Controlling Access to a Digital Library Ontology - A Graph Transformation Approach

Subhasis Dasgupta, Aditya Bagchi
Electronics and Communication Sciences Unit,
Indian Statistical Institute, Kolkata

This paper presents a graph-based formalism for an Ontology Based Access Control (OBAC) system applied to Digital Library (DL) ontology. It uses graph transformations, a graphical specification technique based on a generalization of classical string grammars to nonlinear structures. The proposed formalism provides an executable specification that exploits existing tools of graph grammar to verify the properties of a graph-based access control mechanism applicable to a digital library ontology description. It also provides a uniform specification for con trolling access not only at the concept level but also at the level of the documents covered by the concepts including node obfuscation, if required. Authors have shown the need of using both positive and negative authorizations for effective access control to the DL ontology. However, it gives rise to a decidability problem. A view creation mechanism and associated algorithm has been presented as a solution to the decidability problem.

Keywords: Ontology, Digital Library, Multiple Inheritance, Access Control, Graph Transformation

1. INTRODUCTION

A Digital Library supports documents on different subject areas. Here, a set of related documents is usually referred by a common index. This common index may be formed by a set of common key words or a common author name or a common subject name etc. A subject area can again be the subset of another subject area forming a hierarchy of subjects. For example, documents on "Balanced Tree" will be the subset of documents on "Tree" which in turn will be the subset of documents on "Data Structure". Thus a classification structure may be formed for bibliographic indexing. Dewey Decimal [Dew 2010] Classification(DDC) is the most popular way to implement bibliographic index. Currently it is used by many countries and available in different languages. After being developed in 1876 by Mevill Dewey, this hierarchical classification schema has been revised many times. Here each concept/class/subject area is identified by a number that signifies the position of the concerned concept in the classification hierarchy. With the advent of semantic web, a new version of DDC has been published for Digital Library [Dew 2010] and made available online. Some research effort has also been made to build the taxonomy of Digital Library using DDC [Saeed and Chaudhry 2002]. Another popular classification scheme is the Library of Congress Classification(LCC) [LCC 1990]. It goes to a hierarchy of about 7 levels and offers alphabetic indexing facility. However, both of these classification schemes consider only super-class/sub-class link among concepts and no other semantic relationship. Moreover, underlying hierarchical arrangement considers tree structure only (i.e. a concept can have only one parent concept). For example a book entitled *Data Mining for Degradation Modeling* will usually be classified as 006.3 according to DDC classification scheme (i.e. under the subject area computer science). However, data mining can also be placed under Statistics, i.e. the class number is 31 in DDC. So, it is apparent that an environment is necessary where a subject area may be placed under more than one parent subject area for proper document classification.

A Digital Library environment also has a dynamic user population mostly accessing from remote locations. Other than individual identities, these users are also characterized by other

Author's address: Indian Statistical Institute , 203 B.T. Road , Kolkata, West Bengal, India Pin 700108
This work has been done under the project **Controlled Access to Document Over a Digital Library Ontology Under Multiple inheritance(2012 - 2014)**, funded by Indian Statistical Institute.

properties that in turn control access to the library. For example, to get access to certain group of documents, one may have to be a member of a particular user group or must be over certain age or must have a minimum level of academic qualification. Controlled access to digital library concepts is a challenging area of research. Any user or group of users, usually designated as subject must have appropriate authorization to exercise any type of access (Read, Write etc.). A typical authorization model for a DL must also support varying granularity of authorization ranging from sets of library objects to specific portions of objects. A good amount of work has already been done on the secured access of individual objects, particularly for text documents in XML environment [Carminati et al. 2005][Damiani et al. 2002][Damiani et al. 2000][Gabillon 2005][Bertino and Ferrari 2002][Farkas et al. 2006]. However, being in XML environment, all these research efforts consider the underlying structure as a tree, i.e. one node in the hierarchy can have only one parent.

A recent study on the formal model of digital library (DL) has suggested that a DL can be represented as an ontological structure [Gonçalves et al. 2008], where documents may be classified and stored against appropriate concepts present in the ontology. Each link/edge between two concepts offer the semantic relationship between the concepts. However once again, even this ontological structure needs to support the situation where a concept can have more than one parent concept and then should also offer controlled access on such a structure. In an earlier work, the present authors have shown that in a DL ontology if a concept is allowed to have multiple parents, it gives rise to a flexible access control system hitherto unexplored [Dasgupta and Bagchi 2011]. This multiple parents consideration changes the underlying structure of the ontology from a tree to a Directed Acyclic Graph (DAG). Consideration of multiple parents has two distinct advantages over the earlier tree structure. First, new subjects are getting developed as interdisciplinary areas, a tree structure cannot make proper semantic representation of knowledge unless a DAG structure is allowed. Once again, *Data Mining* should be represented as a child of both *Statistics* and *Computer Science*. Secondly, it will be revealed in later chapters that presence of multiple parents provides the opportunity of categorizing the documents into different document classes relevant for each parent subject area. As a result, a user accessing the concerned child concept through a particular parent class would get faster access to the relevant documents, since less number of documents will be searched. No doubt, such facility and structure are not available in any existing Digital Library. However, the authors are confident that the present research effort would offer a semantically enriched Digital Library Ontology.

In order to offer an ontology based access control system applicable to digital library, the authors have adopted a graphical specification technique based on a generalization of classical string grammars to nonlinear structures [Corradini et al. 1997]. A preliminary work of this approach has already been reported earlier [Dasgupta and Bagchi 2012]. This formalism is useful for following reasons:

(1) To elaborate the properties of a proposed Access Control specification for Digital Library Ontology

(2) To provide an executable specification that exploits existing tools to verify the properties of a graph-based access control mechanism applicable to a digital library ontology description.

(3) To provide a uniform specification for controlling access not only at the concept level but also at the level of the documents divided into document classes placed under different concepts. It also includes node obfuscation.

Similar graph-based formalism for modeling access control system has already been proposed. However, the earlier effort considered modeling from subject point of view where subject grouping (user-groups) and their hierarchies (role hierarchy) have been considered [Koch et al. 2005]. This research attempt, on the other hand, considers hierarchy of objects in the form of concepts in DL ontology. So graph transformation against the authorization set of a user would give rise to a section of the ontology where the concerned user would have access. The present paper discusses

an environment with single administrator where access of a single user is modeled using the formalism of graph transformation. However, this model can also be used for user-groups where the same authorization set will be applicable to all the members of a user-group. Inheritance of authorization as applicable to user hierarchy like role based system has not been considered here. On the other hand, object based inheritance as available in an ontological structure has been discussed. So an access right given to a concept definitely percolates down to the concepts below the concerned concept following the ontological hierarchy of the Digital Library.

After introducing the research effort in Section 1, Section 2 discusses the relevance of having multiple parents in a DL ontology alongside other technological preliminaries. Section 3 covers the access policy specification for the DL ontology. Section 4 deals with the graph transformation methodology and then provides the detail of the access control model for DL ontology. Section 5 draws the conclusion and also indicates about future work.

2. RELEVANCE OF MULTIPLE PARENT CONCEPTS IN DL ONTOLOGY

As explained earlier, in a DL ontology, a concept in the ontological hierarchy may need to have multiple parents. In other words, a concept may belong to more than one subject areas. In order to explain the requirement and to derive a flexible access control system for retrieving documents from a digital library a running example has been considered. Throughout the paper the same example will be used. So for example, in a DL ontology a concept named Database may be reached from Computer Science & Engineering (CS), Geographic Information System (GIS) or Biology/Bio-informatics (BIO). This consideration changes the underlying structure of the ontology from a tree to a Directed Acyclic Graph (DAG) as shown in Figure.1. Now, the three parent concepts of Database may have distinct or even overlapping user communities. As a result, any document under Database may be of interest to more than one of the above three user communities. Research work, done so far, for controlling access to a digital library ensures that a user must possess appropriate authorization to get access to a concept. However, if access to a concept is granted, all documents under it are available to the concerned user. Some work has already been done to control access even at the document level. In other words, a user getting access to a concept may not get access to all the documents covered by that concept, particularly in a situation when a concept has multiple parent concepts [Dasgupta and Bagchi 2011].

