A.V.V.M SRI PUSHPAM COLLEGE (AUTONOMOUS), POONDI – 613503, THANJAVUR (DIST)

Department of Computer Science

EDUCATION ONTOLOGY DEVELOPMENT

User Guide for Protégé Tool

Education Ontology Development

Definition of Ontology

In 1993, Gruber originally defined the notion of ontology as an "explicit specification of a conceptualization".

Concept of Ontology

Ontology is the branch of philosophy that studies concepts such as existence, being, becoming, and reality.

Importance of the study of Ontology in Education

The ontology or an appropriate version of it can be used to guide students to understand the organization of their own learning and to self- assess their own progress. The ontologies are created by sets of people with expertise in content, teaching, psychology, and measurement.

PROTÉGÉ TOOL

- Protégé is an ontology and knowledge base editor produced by Stanford University.
- Protégé is a tool that enables the construction of domain ontologies, customized data entry forms to enter data.
- Protégé allows the definition of classes, class hierarchies, variables, variable-value restrictions, and the relationships between classes and the properties of these relationships.
- Protégé comes with visualization packages such as OntoViz; all of these help the user visualize ontologies with the help of diagrams.

CASE STUDY 1: TEACHING ONTOLOGY WITH C PROGRAMMING LANGUAGE

This case study demonstrates how to create ontology for the domain of "C *Programming Language*".

C is a robust language whose rich set of built-in functions and operators can be used to write any complex programs. Programs written in C are efficient, fast and highly portable. Its strength lies in its built-in functions. It is well-suited for structured programming. It has the ability to extend itself. The specific characteristics are low-level access to computer memory by converting machine addresses to typed pointers, functions, structures, unions, I/O string manipulation, files and mathematical functions. Figure describes this scenario through a class diagram.









Scenario of C programming language through a class diagram.

Identifying the Activities to Build C Teaching Ontology

To build a C teaching ontology, the following activities are required:

- Activity 1: Identification and Creation of Classes and Subclasses
- Activity 2: Identification and Creation of Object Properties
- Activity 3: Identification and Creation of Datatype Properties
- Activity 4: Defining Disjoint Classes
- Activity 5: Specify Property Characteristics
- Activity 6: Assign Domain and Range
- Activity 7: Identify and Create Instance or Individual

Activity 1: Identification and creation of classes and subclasses Identified classes and subclasses are shown in Exhibit 1.1.

C_Tokens
Constants
Datatypes
Variables
Operators
Keywords
Control_Statements
Branching
Looping
Arrays
Single_Dimensional
Multi_Dimensional
Functions
Pre_Defined_Function
User_Defined_Function
Pointers
Arrays_Within_pointer
Functions_within_pointer
Structure_within_Pointer
File_within_pointer
Files
Input_file
Output_file
Structure_and_Union
Array_within_structure

Exhibit 1.1 Identified classes and subclasses.

Activity 2: Identification and creation of object properties

Identified object properties are as follows:

□ hasPart

□ isPartOf

□ hasType

□ isTypeOf

has Array

□ isArrayOf

They are shown in Exhibit 1.2.



Exhibit 1.2 Identified object properties.

Activity 3: Identification and creation of datatype properties

Identified datatype properties are as follows:

- □ hasSubscript
- □ isSubscriptOf

They are shown in Exhibit 1.3.



Exhibit 1.3 Identified datatype properties.

Activity 4: Defining disjoint classes

In the class definition, the classes "*pointers*" and "*Structures and Unions*" should be defined as *Disjoint Classes* and is shown in Figure 1.21.

Activity 5: Specify property characteristics

Ontology permits the meaning of properties to be improved through the use of property characteristics. Following are the different property characteristics:

□ Functional

- □ Inverse Functional
- □ Transitive
- □ Symmetric
- □ Asymmetric
- □ Reflexive
- □ Irreflexive

Functional: The *"hasArray"* property should be assigned *functional* characteristic. It is shown in Figure 1.22.

Description: I Structures-and-Unions	1808
Sub Class Of (+)	
SubClass Of (Anonymous Ancestor)	
Members 🕀	
Target for Key 🔶	
	0000
Disjoint Union Of 🕀	
Annotations Usage	
Usage: I.Structures-and-Unions	
Found 3 uses of I.Structures-and-Unions	
H.Pointers H.Pointers DisjointWith I.Structures-and-Unions	
O I.Structures-and-Unions O H.Pointers DisjointWith I.Structures-and-Unions O Class: I.Structures-and-Unions	

Figure 1.21 Disjoint Classes—Pointers, Structures and Unions.

Object property hears dry head array The Go CbjectProperty Image: Start array Image: Start array	Anestations Deage Anestations Backgray TV_p1 hasArray TV_s0 hasArays TV_s0 hasArays		0 = ≠0
	hasArrayed	Description: Nadarne	THE R
	Functional Werse functional Transitive Symmetric Asymmetric Reflexive	Exercise (INFORMATION)	• • • • •

Figure 1.22 Functional characteristic—hasArray.

Inverse functional:The *"hasPart"* property should be assigned *Inverse functional* characteristic. It is shown in Figure 1.23.

