Using Ontologies to Index Conceptual Structures for Tendering Automation

Ahmad Kayed **Robert M. Colomb**

School of Information Technology and Electric Engineering
University of Queensland,
Brisbane QLD 4072, Australia,
Email: {*kayed,colomb*}@csee.uq.edu.au

Abstract

Using natural language to model the tendering makes any process associated with tendering automation extremely difficult. Conceptual Graph is a well-known mechanism for knowledge representation. We implemented our ontologies using CGs. In tendering domain, we define two ontologies: The Tendering Structure and the Abstract Domain Ontology. In this paper we survey the indexing and retrieving techniques in CG literatures. Then we build a slight modification of these techniques to build our own indexing technique using these ontologies. Using this technique enables us to define different type of matchmaking. Also to define finding policies to direct software agents in retrieving process.

Keywords: Ontology, Conceptual Structures, Indexing, Matching, Software Agents.

1 Introduction

Conceptual Graphs (CGs) are method of knowledge representation developed by Sowa [Sowa, 1984] based on Charles Peirce's Existential Graphs and semantic networks of artificial intelligence [Sowa, 1995]. Besides Peirce's [1] primitives, CGs provide means of representing *case relations, generalized quantifiers, indexicals* , and other aspects of natural languages. For example, figure 1 shows a CG for the sentence *a cat is on a mat.*

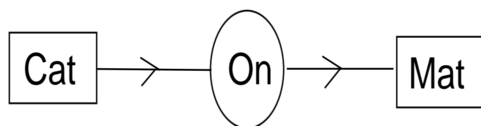

Figure 1: CG for " a cat is on a mat" [Sowa, 1999b]

In a conceptual graph, the boxes are called *concepts*, and the circles are called *conceptual relations.* There is a *formula operator* which translates CGs to formulas in predicate calculus. It maps circles to predicates with each arc as one argument, and it maps concept nodes to typed variables, where the type label inside each concept box designates the type. If no other quantifier is specified inside a concept box, the default quantifier for the variable is the existential $\exists$. The CG in figure 1 could be written as $(\exists x : Cat)(\exists y : Mat)on(x, y)$.

Copyright (c) 2001, Australian Computer Society, Inc. This paper appeared at the Thirteenth Australasian Database Conference (ADC2002), Melbourne, Australia. Conferences in Research and Practice in Information Technology, Vol. 5. Xiaofang Zhou, Ed. Reproduction for academic, not-for profit purposes permitted provided this text is included.

[1] Peirce's primitives are Existence, Co-reference, Relation, Conjunction, and Negation.

A CG g is a bipartite graph that has two kinds of nodes called *concepts* and *conceptual relations* with following properties:

- Every arc a of g must *link* a conceptual relation r in g to a concept c in g. the arc a is said to *belong* to the relation r; it is said to be *attached* to the concept c, but it does not belong to c.
- the conceptual graph g may have concepts that are not linked to any conceptual relation; but every arc that belongs to any conceptual relation in g must be attached to exactly one concept in g.
- Three kinds of CGs are given with distinguished names:
  1. The *blank* is an empty CG with no concepts, conceptual relations, or arcs.
  2. A *singleton* is a CG that consists of a single concept, but not conceptual relations or arc.
  3. A star is CG that consists of single relation and the concepts that are attached to its arcs.

According to Sowa [Sowa, 1984], CGs have a direct mapping to and from natural language and a graphic notation designed for human readability. Conceptual graphs have all the expressive power of logic but are more intuitive and readable. Many popular graphic notations and structures ranging from type hierarchies to entity-relationship or state transition diagrams can be viewed as special cases of CGs [Way, 1994]. CGs are semantically equivalent graphic representation for first order logic (FOL) formalisms. CGs can model many organizational concepts like roles or actors [Sowa, 1998], process [Mineau, 1999], events, procedures, state, situations, and context [Sowa, 1995]. As well there are many tools to represent CG in XML formats [Martin and Eklund, 1999]. CGs have been used to embed knowledge in an XML document [Martin and Eklund, 1999].

Projection and isomorphism are the basic tools used to query knowledge represented by CG. In many cases projection and isomorphism are NP-complete [Cogis and Guinaldo, 1995]. Projection is a function that conserves the structure, the concept and the relation with the conformity relation. To solve the NP-complete problem many researchers use the following techniques [Cogis and Guinaldo, 1995] :

- Use backtracking techniques [McGregor, 1982].
- Restrict the problem to domain where the problem turns out to be polynomial [Mugnier and Chein, 1993a].
- Store the graph into a specified hierarchy and avoid unneeded comparison [Ellis, 1993].

