ISSN: 2277-9655
(I2OR), Publication Impact Factor: 3.785

IJESRT

INTERNATIONAL JOURNAL OF ENGINEERING SCIENCES & RESEARCH TECHNOLOGY

EMMO MODEL FOR E-LEARNING

Bineeta Kumari Gupta
Software Deeveloper 2, Oracle India Pvt.Ltd

ABSTRACT

Online learning is becoming a revolutionary force for improvement in education standards and quality. The applications include digital libraries (text documents, images, sound, video), manufacturing and retailing, art and entertainment, journalism and so on.. Normal databases are incapable of handling such wide range and huge amount of data. So we need database to support storage, indexing, retrieval of huge and wide variety of data. This paper present a different way of storing multimedia data in order facilitate easy indexing and retrieval- EMMO. The motivation for the development of the Emmo model is the desire to realize multimedia content sharing and collaborative applications on the basis of semantically modeled content but to avoid the limitations and difficulties of current semantic modeling approaches implied by their isolated treatment of media, semantic description, and content functionality. ProxyMity Web Publish tool allow the user to create a composite lecture by retrieving Lecture videos and Presentation Slides from the database. This lecture and slides are bind together and displayed to the user in the form of HTML browser application. By using EMMO Model, the media aspects, semantic aspects and functional aspects of Lecture video, Presentation slides and any multi-media data are stored in the database. This provides the efficient storage of content in the database, supports content-based queries over multimedia objects. It is an attempt to create software for integrating different multimedia data with respect to the users' necessity.

KEYWORDS: ProxyMITY, Query optimization, indexing, ontology, EMMO Nodes, B Trees, Lucene, Optical Character Recognition

INTRODUCTION

It is difficult for a large number of students, teachers and professionals to meet the challenges of the fast developing technology. Scarcity of adequate and competent guidance, unavailability of study material, constrained personal circumstances and rigid education system are the few causes for this result. There is a need for an easy access to resources, which may be the key factor of encouraging more learning. With the use of e-learning the above mentioned limitations can be overcome by allowing access to distant learning material thereby making it possible to sync one's learning with rest of the world. Search engines based on Natural Language Processing help us to get to the links, which are more close to what we, demand. But the results shown by search engines and the relevance to what we actually want is still not very close. The Internet service providers are missing out a few basic but important learning factors, such as learning media semantics, learning personalization and re-usability. Much effort has been put into the technical reuse of electronically based teaching materials and in particular creating or re-using Learning Objects. These are self contained units that are properly tagged with keywords, or other metadata, and often stored in an XML file format. Creating a course requires putting together a sequence of learning objects. There are proprietary and open, non-commercial and commercial, peer-reviewed repositories of learning objects. E-learning material is subjected to continuous modifications and revision. We need to maintain all semantic information related to multimedia learning content, user personalization, and substitutes for topics. Searching multimedia content, particularly audio-visual data is still a challenging task. Data extractors from such data based on speech recognition are used but they do not provide accurate results and the relevance of this data to the actual data is considerably low. Another important factor for teaching is the learner's pace and his/her level of understanding on the particular topic. For example a learner may not be able to grasp some difficult part of the live lecture given by his/her professor. But since rest of the class understood the concept well the professor goes further to teach more advanced topics. The learner who could not grasp the current topic would not be able to understand the further course. E-learning reduces the human intervention in the

process of learning, and because of this, special attention needs to be given to the level of difficulty and the quality of subject matter provided to the learner, implying e-learning should be user centric. The content of the lecture should be modified such that it is easy to understand by different types of learners. The multimedia database systems are to be used when it is required to administrate huge amounts of multimedia data objects of different types of media data (optical storage, video tapes, audio records, etc.) so that they can be used (that is, efficiently accessed and searched)for as many applications as needed. The Objects of Multimedia Data are: text, images, graphics, sound recordings and video. The model for Enhanced Multimedia Meta Objects (EMMOs) was developed from the intuition that in a knowledge and content sharing economy, different applications would need to adhere to open standards with respect to the containers that knowledge and media objects are delivered in. The idea was to standardize a number of interfaces that such container objects would require. Examples are interfaces for querying the ontology, for accessing digital rights information, and for adding further knowledge to an existing EMMO, using different authoring tools. The EMMO model can be regarded as an enrichment layer on top of standard database technology. The fundamental idea underlying the concept of Emmo is that an Emmo is an object unifying three different aspects of multimedia content, namely the media aspect, the semantic aspect, and the functional aspect.

