Abstract

User-customized information selection and delivery reduces the complexity of the overwhelming amount of information available to end-users. Our approach employs user profiles, data selection, and presentation facilities to deliver customized audio information to end-users. Specifically, we construct a domain-dependent ontology (a collection of key concepts and their inter-relationships) to enable user-profile construction to support the retrieval of personalized audio information.

In this research, we show how a domain-dependent ontology facilitates the generation of metadata. We demonstrate that ontology provides end-users richer forms of information to query into the system rather than keyword search. We present how this ontology is used to generate information selection requests (database queries in SQL). We develop an efficient algorithm for conjunctive queries in client-server architecture. Finally, we discuss novel optimization techniques that improve query processing performance, utilizing the knowledge associated with the ontology. The approach we have developed is being implemented in the context of the Personal AudioCast project at the USC, Integrated Media Systems Center.

1 Introduction

The huge amount of information available via electronic means can easily overwhelm end-users. Further, the transfer of any irrelevant information over the network to end-users wastes network bandwidth.

---

1 This research has been funded in part by the Integrated Media Systems Center, a National Science Foundation Engineering Research Center with additional support from the Annenberg Center for Communication at the University of Southern California and the California Trade and Commerce Agency.
Therefore, the need for user-customized information selection is clear. The goal in customized selection and delivery is high precision (little irrelevant information) and high recall (not missing relevant information).

Our work on customized information selection and delivery is in the context of an experimental system for a user-customized audio information-on-demand, which we term Personal AudioCast (PAC). The vision here is a facility to select and deliver customized information from a dynamic database, in audio form. While the techniques and mechanisms we devise and implement will be general, the focus of PAC is to demonstrate them in the context of dynamic collections of sports, financial, entertainment, and late-breaking news, as well as public service information. PAC introduces a new level of empowerment for news and information consumers; beyond simply allowing users to select stories according to topics or keywords. Personal AudioCast can build custom ‘broadcasts’ tailored to a user’s interests, communication style, past choices, and even their physical location. And through its auditory delivery system, PAC’s custom presentation of news and information can be a significant benefit to the visually impaired.

Figure 1. PAC Functional Architecture
Fig. 1 presents the functional architecture of the proposed PAC system. A user request is processed by the user interface and dispatched to the PAC selector. The selector constructs a script for a user personalized audio package, employing a profile for the user, and the database. Selections from the PAC database will be assembled and packaged for the user as a continuous audio presentation. The selector then passes the script to the PAC scheduler which dispatches and delivers the packages to the user. Given the above framework for the PAC system, we can identify the key research and engineering problems/tasks that must be addressed:

- **Segmentation**: Audio (broadcast) may consist of multiple news items. Some news items are of interest to the user and some are not. Therefore, we need to identify the boundaries of news items of interest so that the user’s query can directly retrieve these segments. A mechanism is required which segments the audio in a way that will fulfill the above requirements.

- **Metadata acquisition**: To retrieve particular news items, we need to provide descriptions that will be used to match the information provided with user requests. These descriptions are termed *metadata*. The success of retrieval depends on the technique employed for the generation of metadata.

- **Selection**: A mechanism is required which converts user requests into database queries used to retrieve relevant segments of audio. These queries are executed and the results sent to the scheduler. The mechanism should be robust enough so that it can support different types of queries efficiently.

- **Scheduling**: Selected audio segments should be played with minimal delay by utilizing streaming and caching.

- **User profile generation**: To support personalization, a PAC user profile specifies characteristics, preferences, and communication style for tailoring a periodic or on-demand delivery to the end-
user. Thus, user profile information should be obtained by monitoring the end-user, as well as through explicit user input.

In this thesis proposal, we address the research issues related to segmentation, metadata acquisition, selection, and user profile generation. However, we do not present a mechanism for the user profiles needed to make the adaptation toward true user preferences possible via learning. We assume an agent [May94] can be employed to capture user implicit preferences during the presentation of audio. Further, we do not address any research issue related to scheduling. For this, we rely on a commercial product, Java media framework [JMF], which facilitates streaming and caching during presentation of audio.

For the segmentation of broadcast audio a change of speaker and long pauses are used to identify segment boundaries. Further, multiple contiguous segments may form a news item. For metadata acquisition we need to specify the content of media objects. Two main approaches have been employed to this end: fully automated content extraction and manual annotation. Due to the weakness of the content analysis algorithm, especially in audio/speech, we have chosen the latter approach. For example, in the Informedia project, Hauptmann [Hau95] shows that automatic transcription (indexing) of speech is a difficult task because the current speech recognition system has limited vocabulary, at least one order of magnitude smaller than that of the text retrieval system [WS99]. Additionally, environmental noise generates the inevitable speech recognition error. Hence, in the Informedia project information retrieval results in terms of precision and recall are poor after the conversion of speech into text. In other words, the user may receive too much irrelevant data or miss some relevant data, obvious sources of annoyance in the customization domain. Therefore, to allow appropriate selection and presentation we need to explicitly capture the description of audio. The key problem to be solved in this connection is metadata generation. Metadata generation is provided by an ontology. An ontology is a specification of an abstract, simplified view of the world that we
wish to represent for some purpose [Bun77, Gru93]. Therefore, the ontology defines a set of representational
terms (concepts) and the relationships among them to describe a target world. In this thesis proposal, we
construct a domain dependent ontology. In the annotation process particular concepts from the ontology can
be annotated with an arbitrary number of contiguous segments (object). This variable granularity allows users
to add descriptions not only once but incrementally; and more precise access of audio segment(s) can be
achieved with the use of incremental descriptions. To facilitate user requests, we need to store metadata into a
database. We propose a data model that supports the notion of arbitrary attributes, which can be attached for
each object whenever necessary.