- Restrict the problem to a finite set of graphs and use indexing code to retrieve these graphs [Ellis et al., 1994]

Our contribution in this paper can be summarized as modifying existing CG techniques to index conceptual structures using ontological methodology. In our previous work, we have defined how to use CG to build ontologies for tendering system. Here we will discuss some tools that we have built to facilitate indexing and matching among different partners. A detailed description of our system can be found in [Kayed and Colomb, 2001] [Kayed and Colomb, 2000] [Kayed, 2001]. Briefly, our system works as follows: Users (sellers or buyers) use mediator ontology to build their needs/offer. They send these structures to a mediator. Mediator checks if these structures are correct. Mediator stores and indexes them using an agreed ontology. When users need information they send their requests to mediator. Mediator does matchmaking using the ontology and users' policies (preferences).

This paper is organized as follows. Section two summarizes matching, indexing and retrieving techniques in CG literature. In section three we explain how to use CG to implement our ontologies and some definitions which will be used later. Section four discusses our algorithms for correctness, storing, and indexing these structures. Also we discuss how these indexing can be used to define different matching and retrieving policy. Section five concludes the paper.

2 Indexing Conceptual Graphs

In CGs there are many methods of indexing. Ellis [Ellis, 1995] implemented a partially ordered hierarchy to index chemical objects. Objects are arranged in hierarchy based on some partial order over objects. Fundamental operations construct and maintain an index based on such partial orders. Sorting CGs in a partially ordered hierarchy reduces the number of comparisons. The query contents determine its position in the hierarchy. A graph is reconstructed when the hierarchy is traversed then, a general subsumption algorithm is used to subsume the query by this graph [Ellis, 1995].

Wermelinger [Wermelinger, 1997] restricted himself to tree graphs (no cycle except from the same node) so the projection became polynomial. He relaxed the condition for projection from function (a unique value) to a relation and called it semi-projection. He proved that semi-projection is equivalent to projection when the graph is tree. He proposed a polynomial algorithm for semi-projection using the cover theory.

A Graph Descriptor (GD) is used to encapsulate some properties of a graph so two graphs can be compared using these GDs. Instead of looking in the whole graph we look in graph description first.

In injective subsumption domain, where each node of a subsumed graph is matched by only one of a subsuming graph, Ellis used GD to define different types of filters, which can be used in hierarchy to index a graph. Ellis used injective subsumption in the domain of indexing chemical objects. Cycles, cycle length, self-cycle, chain, etc. are descriptors used to build an index lattice for chemical objects. So if the structure has a cycle we do not need to look in any structure that does not have a cycle.

Guinaldo et. al [Cogis and Guinaldo, 1995], [Guinaldo, 1996] defined graph descriptor in different way. They defined a linear descriptor for a graph as a pre-ordered sequence of concepts, relations and the adjacency edges for each relation according to some functions. If the graphs generate descriptors that are linear and equal we can say that these graphs are isomorphism. That graphs generate the same descriptors (but not linear orders) means not isomorphism but that have something in common. So these descriptors can be used to filter and index graphs. Using these descriptors they proposed an isomorphism algorithm without using projection which in many cases is polynomial.

Dealing with descriptors is a way to avoid using projection for matching which is in many cases NP-complete problem. We are influenced by this work. But instead of using cycles or length of graph we use signatures. Moreover, instead of using linear descriptor for the whole graph we use a linear descriptor (signature) for part of the graph. In the tendering domain, the structures are too huge so dealing with pre-determined lattice is not applicable. Ellis et.al. [Ellis and Lehmann, 1994] need a hierarchy with size more than 11 million nodes to index graphs of 10 nodes. In tendering domain always we ask about part of the structure not the whole structure. Splitting the structures into ontological elements enables us to index other elements around these ontological signatures.

Signature is a graph with one relation linking two or more concepts. The signature of a relation type enforces a minimal coherence in knowledge representation. The signature may be used for elicitation or knowledge collection.

In our solution we collect concepts around relations. Since the number of relation is small, this makes indexing around relation signatures more efficient. At the same time this raises another problem. Reducing the number of relations and signatures will affect the level of generalization (abstraction) of the ontology. To illustrate this, suppose that a concept "Thing" subsumes the concept "Electronic Machine" which subsumes the concept "Computer". Users who are looking for a PC may accept to retrieve all sellers who sell computers but they will not accept to retrieve all sellers who sell any THING. Because of that the level of abstraction is important in building the ontology. To solve this problem we create different signatures for the same relation. If the level of abstraction is violated, we go down another level and create another signature for that relation. We create a linear ordered hierarchy for all concepts that are subsumed by each signature concept. These hierarchies are built from an Abstract Domain Ontology (ADO). We will talk about ADO later.

