d_1, d_2 \rangle & \text{, if } vp_{kc} \in OID(vc_k) \text{ or } vp_{kd} \in ORD(vc_k) \end{cases}$$

where d_1 is the predefined type domain property denoted as a pair $<n(d_1), v(d_1)>$ in which $n(d_1)$ is the name of d_1 and $v(d_1) \in PD$ (e.g., $v(d_1)$ = "integer"), and d_2 is the scaling domain property denoted as a pair $<n(d_2), v(d_2)>$ in which $n(d_2)$ is the name of d_2 and $v(d_2) \in SD$ (e.g., $v(d_2)$ = "kilogram").

To solve naming conflicts, the virtual property $vp_{kc} \in OID(vc_k)$ or $vp_{kd} \in ORD(vc_k)$ can also constitute physical instances, that is, instances representing the synonymous physical property names of physical concepts.

Let p_{kct} be a physical instance of vp_{kc} and be defined as a pair

$$p_{kct} = <n(p_{kct}), P(p_{kct})>, \text{ where}$$

- $n(p_{kct})$ is the physical instance name (or physical property name) which is unique within vp_{kc}; and
- $P(p_{kct}) = \{I_1, I_2, \ldots, I_p\}$ is a finite set of unique physical information properties of p_{kct} that describes related physical information to p_{kct}. The definition of these properties are defined as follows:
  - I_1 is a physical data type property denoted as a pair $<n(I_1), v(I_1)>$, where $n(I_1)$ is the name of I_1, and $v(I_1) \in PD$;
  - I_2 is a physical unit type property denoted as a pair $<n(I_2), v(I_2)>$, where $n(I_2)$ is the name of I_2, and $v(I_2) \in SD$;
  - I_3 is a physical concept property denoted as a pair $<n(I_3), v(I_3)>$, where $n(I_3)$ is the name of I_3, and $v(I_3)$ is an object identifier reference for the physical concept name of the physical source configuration;
  - I_4 is a physical source property denoted as a pair $<n(I_4), v(I_4)>$, where $n(I_4)$ is the name of I_4, and $v(I_4)$ is an object identifier reference for the physical source name of the physical source configuration; and
  - I_5 is a corresponding virtual concept denoted as a pair $<n(I_5), v(I_5)>$, where $n(I_5)$ is the name of I_5, and $v(I_5)$ is an object identifier reference for the corresponding virtual concept name.

5.1.2 Relationships

The relationships in domain ontology consist of four types, namely, associative, IS-A, IS-PART-OF, and instantiate relationships.

(1) ***Associative relationship***. An associative relationship enumerates the connectivity among instances of concepts. This relationship is classified into three categories as follows:

- *1:1 relationship*, denoted by $R_{11}(vc_i, vc_j)$, is a relationship that associates an instance of concept vc_i with at most one instance of concept vc_j, and vice versa;

- *1:N relationship*, denoted by $R_{1N}(vc_i, vc_j)$, is a relationship that associates an instance of concept vc_i with many instances of concept vc_j, but only one instance of concept vc_j can be associated with at most one instance of concept vc_i; and

- *N:M relationship*, denoted by $R_{NM}(vc_i, vc_j)$, is a relationship that associates many instances of concept vc_i with many instances of concept vc_j; and vice versa.

(2) ***IS-A relationship.*** An IS-A relationship describes a specialization relationship among concepts that establishes a subsumption hierarchy, whereby a general concept (or superconcept) subsumes more specific concepts (or subconcepts). The subsumption hierarchy is illustrated in Figure 5.1, where the concept `Staff` is a general concept or superconcept, and the concept `Administrator`, `Instructor` and `Official` are specific concepts or subconcepts of the hierarchy. The general concept subsumes more specific concepts and the specific concepts inherit the information from the general concept.

The subsumption hierarchy is used to store information at the level of generality and automatically provide this information to the lower level of specific concepts through the inheritance mechanisms. This helps in reducing the size of information need to be stored at every level of the hierarchy and also reducing data inconsistencies in the update operation.

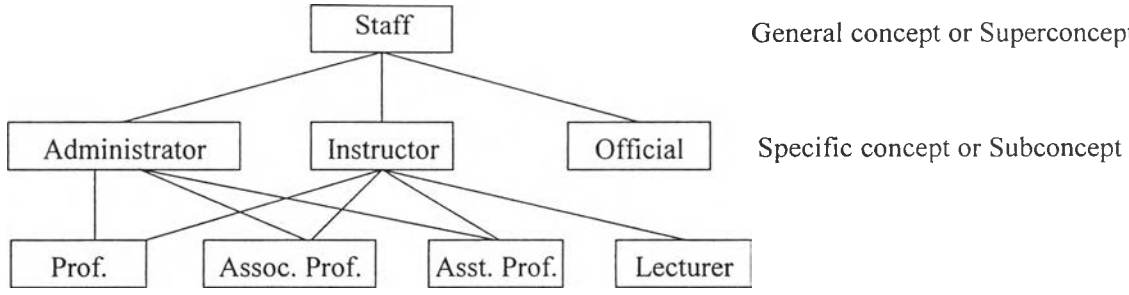

Figure 5.1 The subsumption hierarchy of the ontology.

This hierarchy also supports multiple inheritance, that is, each subconcept is allowed to have multiple superconcepts. For example, the subconcept `Professor` is not only an `Administrator`, but also an `Instructor`.