Now, if a concept has multiple parent concepts, members of different user communities corresponding to different parent concepts may like to have access to different sets of documents even when they are covered by the same child concept. Consequently, the documents covered by a concept can be categorized into number of document classes. Now the users having authorization to access some of the parent concepts but not all, will get access to some of the document classes under the child concept. In other words, a user can access a document only if he/she has appropriate authorization to access the document class in which the document is placed. Figure.1 shows an environment where documents covered under the concept Database may be contributed by or of interest to any users of the parent concepts. So a document under the child concept Database can be a member of one or more than one of the parent concepts. Consequently, documents under a child concept having n parents, can be classified into $(2^n - 1)$ categories. So, the Database concept in Figure.1 can be classified into $(2^3 - 1)$ or 7 categories. Figure.2 shows the Venn diagram corresponding to the concept Database having three parent concepts Computer Science (CS), Geographic Information System (GIS) and Bio-Informatics (BIO) as mentioned earlier.

Situation depicted in Figure.1 and Figure.2 is very common in case of a digital library and a document under the concept Database may be of interest to the users of CS/GIS/BIO or any combinations of them. However, the existing implementations avoid such document classification possibility and keep the child concept under only one parent concept. Such parent concept is usually the one that contributes maximum number of documents. Accordingly, the concept Database in a DL ontology will possibly be kept in the CS path with all documents under it.

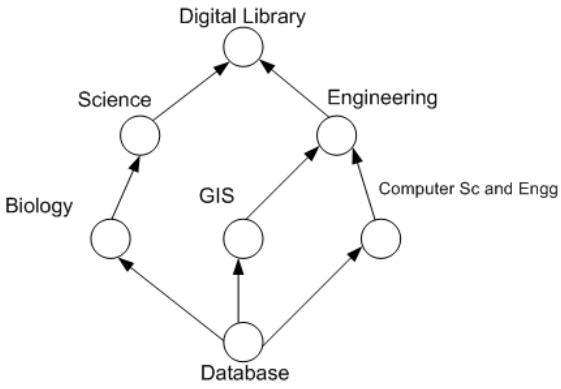

Figure 1. An ontological structure with the concept Database having three parent

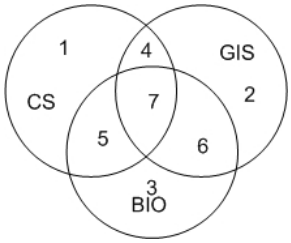

Figure 2. Possible document categories under the common concept "DATABASE"

Any user of GIS or BIO community will directly access the Database concept and would be able to get all the documents under it if he/she has authorization to access Database concept directly. The proposed system, on the other hand, provides $(2^3 - 1) = 7$ document classes as shown in Figure.3. Depending on the parent concept where a user is authorized, corresponding document classes under Database concept and hence the documents covered by them can be accessed.

2.1 Advantages of Document Classification

While a usual ontological structure can control access at the concept level only, availability of document classification helps in implementing access control even at the document class level going beyond the concept level. This facility can also provide more flexible access control policies. Different access control policies may be offered to different user communities. Figure.3 shows the 7 possible document classes under the concept Database. Table I shows the subject areas (parent concepts of Database) that are associated with each such document class. While class 1, 2 and 3 are storing documents exclusively for CS, GIS and BIO respectively, other classes hold documents meant for more than one parent node. Now for example, if there are two user-groups UG1 and UG2, two different access control policies may be adopted. If members of both the groups are authorized to access the nodes in the CS path only but not of GIS or BIO, at the concept Database, however, members of two groups may get two different types of access. Members of UG1 may get access to all documents under document classes 1,4,5 and 7. While documents under class 1 are exclusively for CS group, documents under the other three document classes 4,5 and 7 belong to more than one subject area. Now adopting a different access control policy, members of UG2 may get access to documents under document class 1 only. For documents under class 4.5 and 7, they may get only the abstracts. This is an additional facility that can be extended in the access control system discussed in this paper. Different access control policies can be taken for different user-groups depending on their credentials.

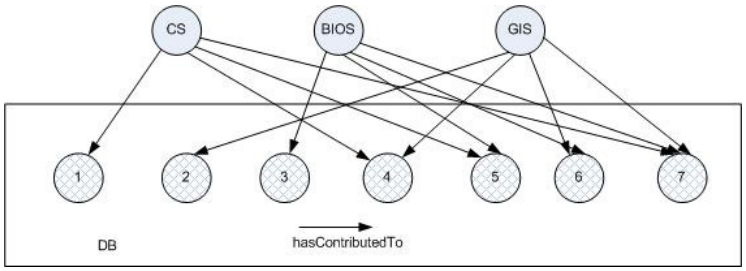

Figure 3. *hasContributedTo* Relationship for all the Classification

Table I. Document Class

Document Class	Access
1	CS
2	GIS
3	BIO
4	CS , GIS
5	CS , BIO
6	BIO, GIS
7	CS, BIO, GIS

2.2 Technological Preliminaries

Different research efforts on ontology based systems and secured access to ontological structure have already been reported [Kashyap and Sheth 1996][Qin and Atluri 2010][Ouksel and Ahmed 1999]. This section describes the related technologies.

(1) **Ontology:** The fundamental objective of Ontology is to form a semantic network based system to organize concepts in a directed graph structure and to provide a mechanism to search a concept from such a structure through which a given schema element is referred. It also finds other related elements/concepts in the ontology. So an ontology can be defined by a directed graph, where each node is a concept. If O is an ontology represented as $O = (C, L)$ then C is a concept, and L is the link between two concepts representing their semantic relationship.

(2) **Properties:** Each ontology has a set of properties, classified into either object property or data property. Data property describes about the data and Object property deals with the concepts. All domain property of a concept c can be represented as $P(c) = DP(c) \bigcup OP(c)$ [Qin and Atluri 2010], where $DP(c)$ is the data property and $OP(c)$ is the concept property. The proposed DL ontology has two types of links to represent semantic relationships among nodes: $isSubClassOf$ and $hasContributedTo$. These are object properties. In Figure.1, $Biology$ $(isSubClassOf)$ $Science$, so the OP is $(C_{Biology}.(isSubClassOf)) \in \{Science\}$.

(3) **Concept:** Each ontology has a set of semantically related concepts, $C(o) = \{c_1, c_2, ..., c_n\}$. Fig. 1 is showing a Digital Library(DL) ontology, hence $C(DL) = \{c_{database}, c_{biology}, c_{computerSCandEngg}, ..., c_{DigitalLibrary}\}$.

(4) **Concept Hierarchy:** Ontology structure, as described in this research effort, is a DAG, where a concept may have more than one parent concepts. Concepts in an ontology o can be represented as a partially ordered set. Given two concept $(Science, Biology) \in (DigitalLibrary)$ where $isSubClassOf(Biology) = Science$, i.e. Biology is more specialized than Science. It can also be denoted as $C_{Biology} \prec C_{Science}$.

(5) **Document Class:** Documents in a concept are classified into several classes depending upon the number of first degree parents. If a concept has n number of parents then the concept should have $(2^n - 1)$ number of document classes. In Figure.1 database has three

parent concepts. So the documents covered by the concept database has 7 possible document classes.

(6) **Document Annotation:** Gonçalves et. al. has published some work on ontological representation of digital library [Gonçalves et al. 2008]. Concept of an ontology can be identified by it's URI. In the present system, a document is identified by its concept URI with a unique document class and document-id suffixed.

3. POLICY SPECIFICATION AND AUTHORIZATION INHERITANCE

Authorization specification in the proposed model has the following entities:

(1) Subject : A Subject may refer to an individual user or a user-group. $S = \{s_i\}, i \geqslant 1$

(2) Object : An Object can be a document or a document class or a concept or the entire ontology. $O = \{o_j\}, j \geqslant 1$

(3) Access rights : In this paper, the proposed access control model has considered *read* and *browse* access rights only. $A \in \{read, browse\}$. A valid user can browse through all the nodes by default but would need explicit positive authorization to read any document under a concept. $A = \{a_k\}, k \geqslant 1$. Importance of browse by default has been explained later. Addition of new documents and modification of existing documents will be considered in the future extension of the model.