3 Ontological Indexing

Ontology-based (called also concept-based) search succeeds in locating artifacts (e.g., documents) in some repository in many applications. Knowledge users accomplish this by using ontology that a search engine applies as an index into repository. In this scenario, users identify their concepts from ontology, the search engine use these concepts to locate desired artifacts from a repository [Uschold and Jasper, 1999]. McGuinness [McGuinness, 1998] used description logic-oriented ontologies in Web-search engine. From query log analysis, she noted that ontologies have improved the search experience from the perspective of recall and precision as well as ease of query formation. Yahoo! categorized and classified documents in subject hierarchy. Yahoo! use this hierarchy to index and retrieve documents for his Web-search engine. In the following, we show how we use ontology to index CG structures.

3.1 Signatures and Ontologies

In our system [Kayed and Colomb, 2000] we have defined four types of ontologies. Two of them are relevant to this paper. These are the abstract domain ontology (ADO) and the tendering ontology. The Abstract Domain Ontology contains classes, which are abstract description of objects in a domain. The class has Class-ID, Class-Properties, Class-Synonyms, Class-Type, Relation, Sub-Class, and Axioms. A relation is a link between classes, and axioms are rules that govern the behavior of the classes. The abstract domain ontology represents a container of abstract data types for sellers' catalogs.

The Tendering Ontology represents the core ontology in our system. The basic part of it is the Tendering Conceptual Structures (TCSs). We divided them into three models: buyer, seller, and mediator models. The buyer model is divided into advertising model, query model, and policy model. The advertising model again can be divided into tender invitation, terms, objects (services), specification, and returned forms. For the sake of space, we will illustrate our tendering ontology by detailing one of the most important TCSs i.e. the Tendering Invitation Structure (TIS).

Tendering Invitation Structure The tender invitation structure (TIS) is to inform the tenderers of the scope of the procurement. The tender invitation provides basic information on the procurement and guidance to the tenderers on the participation. We derived the component of TIS from UN EDIFACT (Electronic Data Interchange For Administration, Commerce and Transport) ISO 9735 request for quote message [Blomberg and Lennartsson, 1997].

We have defined TIS as a nested CG, containing information about the scope of the procurement, the address and conditions of contracting entity, nature of contract, duration/completion of contract, eligibility, award criteria, rules on participation, and objects specifications.

Formally TIS is defined using the type function of CG as follows:

$Type \quad TIS(x) \quad is$
$[TCS : *x]-$
$(ATTR) \leftarrow [ContractingEntity]-$
$\quad (ATTR) \leftarrow [Address]$
$\quad (ATTR) \leftarrow [Name]$
$(ATTR) \leftarrow [ContractingDuration]$
$(ATTR) \leftarrow [Eligibility]$
$(ATTR) \leftarrow [AwardCriteria]$
$(ATTR) \leftarrow [ParticipationRules]$
$(ATTR) \leftarrow [Services])-$
$\quad (ATTR) \leftarrow [Identification]$
$\quad (ATTR) \leftarrow [Description]$
$\quad (ATTR) \leftarrow [Quantity]$
$\quad (Measure) \leftarrow [Unit]$
$\quad (RelevantDate) \leftarrow [ProductionDate]$
$\quad (RelevantDate) \leftarrow [Expirydate]$
$...etc.$

A signature is a graph relation that relates two or more concepts. It's also called a star graph. The following graph:

$$[ContractingEntity(CE)] \leftarrow (ATTR) \leftarrow [CE_Attributes]$$

is a signature for the relation (ATTR). We index conceptual structures around signatures of each relation.

3.2 Indexing Overview

We divide the indexing process into two stages: checking the correctness of a structure then indexing it according to its signature components. To illustrate this, suppose that a buyer wishes to invite sellers to his/her tenders and we wish to index a buyer TIS structure. The whole TIS is stored in a repository. The structure is divided into data signatures (DSs). These DSs will be checked according to the ontological signature. Then DSs are compared with the indexing list associated with each signature. Using binary-search-like algorithm we can determine the position for each DS in that indexing list. These lists are totally ordered by predefined rules of priority and predefined concept ordering sets. These rules will define the finding policies in retrieving process.

Each signature consists of two or more concepts and one relation. Different strategies can be defined to order these signatures. Usually we use the concept arity as our default strategy. We ordered signatures according to the concept with arity_1, arity_2, and so on. For each ordering strategy we create an ordering list. In retrieving data, we should determine which ordering list we are going to use. Usually we use the default. In a system governed by software agents, we could define this parameter in agent's policies. Different agents can use different ordering lists. Also, it could be defined according to the type of query. If a certain concept is queried, we could use the list with high priority for that concept.

In our indexing and retrieving procedure, we use a binary-search-like algorithm. Concepts should be totally preordered. Concepts in ADO are partially ordered. We define Numbering Rules to number the concept for each signature to be totally preordered. There are many ways to define these rules. The following is one example of these rules:

- Parents are always greater than their all children.
- Left nodes with their children are greater than right nodes.