For selection, user requests are first associated with concepts of the ontology before the generation of
database query. Next, specific concepts for each of these associated concepts are generated from the ontology
and are expressed in disjunctive form in database query. Further, we also discuss various techniques for
mapping user requests to other queries such as conjunctive query, disjunctive query, and so on. Our proposed
ontology virtually resides on top of the database, enabling users to query the database in a more expressive or
abstract way. In other words, a broader range of queries is covered than could be achieved by simple keyword
search. For the query we have not built anything from scratch. We use the most widely used query language,
SQL. In addition, we extend the query language to support Allen’s temporal operations [AJ83], using user
defined functions (UDF) as an extension of SQL. We present an efficient algorithm for the temporal
operations of conjunctive queries in client-server architecture. We also demonstrate some novel optimization
techniques which rewrites the SQL with the help of knowledge that comes form the ontology, without
penalizing precision and recall. For user profile generation the user may specify concepts by browsing the
ontology through a user interface.
The remainder of this thesis proposal is organized as follows. Section 2 presents related work. Section 3 presents our approaches for addressing some research issues raised in PAC. Section 4 summarizes future work and describes the current implementation status of PAC. Finally, section 5 presents our conclusion and summarizes our contribution.

2 Related Work

To place PAC in the context of current multimedia and personalization domains, we summarize key related efforts. First, we present related projects and show how our project differs from these relevant projects. Next, we demonstrate how our use of the role of ontology differs from the practice of other researchers in the multimedia domain. Finally, we present related audio/video data models, and discuss how our data model differs.

Table 1 presents information on how the Informedia [HW97], VISION [GGP96], MEDUSA [BFJJ+95], and Canvass [AL98] projects differ from our project, PAC in multimedia domain. In the VISION, and MEDUSA projects indexing to facilitate the query relies only on closed caption data. Further, in the Informedia and Canvass projects indexing not only relies on closed caption data but also upon image features, audio transcriptions, and annotated metadata. However, in PAC, we only consider annotated metadata for indexing. For most of the projects similarity measure is based upon a vector space model [SM83] with the exception of VISION and PAC. In VISION, a commercial product, text datablade, is used which provides “similarity” functionality. In PAC, we use an ontology based framework to facilitate the match between user query and the metadata of audio segments. Only PAC addresses personalization. Besides personalization work in the multimedia domain, there are several works related to personalization in the text domain [Lan, CMS, KBA95]. In these efforts, user profiles and objects are commonly represented as keyword vectors using the TFIDF².
In the text domain, the use of ontology is not new; however, researchers also have started to use ontology in the multimedia domain. For example, Chu et al. [CHIT99] use an ontology to describe various spatial relationships applicable to oncological medical images. However, the uses of ontology we present in this thesis proposal go beyond the role of the ontology in the research of Chu et al. This is because their ontology provides conceptual terms used as the description of specific shapes in queries. In contrast, our ontology not only serves to express query, but also provides a basis for metadata acquisition and improved query response time.

Table 1. Comparison of Different Projects

<table>
<thead>
<tr>
<th>Project name</th>
<th>Index depends on</th>
<th>Similarity is Based on</th>
<th>Query by speech</th>
<th>Personalization</th>
</tr>
</thead>
<tbody>
<tr>
<td>Informedia</td>
<td>Image feature, closed caption (audio transcription)</td>
<td>Vector space</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>VISION</td>
<td>Closed caption</td>
<td>Text datablade</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>MEDUSA</td>
<td>Closed caption</td>
<td>Vector space</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Canvass</td>
<td>Closed caption and annotated metadata</td>
<td>Vector space</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>PAC</td>
<td>Annotated metadata</td>
<td>Ontology</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>

Although we use audio, here we show related work in the video domain which is closest to and which complements our approach in the context of data modeling for the facilitation of information selection requests.

One naive approach to annotating audio/video is to divide it into a number of segments and describe every segment independently; this is termed “segmentation”. However, Smith et al. [SP91] oppose this approach due to its inflexibility. Smith et al. propose an alternative approach called “stratification,” in which a model of layered information is used to describe a variable number of contiguous segments. This variable granularity enables users to retrieve segments in a temporally appropriate scale, and makes the annotation process less cumbersome.