(4) Sign : Sign are of two types, $ve+$ or $ve-$, positive for access permission and negative for explicit denial. $V = \{+, -\}$ and $V = \{v_t\}, t \geqslant 1$.

So an authorization in the system is defined by a four tuple (s, o, a, v) signifying that subject s is authorized to access object o with access right a and sign v.

As mentioned earlier, the DL ontology proposed in this paper supports two semantic relationships: $isSubClassOf$ and $hasContributedTo$.

(1) **isSubclassOf** : *isSubClassOf* relationship represents a partially ordered set. In the graph model, *isSubClassOf* represents the parent-child relationship. In Figure.1, Biology *(isSubClassOf)* Science i.e. in the ontology Biology will be a child concept of Science. *isSubClassOf* is a non-commutative property. However, this property is transitive. So, if B is *isSubClassOf* A and C is *isSubClassOf* B, then C is *isSubClassOf* A as well.

(2) **hasContributedTo** : In the proposed DL ontology a new set of nodes have been added. These nodes are the document class nodes. *hasContributedTo* represents the semantic relationship between a document class and its supporting concept. As discussed earlier, in case of n multiple parent concepts, a child concept will have $(2^n - 1)$ document classes under it. So, *Database* concept having three parent concepts *Biology* , *GIS* and *Computer Sc and Engg* will have 7 document classes under it, as shown in Figure.3. However, Figure.3 shows the links in such a way as if the 7 document classes are connected directly to the parent classes with *hasContributedTo* links. As a matter of fact, the document classes are actually linked to the child class *Database* only. Figure.3 showed it differently in order to explain which parent classes are involved with which document classes. The idea is clearly shown in the associated Table I. This relationship is also non-commutative.

Generically, the inter-concept relationships applicable to the DL ontology described so far are:

(1) Inclusion($\odot$) : $(C_i \odot C_j)$ signifies that concept C_i is included in concept C_j. Inclusion relationship is non-commutative i.e. $C_i \odot C_j \neq C_j \odot C_i$. However, Inclusion relationship is transitive i.e. if $C_i \odot C_j$ and $C_j \odot C_k$, then $C_i \odot C_k$.

(2) Inferable :($\Longrightarrow$) : If a concept C_i infers the existence of another concept C_j then C_j is inferable from C_i, i.e $C_i \Longrightarrow C_j$. Inferable relationship is non-commutative , .i.e $C_i \Longrightarrow C_j \neq C_j \Longrightarrow C_i$, and transitive i.e. if $C_i \Longrightarrow C_j$ and $C_j \Longrightarrow C_k$, then $C_i \Longrightarrow C_k$.

Since the present paper considers only two types of semantic relationships,*isSubClassOf* relation between concepts and *hasContributedTo* relation between concepts and document classes, Inclusion and Inferable would virtually be similar.
This Inclusion/Inferable relationship provides the facility of authorization inheritance in the DL ontology. A positive authorization to a concept in the ontology not only gives access to all the documents under that concept, but also the documents of all the child concepts under the concerned concept in the ontology. So, an authorization to a concept in inherited by all the concepts in the subgraph below the concept node originally authorized.

(3) Partially Inferable: ($\rightharpoonup$) : If C_i and C_j are two concepts, then $C_i \rightharpoonup C_j$ signifies that the concept C_i can partially infer the concept C_j. This relationship is also non-commutative and transitive. This relationship will be particularly important for concepts with multiple parents. As shown earlier, concept *Database* is partially inferable from the three parent concepts. So if a user is authorized to access only the parent *GIS*, for the child concept *Database* the user will inherit the authorization only partially, i.e. it can access documents under the document classes 2, 4, 6 and 7 only, as depicted in Figure.3 and associated Table I.

(4) Non Inferable ($\not\Rightarrow$) : If a concept C_i cannot infer the existence of another concept C_j, the relationship is non-inferable. So considering the ontological structure non-inferable relationship signifies that there is no path from concept node C_i to concept node C_j.

The present research effort has assumed that a user can make only read and browse accesses to a concept node. A read authorization allows access to documents under a concept. A browse access has been considered by default. A browse access is available only if no explicit authorization is assigned to a concept node. A negative authorization, however, makes a concept node and the subgraph below it, unavailable to a user. A default browse authorization is necessary for traversal through the ontology by any concept search algorithm and for node-obfuscation, discussed later in this paper.

Discussion made so far indicates that a user accessing the DL ontology will have explicit authorization at some concept nodes and he/she will also get access rights in some other nodes by inheritance. If P offers the set of explicit authorizations for a user in the form of four tuples (s, o, a, v) described earlier, P_c be the complete set of authorizations available to the same user considering inheritance rules. Without going into the detail of the Policy Algebra as presented by the authors in another paper [Dasgupta and Bagchi 2012], the inheritance rules may be described as:

(1) Reflexivity Rule: All tuples in P are inherited by P_c.

(2) Inheritance Rule: If concept C_i infers concept C_j, i.e $C_i \Longrightarrow C_j$, then any authorization explicitly specified for C_i will be inherited by C_j.

(3) Override Rule: While concept C_i infers concept C_j, i.e $C_i \Longrightarrow C_j$, if C_i has positive authorization for certain access right (only read in this case) but C_j has negative authorization for the same access right (only read in this case) over the same object/concept, then C_j will retain the negative authorization. Inheritance of positive authorization on C_j from C_i will not be available to the user. Not only for C_j alone, same negative authorization will be applicable to all the concept nodes inferable from C_j. In case the negative authorization on C_j is withdrawn later, the same user will inherit the positive authorization to C_j from C_i and hence will also have positive authorization on all other nodes inferable from C_j.

(4) Implicit Authorization Rule: As soon as a user logs on to the Digital Library system, the entire DL ontology with all its concepts will be made accessible to the concerned user. In other words, the Digital Library, in general, is an open system for access. However, before making any access, the credentials of the user will be checked to find his/her zone of access on the ontology. A credential verification sub-system is associated with the Ontology Based Access Control (OBAC) mechanism. Importance of credential verification has been discussed

by Qin and Atluri [Qin and Atluri 2010]. For example in Figure.1, a user from Computer Science & Engineering (CS) may be allowed to access all nodes/concepts inferable from CS node only. However, initially, at the time of logging on to the system, all nodes were accessible to the users. So, nodes not relevant to CS will have to be blocked. Hence, other nodes and paths (in this case, GIS and BIO) will get implicit negative authorization, so that the document classes not relevant to CS are not accessible to the concerned user. This requirement, particularly for selective access on a DAG structure justifies the introduction of negative authorization.

This implicit authorization that causes the introduction of negative authorization, creates a decidability problem for inferring authorization by inheritance. Referring to the earlier example, concept *Database* has multiple parents and the user under consideration is receiving both positive and negative authorizations for reading documents under *Database* concept. This situation will not be a problem for accessing *Database* concept itself, since it will select the relevant document classes which will be available to the concerned user. However, for any node/concept inferable from *Database*, i.e. for any child node of *Database*, authorization inherited from above will be undecidable. Any child node of *Database* is receiving both positive and negative authorization for reading. This problem is resolved by creating ontology views. Creation of such views will be discussed later in this paper.

4. GRAPH TRANSFORMATION AND ACCESS CONTROL MODEL

This section discusses the formal method of graph transformation, i.e. transformation steps and rules. A formal introduction to Graph based formalism can be found at [Corradini et al. 1997] and a RBAC implementation of the model has been developed by Koch et. al.[Koch et al. 2005].