Using the above rules and ADO (see figure 2), we numbered the following data signatures for the relation "Type-of":

$1 : [Service] \leftarrow (Type - of) \leftarrow [Electronics]$
$2 : [Product] \leftarrow (Type - of) \leftarrow [Electronics]$
$3 : [Service] \leftarrow (Type - of) \leftarrow [TV]$
$4 : [Product] \leftarrow (Type - of) \leftarrow [TV]$
$5 : [Service] \leftarrow (Type - of) \leftarrow [Computer]$
$6 : [Product] \leftarrow (Type - of) \leftarrow [Computer]$
$7 : [Service] \leftarrow (Type - of) \leftarrow [PC]$
$8 : [Product] \leftarrow (Type - of) \leftarrow [PC]$
$9 : [Service] \leftarrow (Type - of) \leftarrow [Pentium]$
$10 : [Product] \leftarrow (Type - of) \leftarrow [Pentium]$
$11 : [Service] \leftarrow (Type - of) \leftarrow [Laptop]$
$12 : [Product] \leftarrow (Type - of) \leftarrow [Laptop]$
$13 : [Service] \leftarrow (Type - of) \leftarrow [Car]$
$14 : [Product] \leftarrow (Type - of) \leftarrow [Car]$

Table1: Example of indexing Data Signatures

We adopt this numbering rule from the depth-first-search strategy. Software agents whose concern is generalization and specialization retrieval can use this strategy. Suppose an agent looks for a PC. This rule indexes all the children of PC down (e.g., Laptop, desktop, etc.) and all parents of PC up (e.g., Computers, Electronic machine). Agents, depending on their policies, can go down the index for more specific items, or up for more general items. These policies will control agents' behavior when they try to find the closest information to their needs. Other strategies also can be used. A topological ordering strategy

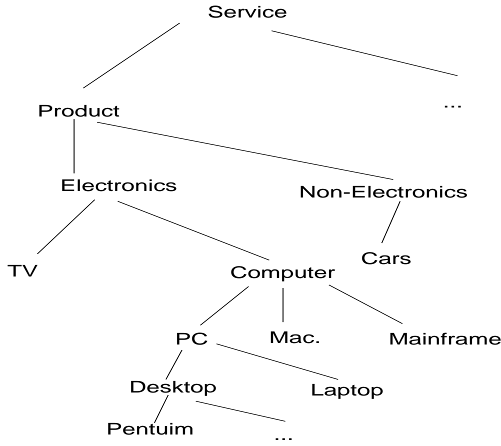

Figure 2: Part of concepts hierarchy

is used when agent is looking for a broad view about some concepts. In our example, agents can use this strategy when they are looking for the types of computer like PC, Mac, mainframe and so on.

We should note that ontological signature concepts are the maximal concepts in the indexing list i.e. signature concepts subsume all data signature concepts. The level of abstraction can be specified more in the agents' policy. Agents can traverse up or down only a specific number of levels determined by their policies.

In the following we will define some structures needed for our algorithms. Then we will explain the correctness and indexing algorithms in more detail.

3.3 Definitions

Definition 1. [Guinaldo, 1996] A conceptual graph is bipartite labeled graph **G=(R,C,E,Lab)**. **R** and **C** denote the set of **R**-vertices (relation nodes) and **C**-vertices (conceptual nodes) respectively. **E** is the set of edges. Each vertex $\mathbf{x} \in \mathbf{R} \cup \mathbf{C}$ has label Lab(x). Labels of vertices are ordered by two partial orders $(T_C, \leq_{T_C})$ and $(T_R, \leq_{T_R})$. The set of the edges adjacent to each **R**-vertex **r** is numbered from 1 to degree(r) and $G_i(r)$ denotes the ith **C**-vertex adjacent to **r** [Guinaldo, 1996].

Let **TCS** be a set of CGs for tendering conceptual structures. Let **R** be the set of relations for a tendering structures **TCS**.

Definition 2. [Sowa, 1999a] A star graph is a CG that contains a single conceptual relation and the concepts that are attached to each of its arcs.

Definition 3. A signature $S_i(R) = (C, E)$ represents the ith star graph which fixes the arity of a relation R and shows the greatest concept types this relation type can link. C denotes the set of Concepts and E is the set of edges linking the concept with R. In [Mugnier and Chein, 1993a] the set of all star graph is called the basis of the support (well-formed graph) and there is only one signature for each relation.

Definition 4. Let $Ordering_Strategy$ be a set of all possible linear sequences for a finite set of concepts.

Definition 5. Let $L(C_i)$ be the set of concepts subsumed by the concept C_i. Let $TOL(C_i, J)$ be a set of total pre-ordered concepts subsumed by the concept C_i under $Ordering_Strategy$ J. Let $<_{TOL(C_i,J)}$ be a subsuming relation defined under this total pre-order.