More formally, if the set of instances of vc_k is a subset of the set of instances vc_h, this can say that vc_h subsumes vc_k. In other words, vc_k is a vc_h, denoted $IS\text{-}A(vc_k, vc_h)$. A vc_h is called a superconcept of vc_k, denoted $Superconcept(vc_h, vc_k)$, and a vc_k is called a subconcept of vc_h, denoted $Subconcept(vc_k, vc_h)$. Notationally, this relationship can be defined as

$$IS\text{-}A(vc_k, vc_h) \Rightarrow Superconcept(vc_h, vc_k) \wedge Subconcept(vc_k, vc_h)$$

In inheritance context, a subconcept vc_k can define its own properties and inherit the common properties from its superconcept vc_h. In other words, the set of virtual properties of vc_h is a subset of the set of virtual properties of vc_k as follows:

$$IS\text{-}A(vc_k, vc_h) \Rightarrow P(vc_h) \subseteq P(vc_k)$$

This implies that the relationships that associate the superconcept vc_h and other concepts can also propagate to its subconcept vc_k.

(3) *IS-PART-OF relationship*. An IS-PART-OF relationship denotes one instance of an aggregate (or assembly) concept comprising of a set of component instances. The example in Figure 5.2 illustrates a car to be an aggregate concept consisting of three components, that is, the *engine*, *body*, and *transmission*. A car is related to the component *engine*, *body*, and *transmission* through the IS-PART-OF relationship. Each component has its

component instance with its own associated properties and functions. Each component instance belongs exclusively to one instance of an aggregate concept.

More formally, if vc_k is a part of vc_h, denoted *IS-PART-OF*(vc_k, vc_h), vc_h is called an aggregate concept of vc_k, denoted *Aggregate_concept*(vc_h, vc_k), and vc_k is called a component concept of vc_h, denoted *Component_concept*(vc_k, vc_h). Notationally, this relationship can be defined as

$$IS\text{-}PART\text{-}OF(vc_k, vc_h) \Rightarrow Aggregate\_concept(vc_h, vc_k) \wedge Component\_concept(vc_k, vc_h)$$

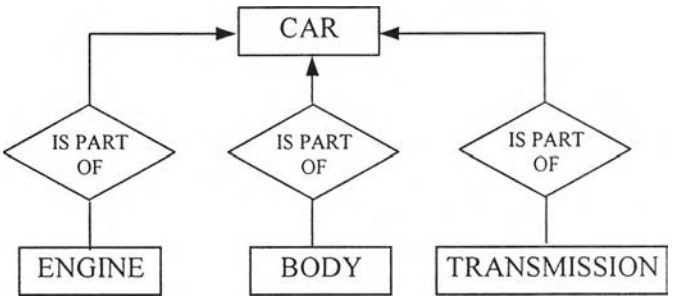

Figure 5.2 The IS-PART-OF relationship.

(4) *Instantiate relationship*. An instantiate relationship, denoted *InstanceOf*(p_{kct},vp_{kc}), is a relationship between vp_{kc} and its physical instance p_{kct}. This relationship acts as a bridge to map ontology at the conceptual level to ontology at the physical level. The instantiate relationship can be used to verify synonymous (or equivalent) physical instances in which p_{111} and p_{112} are synonymous, and write $p_{111} \sim p_{112}$, if and only if both p_{111} and p_{112} are physical instances of the same class property vp_{11}, that is,

$$(p_{111} \sim p_{112}) \Leftrightarrow InstanceOf(p_{111}, vp_{11}) \wedge InstanceOf(p_{112}, vp_{11})$$

In other words, if this relationship is considered as a tree, if $p_{111} \sim p_{112}$, both p_{111} and p_{112} are children of the same parent node vp_{11} as follows:

$$(p_{111} \sim p_{112}) \Leftrightarrow ChildOf(p_{111}, vp_{11}) \wedge ChildOf(p_{112}, vp_{11})$$

5.1.3 Physical Source Configurations

The physical source configurations at the physical level of the ontology are formally defined as follows:

Let $S = \{S_i \mid i = 1 \ldots n\}$ be a finite set of n physical information sources within D.

A physical source, denoted $S_k \in S$, is defined as a pair

$\boldsymbol{S_k = <n(S_k), P(S_k)>}$, where

- $n(S_k)$ is the name of a physical source which is unique within D, and
- $P(S_k) = \{pc_{k1}, pc_{k2}, \ldots, pc_{km}\}$ is a finite set of unique properties of S_k, that is, properties for storing the physical concept names (or entity names) in S_k.

A physical concept $pc_{kc} \in P(S_k)$ is defined as a pair

$\boldsymbol{pc_{kc} = <n(pc_{kc}), P(pc_{kc})>}$, where

- $n(pc_{kc})$ is the name of pc_{kc} which is unique within S_k, and
- $P(pc_{kc}) = \{c_j \mid j = 1 \ldots m\}$ is a finite set of unique physical configuration properties of pc_{kc}, that is, properties for storing the associated physical configurations of each pc_{kc} (e.g., physical data model, permission, owner). A property $c_k \in c_j$ is defined as a pair $<n(c_k), cnf>$, where $n(c_k)$ is the name of c_k, and cnf is the value of physical configuration property which is an atomic string value.