4.1 Type Graph and State Graph

Here, the ontological structure of a Digital Library and the access control policies imposed on it are represented by a type graph. In a type graph the edges are directed, i.e. each edge runs from a source node to a target node. Each node represents a node type and each edge also has an edge type. Figure 4, represents the type graph of the proposed access control model for DL ontology. The type graph provides the node types au, u, p, c and dc. Node type au represents the administrative user. A node type administrative user may or may not be an actual user of a digital library but will administer or control all other node types. The model presented in this paper considers a single administrator, i.e. a centralized ontological structure accessed by all users of the Digital Library. A distributed environment with multiple administrators will be considered as a future research effort. Nodes of type u represent users. Node types c and dc are the concepts and document classes respectively. Node type p represents permissions that cover all the access rights available along with any other administrative permission needed. Edges from node type au to node types u, p and c represent the administrative control of administrative user on other node types. Edge from c to dc represents the document classes under a concept. A combination of edges that connect u to p and c to p represents the type of authorization given to a user for accessing a concept. These are all explicit authorizations, inherited access rights will be discussed later. A self loop on node type c represents the concept hierarchy. A type graph is a pattern for a class of graphs. A graph G will be a member of this class if each node and edge in G has a corresponding node and edge type in the type graph. Each such member graph having more than one instances of the different node types described above represents a system state and thus called a state graph. Figure 5 represents one such state graph. Figure 5 shows different instants of the permitted set of nodes present in the system state described. For example, u_1 and u_2 are two users. Since the proposed system developed so far is considering only *read* permission, the state graph has only one p type node. While user u_1 is yet to get any authorization to access any concept (perhaps a newly registered member of the library), user u_2 has got the permission (*read* in this case) to access the concept c_3. Edges running from one concept to another represent

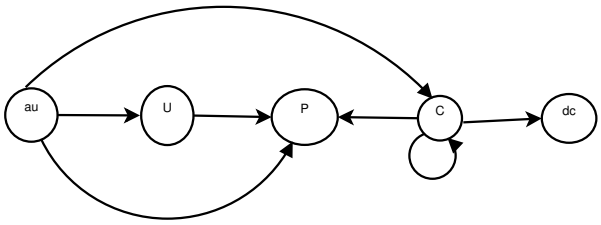

Figure 4. The Type Graph of Concept Hierarchy

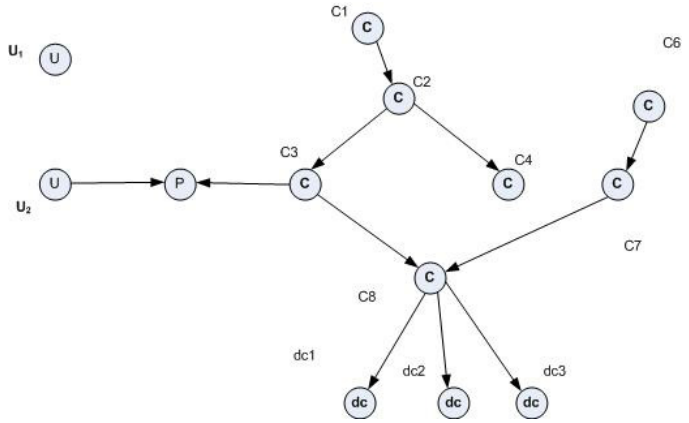

Figure 5. The State Graph of Concept Hierarchy

isSubclassOf relationship. So, c_2 *isSubclassOf* c_1, c_3 *isSubclassOf* c_2 etc. Concept c_8 has two parent concepts c_3 and c_7. So it has $(2^2 - 1)$, i.e. 3 document classes dc_1, dc_2 and dc_3 under it. An edge running from a c type node to a dc type node represents *hasContributedTo* relationship as explained earlier.

4.2 Graph Transformation

This section briefly introduces graph transformation rule. Formal introduction can be found in [Corradini et al. 1997]. Application of such graph transformation in a role-based access control environment has been discussed in [Koch et al. 2005]. As mentioned earlier, a graph represents a state of the system. Application of an access control policy, specified by a graph transformation rule, causes a change in system state given by a graph morphism $\{r : L \rightarrow R\}$, where both L and R are graphs, called left-hand side and right-hand side respectively. So, rule applied to a graph G causes a transformation $\{r : L \rightarrow R\}$ defined by a injective partial mapping of sets of nodes and edges of L and R. Mapping rule from L to R always maintains the types and labels of corresponding nodes. Nodes and edges of L for which no partial mappings are defined are deleted by the rule during transformation and will not be present in R. Nodes and edges of L for which partial mappings are defined, are preserved and have corresponding images in R. Nodes and edges of R not present in L are newly created by the rule applied.

4.3 Administrative Operations on a Concept Hierarchy using Graph Transformations

Authors of this paper have already done some work on access control model for an ontology based digital library [Dasgupta and Bagchi 2011], [Dasgupta and Bagchi 2012]. This section will show, how graph transformation rules can represent the basic administrative operations relevant to such access control model. Since, this paper considers a centralized digital library ontology under a single administrative domain, a single administrative user type node au has been shown in all the graphs. Future work will involve multiple sites and ontology views which would demand multiple

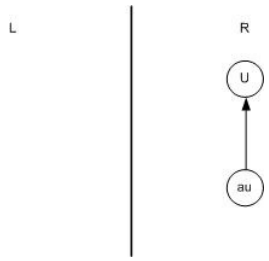

Figure 6. Add user

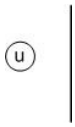

Figure 7. Remove User

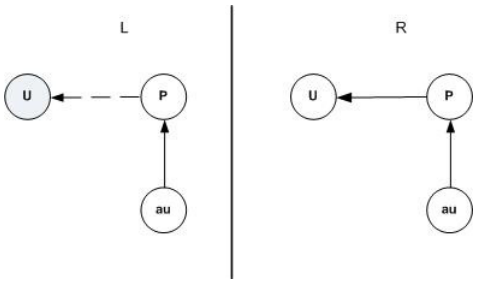

Figure 8. Assign Permission

administrative domains. Administrative operations considered here are *add user, remove user, assign permission, revoke permission, add concept and alter concept.* Algorithm 1 provides the necessary functions for these administrative operations.

(1) Add User : The graph transformation rule for *add user* has an empty $left - hand\ side$, while on the $right - hand\ side$, the administrator adds a new user. Figure 6 represents the scenario. A new user is created with no permission attached (i.e. with an empty permission set associated).

(2) Remove User : *Remove user* is also another trivial rule. Figure. 7 shows the *remove user* operation. this operation removes a user by deleting a user type node. That's why, the $left - hand\ side$ of the Figure shows a user type node while the $right - hand\ side$ is empty. Permission set attached to the removed user would automatically be removed as well.

(3) Assign Permission : This operation grants a permission (only *read* in this case) to a user. Figure. 8, shows the *assign permission* operation. $Left - hand\ side$ proposes an authorization to attach permission type node p to the user type node u. Hence, the administrative node au proposes a link from node p to node u, shown by a dotted line. If this permission assignment is allowable, then after the graph transformation, edge connecting p and u will be permanent, as shown on the $right - hand\ side$.

(4) Revoke Permission : This operation revokes a permission from an user. So an edge from a p type node to an u type node as shown on the $left - hand\ side$ of Figure. 9, is removed by the graph transformation rule, leaving only the user u on the $right - hand\ side$.

(5) Add Concept : Addition of a new concept to the ontological hierarchy is done by *add concept* rule. Any new concept added to the concept hierarchy, will be connected to its immediate