Definition 6. Let $CPR(S_i(r), J) = C1, C2, ..., Degree(r)$ be a linear sequence of concepts of a specific signature **i** for a relation **r** under $Ordering_Strategy$ **J**.

As a general policy, we always order referents using normal keyword ordering. In CG sense, the * dominates all other referents and also the top concept T dominates all other concepts.

3.4 Correctness and Indexing Procedures

Using the previous definitions we define the correctness and indexing algorithm.

Correctness Checking Procedure : The idea here is to check whether a structure is correct or not. If all its elements are correct according to the ontological signatures then we can store the data in the repository and add each signature of this structure in the appropriate indexing list.

Input: A **TCS** structure

Output: **T = True** if each element of TCS follows a signature, otherwise false. LS is an array contains signatures with the address of their associated index list, LS will be empty if T = False.

Start:
0-T = True
1-Split the TCS structure into star graph form
2-Store results of step 1 in the set **SEQ**
3-For each signature **S** *in* **SEQ** *Do*
  Find the appropriate signature that matches **S** *in the repository by*
    Find a relation **r** *in the repository that matches the Relation of* **S**
    If **r = Empty** *then* **T = False***: Return*
    Else
    For each signature S_i of **r** *DO*
      Check if all Arity Concepts of **S** *are <= of all Arity Concepts of S_i respectively If the result is True for S_i then go step 4*
      Else
      Next signature S_i
    T = False*: Return*
4-Insert the signature **S** *and the address of associated index list of S_i into* **LS**
5-Return

Suppose we like to check the correctness of the data signature $[Product] \leftarrow (Type-of) \leftarrow [PC]$ of some TIS structures. We find that the relation **Type-Of** has many signatures, one of them is $[Service] \leftarrow (Type-Of) \leftarrow [Product]$. Since $Product < Service$ and $PC < Product$ according to the concept type hierarchy (see figure 2) so we return the index list for the signature $[Service] \leftarrow (Type-Of) \leftarrow [Product]$.

If all elements of data structure is correct then we store the whole structure in the repository and start insert its data signatures in the indexing list.

Procedure $Indexing_data_structure$

Purpose: Insert data structure in the signature-indexing list using concept priority rule **K** and Pre-ordering strategy **J**.

Input : The Array **LS**

Start:

For each element in **LS** Do
L = **LS(i).Index_List.(K,J)**
S = **LS(i).Signature**
B = The position of the first element in **L**
E = The position of the last element in **L**
Pos = **Adding-Signature(B,E,L,S)**
Insert the signature at the position **Pos**
Next element
End Procedure

Function *Adding_Signature*
Purpose: Add a signature **S** in the right place in the index list **L** by using Binary Search like techniques and priority indexing rules.
Input : **B** the first position in the list **L**, **E** the end position in the list **L**, **S** the signature to be added in the list **L**, the Degree(**r**) is function returns the number of arity of a relation **r**.
Output: The position where to add the signature
Start:
$\mathbf{P}_1 = \mathbf{1}$ ′ Priority level 1 for concept caparison, 2 for referent comparison.
$\mathbf{P}_2$ = First element in **CPR(S, K)**
M = **1** ′ The Middle Element in the list **L**
S1:
If $E < B$ then Go **Terminate1**
M = $INTEGER(B + E)/2$
Let **x** be the concept vertex of arity $\mathbf{P}_2$ of **S**
Let **c** be the concept vertex of arity $\mathbf{P}_2$ of **L(M)**
If $\mathbf{x} <_{TOL(x,J)}$ **c** Then **B**= **M** + **1**:Go **S1**
If $\mathbf{x} >_{TOL(x,J)}$ **c** Then $E = M - 1$:Go **S1**
If $\mathbf{x} =_{TOL(x,J)}$ **c** Then
(*We check if we can go a lower level comparison priority*)
If $\mathbf{P}_2$ <Degree(**r**) Then
$\mathbf{P}_2 = \mathbf{P}_2 + +_{UnderPriorityOrder\mathbf{K}}$ **1**
Go **Compare2**
If $\mathbf{P}_2$ =Degree(**r**) Then $\mathbf{P}_1 = 2$:$\mathbf{P}_2 = 1$:Go **Compare2**
Compare2:
Case $\mathbf{P}_1$
1: (Concept Comparison):
Let **x** be the concept vertex of arity $\mathbf{P}_2$ of **S**
Let **c** be the concept vertex of arity $\mathbf{P}_2$ of **L(M)**
If $\mathbf{x} <_{TOL(x,J)}$ **c** Then
$B = M : P_1 = 1 : P_2 = 1$:Go **S1**
If $\mathbf{x} >_{TOL(x,J)}$ **c** Then
$E = M : P_1 = 1 : P_2 = 1$:Go **S1**
If $\mathbf{x} =_{TOL(x,J)}$ **c** Then
If $\mathbf{P}_2$ <Degree(**r**) Then
$\mathbf{P}_2 = P_2 + +_{UnderPriorityOrder\mathbf{K}}\mathbf{1}$
Go **Compare2**
If $\mathbf{P}_2$ =Degree(**r**) Then
$\mathbf{P}_1 = 2 : \mathbf{P}_2 = \mathbf{1}$
Go **Compare2**
2: (Referent Comparison)
Let **x** be the referent of the concept vertex of arity $\mathbf{P}_2$ of **S**
Let **c** be the referent of the concept vertex of arity $\mathbf{P}_2$ of **L(M)**
(* If **x** has no referent this means that **x=c***)
If **x**<**c** Then
$B = M : P_1 = 1 : P_2 = 1$:Go **S1**
If **x**>**c** Then
$E = M : P_1 = 1 : P_2 = 1$:Go **S1**
If **x=c** Then
If $\mathbf{P}_2$ <Degree(**r**) Then
$P_2 = P_2 + +_{UnderPriorityOrder\mathbf{K}}\mathbf{1}$
Go **Compare2**
If $\mathbf{P}_2$ =Degree(**r**) Then
Go **Terminate2**
End Case
Terminate1:
If **x**<**c** then Return **M+1**
If **x**> **c** then Return **M-1**
Terminate2:
(*We can Return **(M-1)** or Return **(M+1)** since there are equal*)
Return **(M+1)**