PROXIMITY MODEL

Proximity is intended to be an open source e learning tool that helps create dynamic rich media lectures. It allows sharing of prepared lectures to others through web, through intranet, or through a third party e-service provider. As the name suggests, "ProxyMITY" serves as a "proxy", i.e. it provides virtual closeness (proximity) without the actual classroom. It is a tool, that helps teachers create dynamic, rich-media lectures: ones that go beyond using simple text and images. It provides easy incorporation of audio, video, and still images with the lecture presentation slides, to create dynamic, distributable, rich-media lectures.

EMMO

Enhanced Multimedia Meta Object is a multimedia content object which comprises three basic units of information described below. It also supports the "Version" Aspect of each of the multimedia object.

1. **Media Aspects:**

Media Aspect of EMMO involves, the actual multi-media content, which may be in the form of either Video or Images or presentation files etc. The *media aspect* describes that an EMMO aggregates the media objects of which the multimedia content consists.

2. **Semantic Aspects:**

The *semantic aspect* reflects that an EMMO further encapsulates semantic associations between its contained media objects by means of a graph-based model similar to conceptual graphs. Hence, an EMMO constitutes a unit of expert knowledge about multimedia content.

3. **Functional Aspects:**

Functional aspect, gives a list of domain-specific actions involved with each different type of MO. The functional aspect expresses that an EMMO offers operations for dealing with its content, which can be invoked by applications.

4. **Version-ning:**

EMMO allows us to keep track of versions of each of the MO created. An EMMO can be ***serialized*** into a bundle that completely encompasses all Multimedia Database three aspects, and is thus ***transferable*** in its entirety between different EMMO providers, including its contained media objects, semantic associations between these objects, and f unctionality. Moreover, ***versioning support*** has been a central design objective: all the constituents of an EMMO can be versioned, thereby paving the way for the distributed, ***collaborative construction*** of EMMOs.

EMMO ENTITIES

Each Emmo object has a distinct set of four entities to completely define it

- Logical media part (Γ)
- Ontological Objects (Θ)
- Associations (A)
- EMMO Nodes (Σ)

Each entity Ω is characterized by thirteen different properties.

Ω= (ow, nw, kw, sw, tw, Tw, Aw, Cw, Nw, Pw, Sw, Fw, Ow) where,

Ω - An entity, thirteen tuple
Ow - Global and unique Identifier
nw - Entity name
kw - Kind of entity kw are "lmp", "ont", "asso", "emmo"
sw -Source entity of an association
tw - Target entity of an association
Tw - Type of the entity, giving ontology information about the object
Aw - List of application dependent attributes
Cw - Connectors, for the involved physical media

Nw - List of all the entities in the current EMMO instance
Pw - Preceding version
Sw - Successor version
Fw - set of more features related to the current EMMO, example- time stamp etc.
Ow - Operations permitted for the current EMMO

Logical Media Part (Γ):
These represent the Media aspect of EMMO. Connectors help to model the media part at logical level, along with the connection to its physical media content representing the object.
Ontological Objects (Θ): This helps maintain the semantics behind each MO node and its entities, thus catering to the Semantic aspect of EMMO model. These entities represent the concept ofentity.
Associations(A): These entities maintain the binary semantic relation between any two other entities, again catering to the semantic aspect.
EMMO Nodes(Σ): EMMO model is extensive graphical structure of object instances. Thus a single EMMO object may have its entities associated to yet another EMMO object. All the integrated in-formation is in the form of root node, comprising of more EMMO objects, in a hierarchical organization. EMMO objects might be independent i.e. in no association with all other objects in the same level. Sometimes they may even have sister object in hierarchically lower level of the same EMMO. We are using the EMMO's concept in representing the data which consists of some videos in an efficient manner without any loss of information. As the definition of EMMO, it defines the information in the form of objects and entities. The data is stored using different tables and relations.