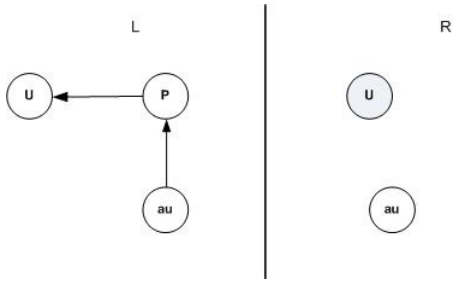

Figure 9. Revoke Permission

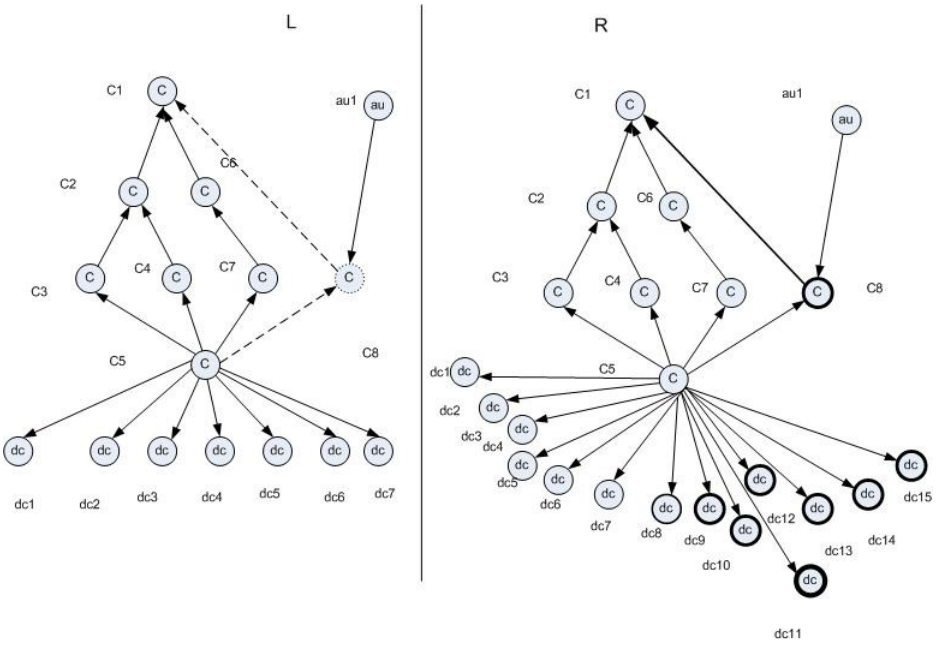

Figure 10. Add Concept

parent(s) node(s) and immediate child(children) node(s) by *isSubClassOf* links. So the rule should also introduce the required edges during $\{L \rightarrow R\}$ transformation. In addition, each new concept added to the ontology will have at least one document class (*dc* type node) created and connected to the new concept by *hasContributedTo* relationship. However, if a new concept is added as a parent of a child concept C_i already having one or more parent concept(s), more than one new document classes will be created depending on the number of parent concepts of C_i. Figure.10 shows the addition of a new concept C_8 as the parent of child concept C_5 that already has three parent concepts. Left side of the state graph shows that the administrator node *au* is proposing addition of a new concept C_8, which will be the child concept of C_1 and the parent concept of C_5. So, from ontological perspective C_5 $isSubclassOf$ C_8 and C_8 $isSubClassOf$ C_1. The proposed concept and the required edges have been shown by dotted line on the *left-hand side*. After transformation, *right-hand side* shows the corresponding permanent edges.

Before addition of the new concept C_8, concept C_5 had three parent concepts, C_3, C_4 and C_7. So there were $(2^3 - 1) = 7$ document classes under C_5, shown by the nodes dc_1 to dc_7 on the *left-hand side* of the Figure.10. Now, after addition of concept C_8 as the fourth parent concept of C_5, there will be $(2^4 - 1) = 15$ document classes under C_5, creating additional

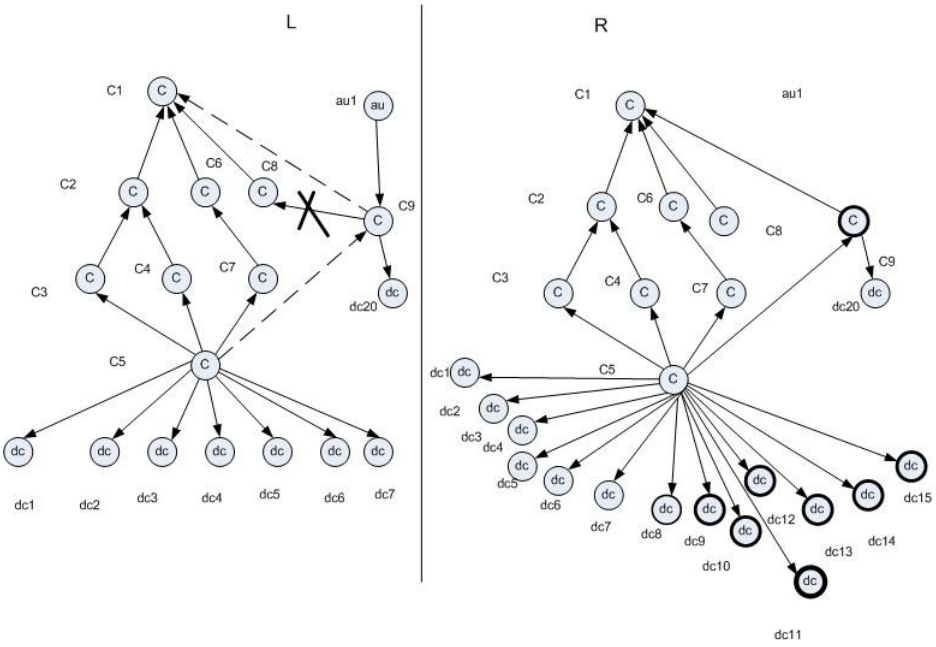

Figure 11. Alter Concept

document classes dc_8 to dc_{15} as shown on the *right-hand side* of Figure.10.

(6) Alter Concept : Since this research effort is dealing with library application, deletion of concept (usually representing a subject area) has not been considered. However, the concept hierarchy or the ontology structure may change. Alteration of Concept actually means the restructuring of the concept hierarchy. It is always possible that a concept/subject area in a Digital Library, as a result of its development, encroaches into the area of another subject. For example, a particular style of storage and retrieval of geographical entities using computers ultimately gives rise to Geographical Information System (GIS). So, GIS becomes a branch of both Geography and Information Technology. Hence in the DL ontology, alternation needs to be made to show GIS concept as a child of both Geography as well as Information Technology. In other words, an existing concept may change its position, thereby changing its parent(s) and child(children) concepts and also causing changes in the document classes under it. Figure.11 represents a state graph on the application of *alter concept* rule. As shown on the *left-hand side*, concept C_9 which was originally a child of the concept C_8 with no child concept of its own, has now been placed as child of C_1 and parent of C_5 disconnecting it from C_8. Proposal made on the *left-hand side*, has been made permanent on the *right-hand side* by graph transformation. In addition, the concept C_9, being added as the fourth parent of the concept C_5, gives rise to additional document classes dc_8 to dc_{15} under C_5 because of the same reason as explained in the *add concept* section. However, concept C_9 retains its own document class dc_{20} even after transformation.

4.4 Decidability Issues and Relevance of View Creation

Problem of decidability for inferring authorization has been an area of interest for many researchers. As a result, ample study has been made to establish the theoretical foundation of the decidability problem. Earlier under implicit authorization rule, the authors have admitted that for the presence of both positive and negative authorization to a child concept node C_i having multiple parents, inherited authorization to all nodes under the concept C_i will be undecidable. In order to avoid this problem, the authors have proposed the creation of user-specific ontology

Algorithm 1 Administrative Operations

```
1: function AddUser(user)
2:     if user ∉ USER then
3:         USER ← USER ⋃ user
4:         Permission = Permission ∪ {User → ∅}
5:     end if
6: end function
7: function RemoveUser(user)
8:     if user ∈ USER then
9:         USER ← USER\user
10:        Permission = Permission\{User → Permission(user)}
11:    end if
12: end function
13:
14: function AddConcept(Concept, ListOfParents, ListOfChildren)
15:    if Concpet ∉ O then                                  ▷ Where O is the Ontology
16:        Create(Concpet);
17:    end if
18:    for each Parent from ListOfParents do
19:        Add(Concept, isSubClassOf, Parent)
20:    end for
21:    Classify(Concept, ListOfParents)                     ▷ hasContributed links will be created
22:    for each Child from ListOfChildren do
23:        Add(Child, isSubClassOf, Concept)
24:        getListOfParent(Child)                           ▷ Finds the complete list of parents including new Concept
25:        Classify(Child, ParentList);
26:    end for
27: end function
28:
29: function AlterConcept(Concept, ListOfParentConceptAdd, ListOfParentConceptRemove, ListChildConceptAdd,
    ListChildConceptRemove)
30:    for each parent from ListOfParentConceptRemove do
31:        remove(concept, isSubClassOf, parent );
32:    end for
33:    for each child from ListChildConceptRemove do
34:        remove(child, isSubClassOf, concept );
35:        ListOfParentOfChildNode ← getListOfParent(child)   ▷ Find the complete list of parent including new
    Concept
36:        Classify(Child, ListOfParentOfChildNode);
37:    end for
38:    AddConcept(Concept, ListOfParentConceptAdd, ListChildConceptAdd)
39: end function
40:
41: function Classify(Concept, parentConcept )
42:    n = ParentConcept.size();
43:    classes ← (2^n − 1);
44:    for each class from classes do
45:        class ←(Hash Appropriate Parent Class)       ▷ Appropriate Class will be selected from the Combination of
    immediate Parent Classes
46:        Add(ParentConcept, hasContributed, class)
47:    end for
48: end function
49: function AssignPermission(user, Permission )
50:    if Permission ∉ PERMISSION(user) then
51:        PERMISSION(user) = PERMISSION(user) ∪ {Permission)
52:    end if
53: end function
54:
55: function RemovePermission(user, Permission )
56:    if Permission ∈ PERMISSION(user) then
57:        PERMISSION(user) = PERMISSION(user)\{Permission)
58:    end if
59: end function
```