Explaining the algorithm The above function is a binary-search like algorithm. It aims at finding a position in a pre-ordered list to insert a certain signature. Since a signature contains two or more concepts, these signatures can be pre-ordered in different ways according to which concept you will start with and according to which hierarchy is being used to order the signature's concept. After setting these parameters, the *Indexing-data- structure* procedure calls *Adding-Signature* function to find where to add this signature. The function *Adding-Signature* starts in the middle of an indexing list, and compare the Indexed Signatures (ISs) with a Data Signature (DS). If an IS matches the DS we stop and insert the DS in that position. If not, we go up (if $DS < IS$) or down (if $DS > IS$) and repeat the process until we terminate. The first termination occurs if we jump out of the list (the end of list is less than the starting positing), so we insert the signature close to the last middle. The second termination occurs when we find a match, so we can insert the signature before or after that position. The main difference between this algorithm and binary-search algorithm is in way of comparing. Here we are comparing signatures. Using these lists data could be retrieved. A retrieving algorithm is the same as *Adding-Signature* function with small difference. In *Adding-Signature* function we try to find a position to insert a new DS, while in retrieving process we try to find the positing of an IS.

3.5 Ontological Matching

Formal ontologies define the roles, the concepts, and the relations between concepts. In the literature, there are many compatibility measures to check the similarity between concepts using ontology. In [Kayed and Colomb, 2000] we have defined different levels of matching. Starting from exact matching to weak matching. Our matching algorithm is the same as the insert procedure with little modification in *Terminate1* and *Terminate2*. In *Terminate1* we return no answer and stop. In *Terminate2* we return true and start traversing the index down or up for more answers. We stop traversing depending on agent's policy i.e. when the distance between the indexed signature (IS) and the data signature is zero or small.

To illustrate this, suppose a seller agent wants to find a buyer who likes to buy a PC. Two signatures can be formulated as follows:

$$DS1 : [Product] \leftarrow (Type - Of) \leftarrow [PC]$$

$$DS2 : [PC] \leftarrow (ATTR) \leftarrow [Name : Dell].$$

Suppose we have the following ontological signatures:

$$S1 : [Service] \leftarrow (Type - Of) \leftarrow [Product]$$

$$S2 : [Product] \leftarrow (ATTR) \leftarrow [Name].$$

The relation "Type-Of" and "ATTR" may have other signatures. DS1, DS2 are correct since they are

subsumed by S1, S2 respectively. Using the arity policy, we easily can find that the indexed signature (IS) in line 7 table 1 is the same as DS1. Indexed signature points out to structures in the repository that contains these signatures. Using the same method we can find the address for DS2. The common addresses will be returned as answers for the query.

If we traverse down line 7, we note that there is a "Kind-Of" relation between the corresponding concepts of lines 8-12 and DS1. We stop traversing down line 13 since there is no relation between Car and PC in ADO.

Depending on the relation between corresponding signatures, we define the following matching types:

- Exact matching: Each concepts of DS is the same as IS i.e. DS-IS=0.
- Synonym matching: All concepts in DS has "Synonym" relation with the corresponding concepts in IS (i.e. $DS -_{Syn} IS = 0$). These relations are calculated from ADO. We use WordNet [A.Miller, 1995] to add synonyms relation to our ADO ontology.
- Relational matching: There exist a relation **R** between the corresponding concepts signatures (i.e. $DS -_{R} IS = 0$). Where **R** is any relation like kind-Of, Part-Of, etc.