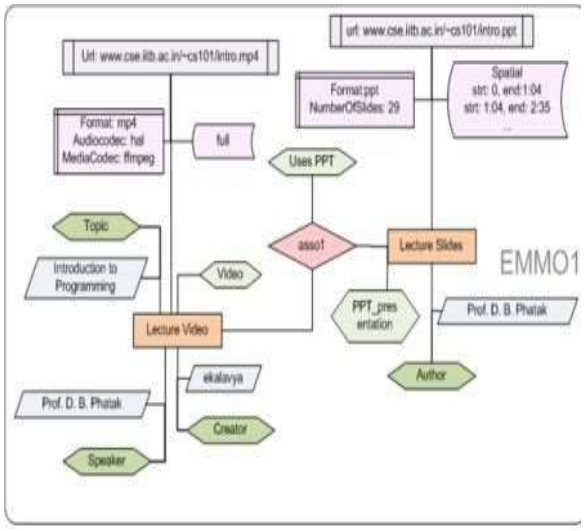

FIG: EMMO Model for Lecture CS101

The different tables used in the EMMO representation are as under:
1. **Entity :** Each entity *w* has a *global* and *unique object identifier (OID) ow* represented as universal unique identifier (UUID), which enables the unique identification of entities in distributed scenarios.
As UUIDs are not really useful for humans, each entity *w* has also a human readable *name* expressed as string value. For classifying whether an entity *w* is a logical media part, an ontology object, an association, or an EMMO, its *kind* is specified accordingly.
2.**Kind:** Kind table is used to classify the entity type whether it is lmp, ontology object, an association, or an EMMO.
3. **Attribute:** Each entity possesses an arbitrary number of application-dependent *attributes*. Attributes are represented as attribute-value pairs with the attribute name being a concept of domain ontology.
4. **Type:** It stores the ontology_id and type_id. Each entity *w* is described by a set of *types Tw*, i.e. a set of ontology objects, enabling the classification by concepts taken from domain ontology.
5. **Connector:** The *connectors Cw* establish a connection to the physical media data of a *logical media part*. Each connector consists of a *media profile*, which describes the storage location by either embedding the *raw media data* or by referencing the media data via a *URI*, and a *media selector*, which provides means to address selected parts of the media object.
6. **MediaSelector_kind:** MediaSelector_kind stores the information that whether the given kind is spatial, textual, temporal or full part.
7. **MediaSelector_parameter:** Media Selector parameter table consists of media selector id, its parameter and parameter value. The parameter value consists of the duration of each logical media part.
8. **Media profile: (mediaprofile):** MediaProfile table consists of mediaprofile_id and media instance. Media profile is used to store the path of the stored object.
9. **Metadata:** Meta table consists of mediaprofile_id, metadata and metadata_value. Meta table consists of metadata and metadata value such as format, size of the entity.
10. **Association:** Association table consists of uuid, source_id and target_id. By specifying exactly one *source* and *target entity sw* and *tw*, an *association* establishes a directed binary relationship between those entities. Associations describe binary directed semantic relationships between entities. Thus, they contribute to the *semantic aspect* of multimedia content. By being modeled as entities, associations

can take part in other associations, and thus enable the *reification* of associations in the EMMO model.
11. **EMMO operator: (operator):** EMMO operation table consists of four attributes uuid, designator, operator_function and operator_description. An *EMMO* offers *operations Ow*, which can be invoked by external applications. The implementation of an operation is described by a mathematical *function*.
12. **Operation table:** The Operation table consists of operator_id and emmo_id. The operation can be implemented on the emmo.
13. **Nodes:** Node table consists of emmo_id, nodes_id. An *EMMO* constitutes a container of all entities specified in the set *nodes*.
14. **Successor table: (successor) and Predecessor table: (predecessor)** Each successor and predecessor table consists of emmo_id in both the tables successor_id, predecessor_id respectively. For providing versioning support, a set of *preceding versions Pw* and *succeeding versions Sw* can be assigned to each entity *w*. Each version of *w* is again an entity of the same kind *kw*. By also treating an entity's versions as entities, different versions of an entity can be interrelated just like any other entities, thus allowing to establish relationships between entity versions.
14. **Features:** Features table consists of uuid, feature and feature_value. As it might be necessary in an implementation of the model to augment an entity *w* with further low-level data, such as time stamps or status information, in a flexible, ad-hoc manner, a set of *features Fw*, represented as feature-value pairs, can be attached to the entity. In contrast to attributes, feature names are not ontology objects but simple strings.

EMMO Algebra : Extraction Operator, Navigation operator, Constructors, Selection predicates, Join operators.

Lucene

Lucene has been chosen for performing full text search queries on the database. In our system keyword search on the database takes a few minutes to return results. This is the reason we choose lucene to handle the keyword based searches. Lucene is a high-performance, full-featured text search engine library written entirely in Java. It is a technology suitable for nearly any application that requires full-text search, especially cross-platform. Lucene is a text search engine created in Java. Lucene is a popular text search engine library written entirely in Java. It is developed from the Apache Software Foundation. Lucene offers a powerful feature through a simple API. It can be used to index word documents, HTML and XML files. Lucene is a text search engine created in Java. Lucene is a popular text search engine library written entirely in Java. It is developed from the Apache Software Foundation. Lucene offers a powerful feature through a simple API. It can be used to index word documents, HTML and XML files.