views. Some researchers have already worked on the creation of ontological views and processing queries on them [Noy and Musen 2004] [Seidenberg and Rector 2006]. A user specific view will keep only those concept nodes where the concerned user has positive authorization (no negative authorization) either explicitly specified or obtained by inheritance. After his/her first log-in, the ontology management system after verifying the credentials of the concerned user will create the relevant view and will transfer it to the client system of the user. Since the policy set for the concerned user and hence his/her set of permitted concepts will remain unchanged till there is any change in the user's credential, the concerned user will access only his/her view in his/her

own system without accessing the original ontology. This approach will avoid the undecidability problem, since the user will have access to permitted set of concepts only that have positive authorization and will appear to be a open system to the user. However, if a user updates his/her credentials, his/her policy set will also change and thus the permitted set of concepts. Ontology management system will then create a new ontology view for the concerned user.
This approach may create a confusion that creation of user specific views, instead of application specific views as done in standard databases, would give rise to too many ontology views. The research group engaged in this effort have found that most of the users for a digital library are making remote accesses. The idea of user specific view is to transfer a particular view to the client end. It will be created only once and would be transferred to the client machine so that further queries from the same client can be made on the view at the client end itself. The view will be changed only if a user earns different set of authorizations or there is a change in the ontology structure. Depending on the environment, a user specific view at the client end can as well be considered as a site specific serving a group of users. So, the entire procedure is equally applicable for a group of users as well. The access policies will then be applicable to the entire group and all members of the group will inherit them under the assumption that all the members of a user group will have a common credential set. The View-Creation Algorithm is given below:

Algorithm 2 View Creation algorithm

1: Input : Set of Ontology data O and User u
2: Output : User Specific view $\bar{O}$
3: **function** CREATEVIEW(O, u);
4: $\bar{O} = \{\}$; ▷ Initialization of Empty Ontology
5: **for** $i = 0$ to n **do**
6: $color[i] \leftarrow WHITE$; ▷ *color* is the node coloring to differentiate visit. We have use three variation of color i.e. WHITE , Gary and Black.
7: $mCountNode[i] \leftarrow 0$; ▷ $mCountNode$ is Hash list to store Number of visit remaining for a multiple parent Concept.
8: $mStatusNode[i] \leftarrow 0$; ▷ $mStatusNode$ is a Hash List to store the permission status flag.
9: **end for**
10: **repeat**
11: for each i
12: $v \leftarrow O_i$;
13: ONTOLOGYACCESS(v, u); ▷ Initialization of Ontology Access Algorithm.
14: $i++$;
15: **until** $i \preceq n$;
16: **end function**
17: **function** ONTOLOGYACCESS(v, u) ▷ Recursive function of Pre-order Traversal
18: $color[v] \leftarrow GRAY$;
19: $mCount[v] \leftarrow$ GETPARENTS(v);
20: $\alpha \leftarrow$ GETXACMLSERVICE(v, u);
21: $\pi \leftarrow$ GETLISTOFCHILDREN(v);
22: $\delta \leftarrow 0$;
23: $passover \leftarrow 0$
24: **if** $(\pi.size() = \emptyset \bigwedge color = Black)$ **then**
25: REMOVENODE($v, \bar{O}$) ▷ Remove node will remove the node from view.
26: **else if then**
27: $\bar{O} \leftarrow v$;
28: **end if**

```
29:     if α ≠ true then
30:         v ← nodeObfuscationService(v)
31:     end if
32:     for i = 0 to π.size() do
33:         passover ← 0
34:         μ ← π[i];
35:         mCount[μ] ← GETPARENTS(μ);
36:         if mCount[μ] ≻ 1 then
37:             mCount[v] − −;
38:             if α ≠ true ⋀ mStatusNode[μ] ≠ 1 ⋀ mCount[μ] ≠ 0 then
39:                 Remove Link of v ;
40:                 RemoveHasContributed(μ, Ō);
41:             else if α ≠ false then
42:                 passOver ← 1
43:             else if α ≠ true ⋀ mStatusNode[μ] ≠ 0 ⋀ mCount[μ] ⪯ 0 then
44:                 RemoveHasContributed(μ, Ō);
45:                 μ ← nodeObfuscationService(μ)
46:             end if
47:         else if mCount[μ] ⪯ 1 ⋁ passOver ≠ 0 then
48:             if α ≠ true then
49:                 δ ← 1;
50:                 if getListOfChildren(μ) = ∅ then
51:                     color[μ] ← Black;
52:                 else if getListOfChildren(μ) ≠ ∅ then
53:                     Ψ ← getListOfChildren(μ)
54:                     n = Ψ.size()
55:                     flag ← 0;
56:                     repeat
57:                         p ← Ψ[i]
58:                         if color[p] = Gray then
59:                             flag ← 1;
60:                         end if
61:                         i + +;
62:                     until (flag! = 1 | n = 0)
63:                     if flag = 0 then
64:                         color[μ] ← Black;
65:                     end if
66:                     OntologyAccess(μ, u)
67:                 end if
68:             end if
69:         end if
70:     end for
71: end function
```

Before explaining the view creation algorithm, some basic operations done on the ontology for creation of such view need to be explained. These operations are:

(1) Branch Removal: Whenever a user tries to access a node/concept, security policy server returns the corresponding authorization of that user for that particular node/concept. Repeating the discussion made earlier, it is important to note that the authors have considered that a library would be accessed primarily in an open environment. In other words, unless

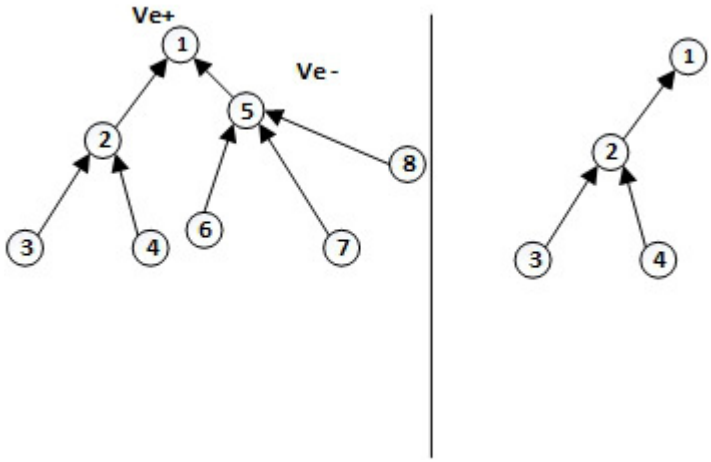

Figure 12. Sub-Graph Removal

otherwise blocked, i.e. denied access, a user will have positive authorization, by default, to all concepts with their documents as soon as he/she logs in successfully. A user may get a negative authorization to a node/concept by analyzing his/her credentials and assigning explicit denial to some nodes/concepts, or by inheritance from parent nodes. However, if a user has negative authorization (implicit or explicit) to a concept node, all nodes inferable from that node will also inherit that negative authorization. The branch removal algorithm during user-specific view creation will remove those nodes of the ontology that have negative authorization for the concerned user. The associated links are also removed. Thus, the sub-graph of the ontological structure that the concerned user is not supposed to access for negative authorization would not be present in his/her view.
Figure.0**??** explains the situation. Left side of the figure shows the ontological structure before branch removal. In Figure.0**??**, node-1 has positive authorization. Hence all nodes below the node-1 (i.e. inferable from node-1) will inherit the same positive authorization. Now, node-5 has an explicit negative authorization. So once again, nodes inferable from node-5, i.e. nodes 6, 7 and 8 will inherit the same negative authorization. Branch removal algorithm will remove these nodes and associated links to generate user-specific view, as shown in the right-hand portion of the Figure.0**??**. As mentioned earlier, +ve authorization is available by default, hence specifying -ve authorization should have been sufficient. However, in Figure.0**??**, +ve authorization has been shown for the purpose of clarity only. It will be shown later that even explicit +ve authorization will be required in some other operation.