Our algorithm provides two types of comparison. One compares concepts and the other compares referents. We use the * in the referent side to indicate any referent. Referents are compared by keyword matching. User can use * to ignore referent comparison. Since we use binary-search like algorithm, the complexity of the algorithm is log(n).

3.6 Discussion

Matching between a query (which is a CG) and the facts knowledge in the repository is done by finding a projection between the query and some knowledge facts. Mugnier et. al. [Mugnier and Chein, 1993b] proved that finding a projection between two graphs is an NP-complete problem. They also proved that if the projected graph is tree then the projection turns to be polynomial. In our case, we build the query around the signature, which is a tree. So projection using signature is polynomial. To locate data efficiently, we sort these data signatures using an ADO ontology. We retrieve data by a binary-search like algorithm. By using the ontology to locate data we gain the following:

- We could split the query into signatures which turn it from graph to tree which changes projection to polynomial.
- Ontology enables us to sort signatures in a totally ordered set, so we could apply a binary-search like algorithm which turns the complexity to log(n).
- Using different policies to order, index, match, and retrieve data. This helps us to build a mediated-agent-based tendering system.
- Data signatures ordering list reflect the generalization/specialization properties. We use these properties to traverse the list for more likely answers or to apply soft-matching.
- The predefined signatures enable us to define bid evaluation rules. We can determine which particular signatures from seller model are best fit with other signatures from buyer model.

Other methods of indexing CG (e.g., [Ellis, 1993]) direct their effort to index the whole graph in a hierarchy. We use the ontology to index graph's parts (signatures) where these parts point to the whole graph (structure) in the repository. The explicit ontology and the numbering rules enable us to define a subsuming relation among signatures. This enables us to retrieve data as well as structures by a binary-search like algorithm.

4 Conclusion and Future Work

There are increasing interesting in ontology in many domains. Ontological analysis will be the first step in building a robust online system. The web is changing the way of building systems. In an open environment, is not enough to define terms to be used by internal users. Now the whole world needs to understand your terms. The whole world will be your customer. The complexity in building online systems should reflect simplicity on user side. Define your term, classify them, formalize them, using an existing or updated an existing ontology, and providing tools to solve any semantic conflicts become essential steps before putting a system online.

In this paper, we defined different types of matching through different relations defined in the ontology. This helps software agent in performing different type of policies depending on the users' needs. Agency means that agent can behave on behalf of the user, but this behavior is controlled by given strategies or policies. These policies define how the agent could work autonomously. We use relations to apply different type of matching. All these relations can be extracted from the ADO ontology.

We define our indexing procedure according to concept order set. This ordering can be simple or complex according to the user preferences. We usually use the Is-A relation to build these sets. In the future we will use other relation like part-of, collection-of, and feature-of, etc. These set will be built according to the application domain.

We retrieve query by matching query signatures and indexed signatures. The distance between the indexed signature and data signature is calculated. If the distance is acceptable we start traversing for more relaxed matching. We used exact match or any relational matching determined by agent policy.

The distance between signatures is calculated according to the relation types among the corresponding concepts. To say that a signature S_1 is a "Type-Of" S_2, all concepts in S_1 should have a "Type-Of" relation with S_2. In our future work, we will improve it so that all concepts in S_1 should have a relation subsuming the "Type-Of" relation.

Since our structures are built upon predefined signatures, our ongoing work is to define rules of matching for bid evaluation. A signature S_1 in structure TCS_1 is best fit with signature S_2 in structure TCS_2. We are defining rules that match signatures between tender invitation and sellers profile.

The contribution of this work can be summarized as use of ontological matching with existing CG indexing technique to index tendering structures. This method can be applied in any domain where semantic matching is required. Our work is distinguished from other in that; our work is care about the data as well as the structure, matching parts of graphs not only the whole graphs, and indexing structures using ontological methodology.

5 Acknowledgment

The authors acknowledge the School of ITEE at the University of Queensland for financial support for this project.

References

[A.Miller, 1995] A.Miller, G. (1995). Wordnet a lexical database for english. *Communications of the ACM*, 38(11):39–41.

[Blomberg and Lennartsson, 1997] Blomberg, P. and Lennartsson, S. (June 1997). *Technical assistance in Electronic Tendering Devlopment-FINAL REPORT Technical assistance in electronic procurement to EDI - EEG 12 Sub-group 1.* http://simaptest.infeurope.lu/EN/pub/src/main6.htm.

[Cogis and Guinaldo, 1995] Cogis, O. and Guinaldo, O. (1995). A linear descriptor for conceptual graphs and a class for polynomial isomorphism test. *Lecture Notes in Computer Science*, 954:263–277.