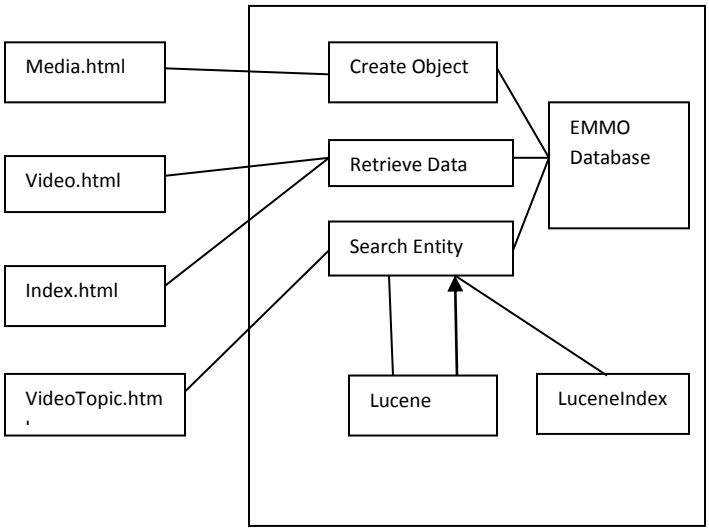

FIG: Main Diagram from ProxyMITY Model

EMMO IMPLEMENTATION FOR PROXYMITY MODEL

EMMO, being a hierarchical graph structure, while implementing it on the lines of Relational database model, we come across few constraints, as listed below:

1. EMMOs thirteen tuple structure : EMMO, is represented in the form of a 13-tuple set. Having this 13 tuple set as a record in relational database is wastage of memory. This occurs due to the fact that not all attributes involved in this 13 tuple are utilized by all entities. Consider the basic one, Ontology part, which only has the type, kinds and name defined. Along with no description as to which other EMMO entity it does belong to. An EMMO entity, requires a set of other entitys uuid/reference.

2. Multivalued Attributes : The attribute Aw, a list of all application dependent attribute and Fw, a set of features related to given entity, are both multivalued attributes. These values are not only used by specific application requirement, but also provide with the ontology information pertaining to given entity, so are necessary for information specific queries. Further, Operations, Media Selectors, EMMO entities are all multivalued attributes.

3. Connectors : They are pairs of MediaProfiles and MediaSelectors, specific to a given Logical media part entity. Media profiles are allowed to be shared

among other logical media part entities. The relation between a video and the associated slide is created through associations. Any media whether it be a video or a slide image is represented using a lmp(logical media profile) or a connector which stores meta information of the media and the location of the file. The connector also contains the attributes of how the association is made with another lmp.

SEARCHING IN THE DATABASE

Here we describe how search and data retrieval has been implemented in the backend of ProxyMITY. We also describe the challenges that we faced on performing search on a database with a million rows.
Building of Sample Database : For being able to perform search on the database first we need to insert data into the database. We had compiled data for 6 courses in XML format. This data was converted into the emmo format and then inserted in the database. A Java code was written for conversion of the xml data into the emmo structure and then dumping it into the database.

LUCENE FOR TEXTUAL SEARCH IN PROXYMITY

We utilized lucene to index all the tables containing text content in our database. The following tables were indexed using lucene –
• entitymain : This table is the core table of emmo database as every entity present in the database has an entry in it. It has a 'varchar' field which stores the name of the field. We want to perform the full text search on this field. As this field contains the name of the entity which could be a lecture name, topic name, course name, professor name etc. So we want to store the 'varchar' field and the associated 'uuid' field in the lucene index thereby making it fast. In our experiments the full text search performed by the MySQL database takes 117 seconds.
• mediaprofile : This table contains the location of the media file. If we are searching for a particular media file by its name say we are looking for the video on lectures than this table could come out to be helpful. The results of the search performed by lucene are utilized to find out more information about the entities. For eg. in the 'entitymain' table we have indexed 'entityname' and 'uuid' of the entity. So through the lucene based search we retrieve the uuid's of the entities and then use the uuid to gather more information about the entity from the database.