(2) Concept Obfuscation: No doubt, branch removal algorithm retains only those concepts/nodes in the user specific view for which the concerned user has positive authorization. As a result, the undecidability problem is avoided. However, the same algorithm may leave the view as a collection of disconnected sub-graphs. Node obfuscation is required to solve the problem.
Figure.13 explains the situation. Left side of the figure shows the ontological structure before node obfuscation. In Figure.13, node-1 has positive authorization. Hence all nodes below the node-1 (i.e. inferable from node-1) will inherit the same positive authorization. Now, node-5 has an explicit negative authorization. So once again, nodes inferable from node-5, i.e. nodes 6, 7 and 8 should inherit the same negative authorization. Now nodes 6 and 8 are again assigned with explicit positive authorization. So branch removal algorithm would remove nodes 5 and 7 with associated links and leave nodes 6 and 8 as isolated nodes away from the original ontological structure. To avoid such a situation, node-5 is retained in the view to connect nodes 6 and 8. However, since the concerned user has a negative authorization for concept represented by node-5, it is obfuscated. Right side of the Figure.13 shows the user-

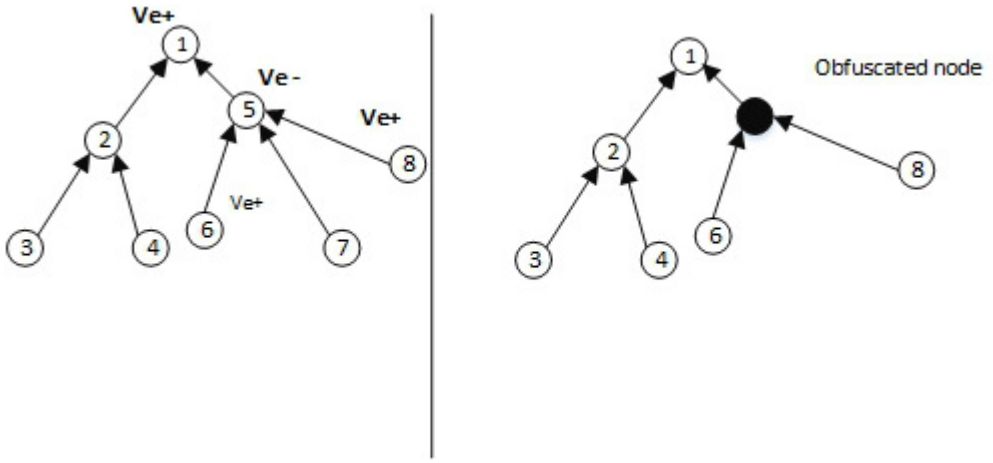

Figure 13. Node Obfuscation

specific view after node obfuscation. While an user query traverses the ontological structure, it identifies the existence of node-5 but doesn't get its identity or cannot access any document under it. This problem may occur for controlled access to any ontological structure and it is quite common for semantic web applications [Qin and Atluri 2010][Kaushik et al. 2005]. The authors of this paper are not sure whether such situation would occur in a Digital Library application, never the less decided to extend the facility. However, node obfuscation would occur only if explicit +ve authorization is allowed which may create one or more nodes with -ve authorization in between two positively authorized nodes.

(3) Partial Access: This situation happens when a child node/concept has multiple parents and the child node is partially inferable from all of them. Referring back to Figure.1, concept *Database* has three parent concepts, CS, GIS and BIO. So, *Database* is partially inferable from all three of them. Now, if from the credentials of a user, administrator infers that he/she can access only the CS concept as the parent of *Database* concept, then the system would impose implicit negative authorization on concept GIS and BIO. Now the concerned user for its positive authorization to CS and inherited positive authorization to *Database*, will get access to document classes 1, 4 and 5 only, as shown in Figure.3 and associated Table I. So, the branch removal algorithm will remove the GIS and BIO nodes and their children nodes other then *Database* and the nodes directly inferable from *Database* concept. Continuing the example, document class nodes of *Database* concept not relevant for CS concept will also be removed, keeping only the CS related document classes (i.e. document classes 1, 4 and 5). As a result of this branch removal, in the resultant view, the children of *Database* concept will only have positive authorization inherited from *Database* which would also have only positive authorization inherited from CS.
Figure.14 shows an example ontology where for a particular user, +ve and -ve authorizations are clearly indicated in Figure.(a) on the left side. Figure.(b) on the right side shows the created view along with the obfuscated nodes.

The important steps of the view creation algorithm are listed below.

Input: . Digital Library Ontology O, credential and access rights of a user u.

Output: . User's specific Ontology view.

(1) A node will be added to user's view once it is visited.
(2) If a node is not a leaf node, algorithm will start traversing to it's child. This algorithm follows pre-order traversal rules.

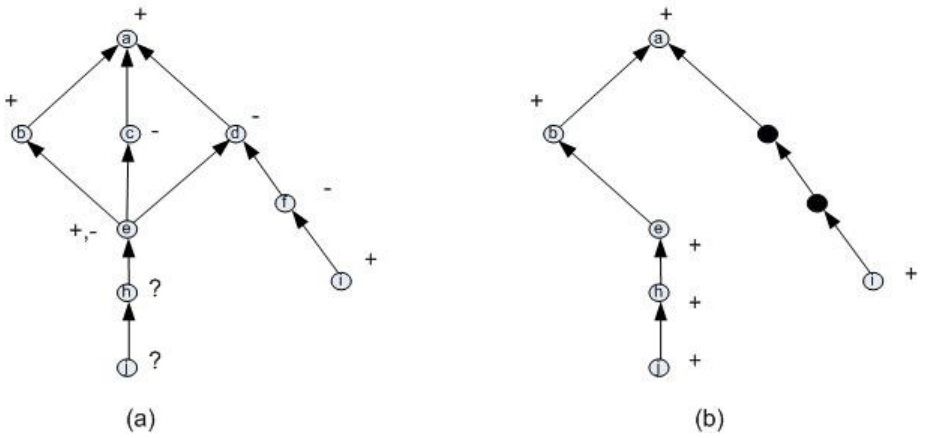

Figure 14. Creation of View

(3) If a leaf node which has single parent and does not have authorization for the user u, the node will be deleted from user's view with all its document classes.

(4) If a node has single parent, has negative authorization for user u and it is not a leaf node, then the node will be obfuscated. Its document classes will not be available to the concerned user.

(5) If a node has single parent and has positive permission for user u, all its document classes will be added to user's view.

(6) If a node C_i has multiple parents, then the node will be traversed completely to reach its child nodes only after visiting C_i from all its parent nodes to find the inherited authorizations from all its parents.

(7) If a node C_i has multiple parents and if anyone of the parent has positive authorization, node C_i will inherit positive authorization which will also be propagated to the child nodes of C_i. Other parent nodes of C_i having negative authorization will be deleted from the view.

(8) If a node C_i has multiple parents and all parent nodes have negative authorizations, then C_i will be obfuscated and traversed. Similarly the negative permission will be propagated to its child nodes unless there is any explicit positive authorization.

(9) Starting from the first node with negative authorization, all nodes inferrable from it will be obfuscated till a leaf node is encountered. Then the entire path with negative authorizations starting from the first node mentioned to the leaf node will be deleted from the view. However, if any node in between including the leaf has a explicit positive authorization, the intermediate nodes with negative authorizations will be obfuscated.