[Ellis, 1993] Ellis, G. (1993). Efficient retrieval from hierarchies of objects using lattice operations. *Lecture Notes in Computer Science*, 699:274–284.

[Ellis, 1995] Ellis, G. (1995). *Ph.D. Thesis: Managing Complex Objects.* Computer Science Department, University of Queensland.

[Ellis and Lehmann, 1994] Ellis, G. and Lehmann, F. (1994). Exploiting the induced order on type-labeled graphs for fast knowledge retrieval. *Lecture Notes in Computer Science*, 835:293–308.

[Ellis et al., 1994] Ellis, G., Levinson, R. A., and Robinson, P. J. (1994). Managing complex objects in peirce. *International Journal of Human-Computer Studies*, 41(1,2):109–148.

[Guinaldo, 1996] Guinaldo, O. (1996). Conceptual graphs isomorphism: Algorithm and use. *Lecture Notes in Computer Science*, 1115:160–174.

[Kayed, 2001] Kayed, A. (2001). *Home Page. http : //www.csee.uq.edu.au/ ~ kayed/.*

[Kayed and Colomb, 2001] Kayed, A. and Colomb, R. (2001). *Book Chapter: Business to Business Electronic Commerce: The Electronic Tendering.* In Internet Commerce and Software Agents: Cases, Technologies and Opportunities, Edited by Syed M Rahman and Robert J Bignall, IDEA GROUP PUBLISHING Hershey (USA).

[Kayed and Colomb, 2000] Kayed, A. and Colomb, R. M. (2000). Conceptual structures for tendering ontology. In *The Second Workshop on AI in Electronic Commerce (AIEC 2000), conjunction with the Sixth Pacific Rim International Conference on Artificial Intelligence (PRICAI 2000), ISBN 0643066268*, pages 71–82.

[Martin and Eklund, 1999] Martin, P. and Eklund, P. (1999). Embedding knowledge in Web documents: CGs versus XML-based metadata languages. *Lecture Notes in Computer Science*, 1640:230–243.

[McGregor, 1982] McGregor, J. J. (1982). Backtrack search algorithms and the maximal common subgraph problem. *Software Practice and Experience*, 12:23–34.

[McGuinness, 1998] McGuinness, D. L. (1998). *Ontological Issues for Knowledge-Enhanced Search.* In N. Guarino (editor), Formal Ontologies in Information Systems, IOS Press.

[Mineau, 1999] Mineau, G. W. (1999). Constraints on processes: Essential elements for the validation and execution of processes. *Lecture Notes in Computer Science*, 1640:66–79.

[Mugnier and Chein, 1993a] Mugnier, M. L. and Chein, M. (1993a). Characterization and algorithmic recognition of canonical conceptual graphs. *Lecture Notes in Computer Science*, 699:294–304.

[Mugnier and Chein, 1993b] Mugnier, M. L. and Chein, M. (1993b). Polynomial algorithms for projection and matching. *Lecture Notes in Computer Science*, 754:239–253.

[Sowa, 1999a] Sowa, J. (1999a). Conceptual graphs: Draft Proposed American National Standard. *Lecture Notes in Computer Science*, 1640:1–46.

[Sowa, 1984] Sowa, J. F. (1984). *Conceptual Structures: Information Processing in Minds and Machines.* Addison-Wesley, Reading, Mass.

[Sowa, 1995] Sowa, J. F. (1995). Syntax, semantics, and pragmatics of contexts. *Lecture Notes in Computer Science*, 954:1–15.

[Sowa, 1998] Sowa, J. F. (1998). Conceptual graph standard and extensions. *Lecture Notes in Computer Science*, 1453:3–43.

[Sowa, 1999b] Sowa, J. F. (1999b). *Knowledge Representation: Logical, Philosophical, and Computational Foundations.* Thomson Learning, Stamford, Connecticut.

[Uschold and Jasper, 1999] Uschold, M. and Jasper, R. (1999). A framework for understanding and classifying ontology applications. In Benjamins, V., anf A. Gomez-Perez, B. C., Guarino, N., and Uschold, M., editors, *Proceedings of the IJCAI-99 workshop on ontologies and Problem-Solving Methods (KRR5)*, volume 18, pages 1–11, Stockholm, Sweden. CEUR-WS.

[Way, 1994] Way, E. C. (1994). Conceptual graphs: Past, present, and future. In Tepfenhart, W. M., Dick, J. P., and Sowa, J. F., editors, *Proceedings of the 2nd International Conference on Conceptual Structures : Current Practices*, volume 835 of *LNAI*, pages 11–30, Berlin. Springer.

[Wermelinger, 1997] Wermelinger, M. (1997). A different perspective on canonicity. *Lecture Notes in Computer Science*, 1257:110–124.