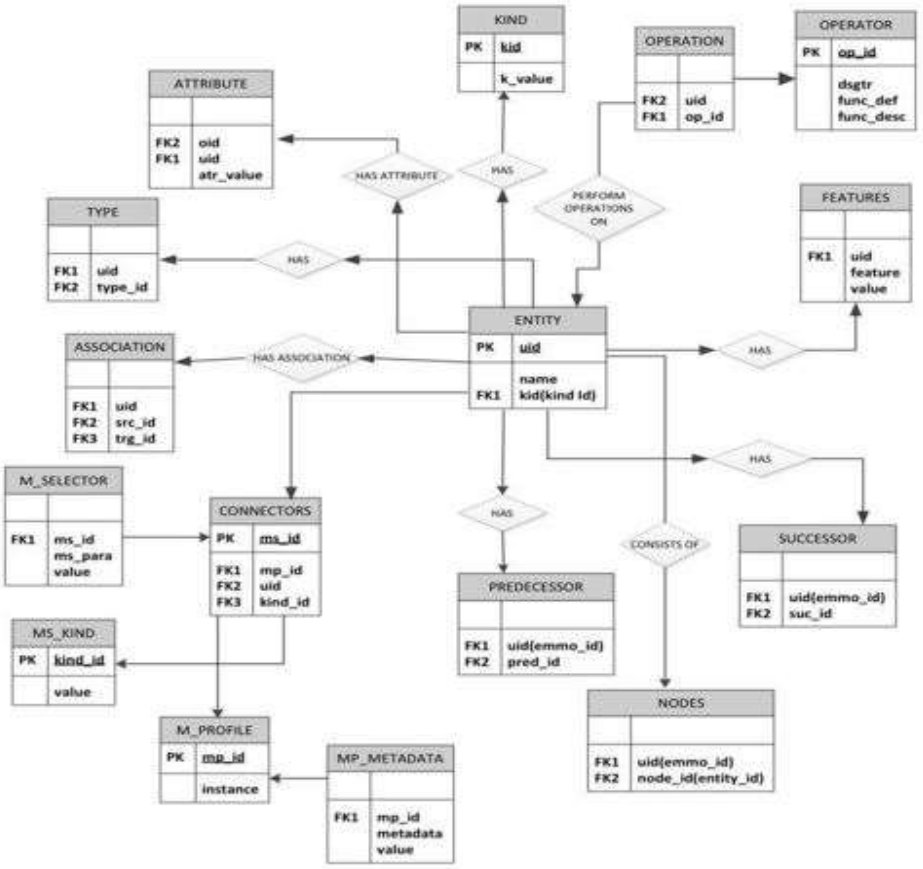

FIG: E-R Representation of EMMO

ISSN: 2277-9655
(I2OR), Publication Impact Factor: 3.785

INDEXING ARCHITECTURE IN DATABASE

Non-clustered The data is present in arbitrary order, but the logical ordering is specified by the index. The data rows may be spread throughout the table regardless of the value of the indexed column or expression. The non-clustered index tree contains the index keys in sorted order, with the leaf level of the index containing the pointer to the record (page and the row number in the data page in page-organized engines; row offset in file-organized engines). In a non-clustered index:

• The physical order of the rows is not the same as the index order.

• Typically created on non-primary key columns used in JOIN, WHERE, and ORDER BY clauses. There can be more than one non-clustered index on a database table. Clustered Clustering alters the data block into a certain distinct order to match the index, resulting in the row data being stored in order. Therefore, only one clustered index can be created on a given database table. Clustered indices can greatly increase overall speed of retrieval, but usually only where the data is accessed sequentially in the same or reverse order of the clustered index, or when a range of items is selected.

B TREES

In the current version of MySQL that we are using B Tree indexing is the standard indexing algorithm used for indexing of the tables. B-tree is a tree data structure that keeps data sorted and allows searches, sequential access, insertions, and deletions in logarithmic time. The B-tree is a generalization of a binary search tree in that a node can have more than two children. Unlike self-balancing binary search trees, theB-tree is optimized for systems that read and write large blocks of data. In B-trees, internal (non-leaf) nodes can have a variable number of child nodes within some pre-de_ned range. When data are inserted or removed from a node, its number of child nodes changes. In order to maintain the pre-de_ned range, internal nodes may be joined or split. Because a range of child nodes is permitted, B-trees do not need re-balancing as frequently as other self-balancing search trees, but may waste some space, since nodes are not entirely full. The lower and upper bounds on the number of child nodes are typically _xed for a particular implementation. Each internal node of a B-tree will contain a number of keys. The keys act as separation values which divide its subtrees. For example, if an internal node has 3 child nodes (or subtrees) then it must have 2 keys: a1 and a2. All values in the leftmost subtree will be less than a1, all values in the middle subtree will be between a1 and a2, and all values in the rightmost subtree will be greater than a2. A B-tree is kept balanced by requiring that all leaf nodes be at the same depth. This depth will increase slowly as elements are added to the tree, but an increase in the overall depth is infrequent, and results in all leaf nodes being one more node farther away from the root.