(10) The algorithm will continue till all the nodes of the ontology are visited.

Process of view creation has made the following assumptions:

(1) There are sufficient number of users in the system so that any concept on the ontology is accessed by at least one user.

(2) Any node pair in the ontology is present in at least one view.

The above assumptions give rise to the following properties of the views created:

(1) Completeness: All user specific views taken together represent the entire ontology with all its concepts and inter-concept relationships.

(2) Finiteness: Number of views generated by view creation algorithm is finite. If the ontology has n concepts, number of views can maximum be 2^n. Actual number of views will be much

less since each created view will depend on the connection structure of the ontology hierarchy and all nodes are not reachable/inferable from all other nodes.

(3) Safety: A view created against a given set of authorizations is unique. Credentials of a user give rise to a user specific policy set that in turn decides the authorizations. If the authorizations are different, different sets of concepts will be available with different sets of inter-concept relationships and hence would generate different views. If two views for two users are found to be same, they should belong to same user group. This is a necessary safety property for view creation. If same view can be created by more than one set of authorizations, security policies can be comprised.

5. CONCLUSION AND FUTURE SCOPE

Since the present research effort considers a library ontology, no removal of concept (subject area) has been considered. However, an *alter concept* and/or *add concept* operation can not only add a new concept, it can also change the document classes and redistribute the documents among those classes. So basically, after each *add concept* and/or *alter concept* operation, document classification is done implicitly and hence no special operation like *add classification* or *alter classification* is required. Identifying the document class of any document is done by the algorithm run by the underlying IR engine. Placement of a document to a particular document class is done by a document hashing mechanism against the unique id assigned to each document stored in the library. Such implementation details are not within the scope of this paper. It has already been discussed by the authors in [Dasgupta and Bagchi 2011]. This paper has highlighted the importance of view creation. Transfer of views to users' systems, reduces unnecessary loads to the ontology server and also ensures users' access to permitted set of concepts only. This paper discusses control of access for reading only. Future research effort would be on addition of documents and also updation of existing documents if necessary. This paper has not considered the implementation details which will be covered separately later.

REFERENCES

DDC-2010. Implementing dewey.info as a linked data platform. Dewey Summaries as Linked Data. OCLC Developer Networks.

LCC-1990. *Library of Congress. (1990) LC Classification Outline (6th Ed.)*. Library of Congress.

Bertino, E. and Ferrari, E. 2002. Secure and selective dissemination of xml documents. *ACM Trans. Inf. Syst. Secur. 5*, 290–331.

Carminati, B., Ferrari, E., and Bertino, E. 2005. Securing xml data in third-party distribution systems. In *Proceedings of the 14th ACM international conference on Information and knowledge management*. CIKM '05. ACM, New York, NY, USA, 99–106.

Corradini, A., Montanari, U., Rossi, F., Ehrig, H., Heckel, R., and Löwe, M. 1997. *Algebraic approaches to graph transformation. Part I: basic concepts and double pushout approach*. World Scientific Publishing Co., Inc., River Edge, NJ, USA, 163–245.

Damiani, E., De Capitani di Vimercati, S., Paraboschi, S., and Samarati, P. 2002. A fine-grained access control system for xml documents. *ACM Trans. Inf. Syst. Secur. 5*, 169–202.

Damiani, E., di Vimercati, S. D. C., Paraboschi, S., and Sarnarati, P. 2000. Securing xml documents. In *INTERNATIONAL CONFERENCE ON EXTENDING DATABASE TECHNOLOGY*. Springer, 121–135.

Dasgupta, S. and Bagchi, A. 2011. Controlled access over documents for concepts having multiple parents in a digital library ontology. In *Computer Information Systems Analysis and Technologies*, N. Chaki and A. Cortesi, Eds. Communications in Computer and Information Science, vol. 245. Springer Berlin Heidelberg, 277–285. 10.1007/978-3-642-27245-5_33.

Dasgupta, S. and Bagchi, A. 2012. A graph-based formalism for controlling access to a digital library ontology. In *CISIM*, A. Cortesi, N. Chaki, K. Saeed, and S. T. Wierzchon, Eds. Lecture Notes in Computer Science, vol. 7564. Springer, 111–122.

Farkas, C., Gowadia, V., Jain, A., and Roy, D. 2006. From xml to rdf: Syntax, semantics, security, and integrity (invited paper). In *Security Management, Integrity, and Internal Control in Information Systems*, P. Dowland, S. Furnell, B. Thuraisingham, and X. Wang, Eds. IFIP International Federation for Information Processing, vol. 193. Springer Boston, 41–55. 10.1007/0-387-31167-X_3.

Gabillon, A. 2005. A formal access control model for xml databases. In *Secure Data Management*, W. Jonker and M. Petkovic, Eds. Lecture Notes in Computer Science, vol. 3674. Springer Berlin / Heidelberg, 86–103. 10.1007/11552338_7.

Gonçalves, M. A., Fox, E. A., and Watson, L. T. 2008. Towards a digital library theory: a formal digital library ontology. *Int. J. Digit. Libr. 8*, 91–114.

Kashyap, V. and Sheth, A. 1996. Semantic and schematic similarities between database objects: a context-based approach. *The VLDB Journal 5*, 276–304.

Kaushik, S., Wijesekera, D., and Ammann, P. 2005. Policy-based dissemination of partial web-ontologies. In *Proceedings of the 2nd ACM Workshop On Secure Web Services, SWS 2005, Fairfax, VA, USA, November 11, 2005*, E. Damiani and H. Maruyama, Eds. ACM, 43–52.

Koch, M., Mancini, L., and Parisi-Presicce, F. 2005. Graph-based specification of access control policies. *Journal of Computer and System Sciences 71,* 1, 1 – 33.

Noy, N. and Musen, M. 2004. Specifying ontology views by traversal. In *The Semantic Web ISWC 2004*, S. McIlraith, D. Plexousakis, and F. Harmelen, Eds. Lecture Notes in Computer Science, vol. 3298. Springer Berlin Heidelberg, 713–725.

Ouksel, A. M. and Ahmed, I. 1999. Ontologies are not the panacea in data integration: A flexible coordinator to mediate context construction. *Distributed and Parallel Databases 7*, 7–35. 10.1023/A:1008626109650.

Qin, L. and Atluri, V. 2010. Semantics aware security policy specification for the semantic web data. *Int. J. Inf. Comput. Secur. 4*, 52–75.

Saeed, H. and Chaudhry, A. S. 2002. Using dewey decimal classification scheme (DDC) for building taxonomies for knowledge organisation. *Journal of Documentation 58,* 5, 578–583.

Seidenberg, J. and Rector, A. 2006. Web ontology segmentation: analysis, classification and use. In *Proceedings of the 15th international conference on World Wide Web*. WWW '06. ACM, New York, NY, USA, 13–22.

Subhasis Dasgupta is a doctoral student working at Electronics and Communication Sciences Unit (ECSU), Indian Statistical Institute Kolkata. Before joining Indian Statistical Institute, he did his Master of Computer Science and Engineering from Jadavpur University, India. He has around 5 years industry experience over various companies like Connectiva Systems , Ayata (Formally DataInforcom) and Kaavo. His research interest includes Access Control , Ontology and Semantic Web and Digital Library.

Aditya Bagchi got his Ph.D. in Engineering from Jadavpur University, India. After serving in various industries in India and USA, he joined the Indian Statistical Institute where he is now a professor in the Electronics & Communication Sciences Unit. Prof. Bagchi's research interest covers Data modeling for large graphs, Social and Biological Network in particular, Development of data mining algorithms, association and dissociation rules in particular and Design of access control models for different application areas. In his areas of research, Prof. Bagchi has published many papers in International journals, edited volumes and peer-reviewed conferences. Prof. Bagchi has also delivered invited lectures and tutorials in many universities, research labs, workshops and conferences in India, Europe and USA. He is a member of Computer Society of India, Cryptology Research Society of India, Society for Research in Information Security and Privacy, ACM and IEEE. Prof. Bagchi is also serving as advisor to many Govt. Departments and Projects in India particularly for E-Governance and Data Security related issues.