Object property hierarchy: hasPart DERD	Annotations Usage		
t o X	Annotations: hasPart		
	Annotations		@80
■ isArrayOf ■ isPartOf ■ isTypeOf	TV_sg hasParts		080
	TV_vbg hasParted		080
			-
	Characteristics: hasPart	Description: hasPart	
	Functional	Equivalent To	•
	✓ Inverse functional		
	Transitive	SubProperty Of 🕞	
	Symmetric	Inverse Of 🕀	-
	Asymmetric	■ isPartOf	0000
	Reflexive		
	rreflexive	TwoDimoncionalArray	0000-

Figure 1.23 Inverse functional characteristic—hasPart.

Transitive: The *"hasType"* property should be assigned *Transitive* characteristic. It is shown in Figure 1.24.

Object property hierarchy: hasType	Annotations Usage		
12 C X	Annotations: hasType		0800
BopObjectProperty BasArray BasPart BasPart	Annotations 🕞 TV_pI hasType		080
■ isArrayOf ■ isPartOf ■ isTypeOf	TV_sg hasTypes		080
	TV_vbg hasTyped		080
			-
	Characteristics: hasType	Description: hasType	
	Functional	Equivalent To	•
	Inverse functional Transitive	SubProperty Of	
	Symmetric	hverse of	0000
	Reflexive Irreflexive	Domains (intersection)	0000

Figure 1.24 Transitive characteristic—hasType.

Symmetric: The "*isArrayOf*" property should be assigned *Symmetric* characteristic. It is shown in Figure 1.25.

Asymmetric: The *"isPartOf"* property should be assigned *Asymmetric* characteristic. It is shown in Figure 1.26.

Reflective: The "is*TypeOf*" property should be assigned *Reflective* characteristic. It is shown in Figure 1.27.

Object property hierarchy: isArrayOf IEBBO	Annotations Usage Annotations: isArrayOf		0883
	Amotations		000
isArrayOf isPartOf isPortOf	TV_sg isArrayOfs		080
	TV_vbg isArrayOfed		080
			-
	Characteristics: isArrayOf IDEMIC Functional	Description: isArrayOf Equivalent To	
	Inverse functional Transitive	SubProperty Of 💮	
	Symmetric	Inverse Of	
	Asymmetric Reflexive	hasArray	7880
	reflexive •	Domains (intersection)	0000-

Figure 1.25 Symmetric characteristic—isArrayOf.

Object property hierarchy: isPartOf 00BBG	Annotations Usage Annotations isPartOf		
BosPart basPart basPart	Annotations (+) TV_pI isPartOf		000
sArrayOf sPartOf stypeOf	TV_sg isPartOfs		000
	TV_vbg isPart0fed		080
			-
	Characteristics: isPartOf	Description: isPartOf	0880
	Functional	Equivalent To 🕀	•
	Inverse functional Transitive	SubProperty OF	
	Symmetric	Inverse Of	0000
	Reflexive	Domains (retersection)	0000-

Figure 1.26 Asymmetric characteristic—isPartOf.

Object property hierarchy: isTypeOf DERO	Annotations Usage		
14 O X	Annotations: isTypeOf		0800
topObjectProperty hasArray hasPart hasPart	Annotations () TV_p1 isTypeOf		080
■ isis ype ■ isiarrayOf ■ isipartOf ■ isiypeOf	TV_sg isTypeOfs		000
	TV_vbg isTypeOfed		080
			•
	Characteristics: isTypeOf 0880	Description: isTypeOf	0880
	Functional	Equivalent To 💮	▲
	Inverse functional		200
	Transitive	SubProperty Of	
	Symmetric	Inverse Of 🕀	
	Asymmetric	■ hasType	0080
	Reflexive		
	reflexive T	Compiler control directiver	

Figure 1.27 Reflective characteristic—isTypeOf.

Activity 6: Assign domain and range

Properties may have specified *domain* and *range*. Properties link individuals to individuals (from the *domain* and the *range*). For example, in this domain ontology, the property *isPartOf* would probably link individuals belonging to the class of *OneDimensionalArrays* to individuals belonging to the class of *TwoDimensionalArrays*. In this case, the domain of the property *'isPartOf '* is *OneDimensionalArrays* (Figure 1.28) and the range is TwoDimensionalArrays (Figure 1.29).

Description: isPartOf	
Equivalent To	
SubProperty Of 🕀	
Inverse Of 🕀	
hasPart	?@ ×0
Domains (intersection)	
OneDimensionalArrays	?@ ×0
Ranges (intersection)	
Disjoint With 💬	
SuperProperty Of (Chain)	

Figure 1.28 Property domain view—OneDimensionalArrays.

Description: isPartOf	
Equivalent To 🛨	
SubProperty Of 🛨	
Inverse Of 🛨	
■ hasPart	?@×0
Domains (intersection) 🕀	
Ranges (intersection)	
TwoDimensionalArrays	?@×0
Disjoirt With 🕀	
SuperProperty Of (Chain)	

Figure 1.29 Property range view—TwoDimensionalArrays.

Activity 7: Identify and create instance or individual

Identified individuals or instances are shown in Table 1.7.

Table 1.7 Identified Individuals

Classes	Instances
Constants	PI constant
Data types	Integer Data type, Float Data type, Character Data type
Logical Operators	AND Operators, OR Operators, Not Operators
Structure	Defining a structure Initializing a structure
Pointers	Defining a pointer Initializing a pointer
Arrays	Defining an Array Initializing an Array