How does a b-tree help with database access?

Most indexes are too large to fit into memory, which means that they are going to be stored on disk. Since I/O is usually the most expensive thing you can do in a computer system, these indexes need to be stored in an I/O e_cient way. A b-tree is a good way to do this. If we make the nodes the size of a physical I/O block, it would take one I/O to move to a lower depth in the tree. A B-Tree utilizes the following ideas to perform faster indexing of data -

A] keeps keys in sorted order for sequential traversing

B] uses a hierarchical index to minimize the number of disk reads

C]uses partially full blocks to speed insertions and deletions

D]keeps the index balanced with an elegant recursive algorithm

INDEXING IN EMMO DATABASE

Most of the tables in the emmo database have millions of entries, especially the table ‘entitymain‘, ‘nodes‘ and ‘entitytype‘. So we need to create an index for these tables so that searching becomes easier.

• entitymain is the most frequently accessed table in the emmo database. It contains the identities of all entities present in the emmo database. Any new entry that is created whether its a lecture, a professor an lmp, or a an association first needs to be defined in the entitymain table. It has 3 columns in it - uuid, entity kid,entityname. The best index for this table would be a primary index on uuid. As uuid is the parameter which is utilized for searches most of the times so indexing the table and data storage in it on the basis of uuid would be most search efficient. Also creating an index on ‘entityname‘ would not serve much purpose as databases themselves are not very efficient in string searches. We would be using lucene for performing searches on the text portions.

• nodes is the table utilized for finding the hierarchy structure in the database. The entries are stored in the format - parent,child. This table is indexed by creating a primary index on the parent i.e. on emmo id. It makes the searches fast as in this way all the children associated with 1 parent can be found by just finding one particular entry in a index. Also as in the current database design a child does not have

multiple parents and in current queries we do not search for parents of multiple entities in a go so a secondary index on 'node id' would be an efficient way of indexing the nodes table.
• entitytype table contains information about the type of entity, whether its a course, a lecture, a professor or a video etc. Again in this we have created a primary index on 'oid'. Because most of the queries fired on the entitytype table are the ones where we are looking for entities of a particular type, say a course, a lecture. Seldom we fire queries . which look for the type of a particular entity, they can be served by creating a secondary index on the 'uuid' column.

QUERY OPTIMISATION

Join Ordering : A good ordering of join operations is important for reducing the size of temporary results, hence most query optimizers pay a lot of attention to join order. The natural join operation is associative

Optimizing 'IN' Subquery in MySQL: The 'IN' subquery is being utilized in our system at multiple points. For instance if I have to find all the possible courses in the database i throw a query like select node id from nodes where emmo id **in {select node id from nodes where emmo id like CSE DEP T} - (1)**=> This query finds all the lectures available under the CSE Dept. i.e. it finds the grandchildren of the provided entity. Now if we closely analyze this statement using the 'EXPLAIN EXTENDED' statement in MySQL we find that the subquery on 'nodes' table becomes a dependent subquery The mySQL system treats a dependent subquery as a 'correlated query'. This correlated evaluation of subquery is not very efficient since the subquery is separately evaluated for each tuple in the outer query. A large number of disk operations result. For every row in the main table i.e. 'nodes' a check is performed using the dependent subquery if it satisfies the 'where' clause or not. This leads to a humongous rise in execution time if the 'nodes' table is huge in size. Now to get around this inefficient query what we can do is move the dependent subquery up to the 'join' and make it a derived subquery. This process of replacing a nested query with a join(possibly with a temporary relation) is called decorrelation . If we rewrite the above provided query as select node id nodes E join **{select node id from nodes where emmo id like CSE DEP T} T on E.emmo id = T.node id - (2)** =>Now in this scenario the subquery gets executed first as it is supposed to be joined with the 'nodes' table. On an approximate analysis the children of a particular entity say CSE DEPT would be much less than the size of the 'nodes' table itself, so the temporary relation that is created of all the children of CSE DEPT is smaller in size and can be stored in memory itself also the join of this relation with the nodes table itself on the specified condition becomes much faster as the 'nodes' table is indexed on emmo id and therefore this new query gives a much quicker performance than the previous one. **NOTE :** Tests were performed on a 2.4 GHz Core i5 Dell Latitude system with 4GB ram.

PROXYMITY WEB INTERFACE

The features provided can be classified into - content creation, display, search and exporting of emmo data.

Content creation-

Instructor Creation : Using this feature we can create a new instructor in our ProxyMITY system. The newly created entity 'Instructor' is assigned the ontology type 'instructor' and its name is stored in the entitymain table. We have specified the 'instructor' as an emmo in our system.
Course Creation : It can be utilized to create a new course. For the creation of a course we need to select a predefined instructor to associate 54 with it. This association between an instructor and the course is stored in the associations table under the association 'instructed by'.
Lecture Authoring : It is utilized for adding lectures to the course. We need to specify here the speaker of the course which can be any one of the instructors stored in the system, we also need to choose the course in which this particular lecture is going to be stored. In the backend the lecture entity is stored as the child entity of the selected course. This parent child relationship is stored in the 'nodes' table.
Viewer : This part of the system displays the lecture content of the selected lecture. There is a topic tree associated with every lecture which is basically topic wise segregation of the lecture video. The basic information about the lecture i.e. the course name, speaker etc. are also displayed.
Search : This interface provides the basic search features on the database. We can type in keywords and the search is performed across all the entities present in the database.

EXPORTING EMMO'S

EMMO's are designed as containers which can be transferred from one system to another. The main idea behind developing ProxyMITY using emmo model is to allow seamless transfer of information. In ProxyMITY a person say a professor or a teaching assistant can create a lecture which can then be transferred to another system say a remote system which does not have Internet connection. The container based structure of EMMO allows export

and import of emmo structures. An EMMO container can completely export EMMOs into bundles encompassing the media, semantic, and functional aspect, as well as the versioning information. As EMMOs are capable of describing quite complex knowledge structures, different export modes reflecting only selected parts of an EMMOs content, are provided.

OBSERVATIONS

To ensure that exporting can be done seamlessly even on a database with large amount of data. We performed a test where we exported 500 courses that were part of a University. Each course had 10-15 lectures in it. Every lecture comprised of 15 emmo's, in which every emmo was basically an association of 2 lmp's one denoting the slide and one denoting the video. So in all there were approximately 2,75,000 entities that needed to be exported. The experiments were performed a macbook pro with 2.4GHz Intel Core 2 Duo Processor, 4GB 1067MHz DDR3 RAM. The exporting of the 500 courses took 8 min 10 sec and it generated a file of size approx 27MB on disk. Also exporting of a single course i.e. approx 550 entities took 10 sec on the same system and generated a file of size 45KB.

META-DATA EXTRACTION TOOLS

Slide Detection: We detect slide transition times by using an algorithm which compares image files generated by ffmpeg and then title extraction from them using an Optical Character Recognition tool tesseract. We extract information from lectures video and presentation slides. Optical Character Recognition is useful for extraction of textual information from a video.

Optical Character Recognition : Optical Character Recognition (OCR) extracts textual information from an image. Our lecture video is divided into individual frames as above which are then fed to OCR to give textual content. We use a well known open source tool Tesseract-ocr which is known for its accuracy in recognition. It takes .tif images as input and produces output text file. It takes an image and saves text present in the image into a separate file. It accepts images which are in TIFF format so we need to convert images in other formats into TIFF. We use the following commands to extract text from JPEG image.

Content Extraction from a PDF file: We convert a pdf file into images corresponding to each slide using PDFBox. PDFBox is a JAVA library which can handle different types of PDF documents including encrypted PDF formats and extracts text and has a command line utility as well to convert PDF to text documents**. Command :** $ java -jar pdfbox-app-1.7.0.jar PDFToImage lecture.pdf This converts all the pdf slides into jpeg images. We then use these images in ProxyMITY. PDFParser object is used to parse an input PDF file. The parsed content is stored in COSDocument object. But this content is not is text format. PDFTextStripper is used to convert this content into a String. Now by using PrintWriter object we output the extracted text to a text file. We also extracted the content of each slide into a single text file. This will be useful to directly get the slide number of the information we are searching for.

INTEGRATING SLIDE TEXT IN EMMO DATABASE

In the above sections we utilized 2 separate techniques - Optical Image Recognition and PDF to text conversion to get the text content from the slides. The basic way of integrating this slide text with the slide in the database is to –

- store the slide content in a text file
- represent the above created text file as an lmp in EMMO database
- associate the slide image with the slide text with an association TextContent Also along-with this we need to index the slide text using lucene so that it could be made searchable. We can create a lucene index where every document represents a text slide entity.

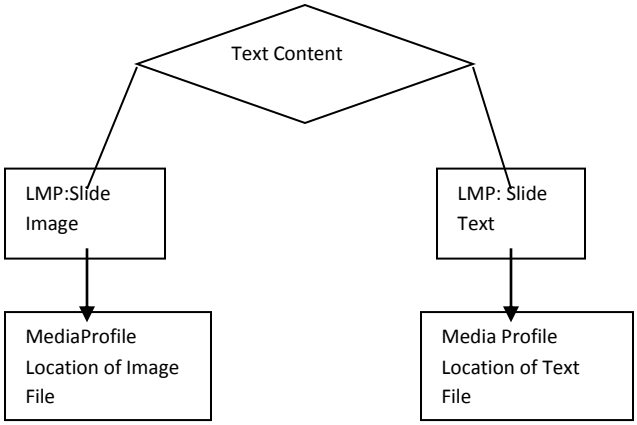

FIG: Respresenting Slide Text in EMMO

FUTURE WORK

Important aspect would be to integrate the video subtitle information in the emmo database, so that even the speech content of the speaker can be searchable thereby providing a much more rich experience to the users. More work is required in direction of interoperability and standards as no. of

multimedia application are increasing tremendously. Also functionality of metadata can be extended by better technique for automatic metadata creation and harvesting metadata created by human[5]. The application can integrate the media player with itself. This will allow the editors to edit the details simultaneously while watching the video. This will decrease the amount of time required to create such files and can also reduce the work load from the editors**.**

CONCLUSION

An EMMO model has been built for the ProxyMITY tool alongwith a backend database design and java code for accessing the database. Now authoring of content in ProxyMITY tool can be done, the new content is going to be directly stored in the database. The search performed over the learning objects returns results quickly in real time thereby giving user the most meaningful index in the video and not forcing him to watch the entire video to reach the point of interest. Also as the system does not have any dependencies except MySQL and is made using Open Source technology so it can be deployed on a server and can become usable for the students and teachers. Extracting contents from the slides has also been discussed thereby providing much more meaning to the content. The use of Lucene for indexing the core table gives us the leeway to put as much content as needed in the EMMO database without worrying about slow search queries.

REFERENCES

1. Dipl. Math Sonja Zillner. A Query Algebra for Ontology-enhanced Management of Multimedia Meta Objects. PhD thesis, University of Vienna, March 2005.
2. Performance and Scalability of EJB Applications - Emmanuel Cecchet , Julie Marguerite , Willy Zwaenepoel.
3. Akhil Deshmukh, Open source implementation of Emmo model, for e-learning content, Dual Degree Project, Stage I report, October 2011
4. Ignatius Periera, Implementation of a structured model (EMMO) for e-learning content and information retrieval, Dual Degree Project, Stage 2 report, June 2012
5. Paolo Boletteri, Favrizio Falchi, Claudio Gennaro and Fausto Rabitti. Automatic meta-data extraction and indexing for reusing e-learning multimedia objects. ACM, September 2007
6. Jane Greenberg: Metadata Extraction and Harvesting: A Comparison of Two Automatic Metadata Generation Applications. Journal of Internet Cataloging, 6(4): 59-82.
7. Chao Boon Teo and Robert Khenge Leng Gay. A knowledge driven model to personalize e-learning. ACM Journal of Educational Resources in Computing, 6(1):15, March 2006
[8] Internship work by Bineeta Kumari Gupta in IIT Bombay 2012- **http://ekalavya.it.iitb.ac.in/summerintern2012/viewProjects.jsp**
8. Dual degree thesis of Anurag Sharma at http://www.it.iitb.ac.in/frg/wiki/images/2/26/Anurag_DDP_Stage2.pdf

AUTHOR BIBLIOGRAPHY

Bineeta Kumari Gupta
Software Developer 2 in Oracle India Pvt.Ltd
Graduate in CS from VNIT,Nagpur.
Research Interests: Database mining, Database management, Algorithm designing, Distributted computer network.