2 Term Frequency Inverse Document Frequency [SM83]
Omoto et al. [OT93] propose a schemaless video object data model. In this model, a video frame sequence is modeled as an object with attributes and attribute values to describe its contents. Each object is described by a set of starting and ending frames. In addition, an interval is described by a starting frame and an ending frame. New objects are composed from the existing objects, and some attributes/values are inherited from these existing objects, based on interval inclusion principle. Omoto et al. propose a query language, VideoSQL to retrieve video objects by specifying some attributes and values. However, this query language suffers from a lack of language expression for specifying temporal relations among video objects.

Adali et al. [ACCE96] develop a video data model that exploits spatial data structures (e.g., characters in a movie) rather than temporal objects which is the basis of our data model. Their objects, activities, and roles are identified in the video frames. Each object and event is associated with a set of frame sequences. A simple SQL-like video query language is developed. This query language provides the user more flexibility for controlling presentations.

Hjelsvold et al. [HM94] propose a “generic” video data model and query language for the support structuring, sharing, and reuse of video. Their data model builds on an enhanced-ER data model. Their simple SQL-like query language supports video browsing. In addition, it supports temporal operations. For annotation, they use a stratification approach. Note that their framework assumes fixed categories for annotations such as persons, locations, and events. Hence, their data model follows conventional schema design with fixed attribute structure rather than arbitrary attribute structure.

Gibbs et al. [GBT94] propose an object-oriented model for time-based multimedia data to demonstrate that the database should be involved with capture, presentation, scheduling, and synchronization of media (audio/video). Hence, their data model addresses the complex organization and relationships present in different media. However, their data model fails to discuss different issues related to annotation and query in a particular medium.
We propose an ontology-based model for customized selection and delivery. We demonstrate that for annotation, an ontology provides representational terms for objects in a particular domain. We employ the stratification approach for the annotation of audio objects. For storage of annotated concepts into a database we introduce schema containing values along with self-explanatory tags for each audio object. In other words, our model supports the notion of arbitrary attributes for objects rather than a fixed common schema for all objects as in a normal database schema design. For the query, we have not built anything from scratch. We use the most widely used query language, SQL. In addition, we extend the query language to support Allen’s temporal operations using user defined functions (UDF) as an extension of SQL. We present an efficient algorithm for the temporal operations of conjunctive queries in client-server architecture, and also demonstrate some novel optimization techniques that rewrites the SQL with the help of knowledge that comes form the ontology, without penalizing precision and recall.

The only data models proposed by us and Omoto et al. use hierarchy. In PAC we use an ontology as a hierarchy while Omoto et al. use IS-A. An ontology automatically provides different levels of abstraction for querying the system. This is because the descriptions which are most specific are annotated with audio objects. The user requests described in generalized descriptions can be expressed in the form of specific descriptions by traversing the ontology, and then expressed in SQL. Thus, this top-down approach enables the user to query the system in a more expressive or abstract way. In contrast, the data model of Omoto et al. fails to describe any mechanism which automatically converts generalized description into specialized descriptions using IS-A. Instead, IS-A facilitates the generation of generalized description from specific descriptions of existing objects. In this case, the user creates a new object by explicitly merging two existing objects. Then the description of the new object becomes generalization of these objects’ descriptions. However, there is a possibility that certain objects annotated with specified descriptions will not be retrieved by a corresponding generalized description because the user has not yet chosen to merge these objects. For example, in the Omoto et al. model one video object is annotated with “President: Bill Clinton” and another video object annotated with “President: George Bush.” When the user defines a new object by merging these two video objects the new object is automatically annotated with “President: American Statesman.” This is because in
their IS-A hierarchy “American Statesman” is the generalization of “Bill Clinton” and “George Bush.” The query, specified by “American Statesman,” can retrieve this new video object consisting of the two video objects. However, if these video objects are not merged by the user, the query specified by “American Statesman,” will not succeed in retrieving them at all.

3 Research Approach

In this section, we first define audio objects and propose a model for the database schema design. Second, we describe how the concept of ontology serves as metadata for a particular audio object. Third, we describe how the ontology is used to express user interests. Fourth we present how the ontology is used to generate information selection requests in SQL. Finally, we show how the ontology can be used to reduce the query response time.

3.1 Segmentation of Audio

Since audio is by nature serial, random access to audio information may be of limited use. To facilitate access of useful segments of audio information, we need to identify entry points/jump locations in the audio recording. A change of speaker and long pauses can serve to identify entry points [Aron94]. By employing the above strategy, we can detect the boundaries of audio segments. An audio object is composed of a sequence of contiguous segments. In our model, the audio object’s interval is described by the starting time and ending time, which are equivalent to time-based media proposed by Gibbs et al. [GBT94]. Further, an audio object is described by a set of self-explanatory tags along with values.

Definition of an Audio Object

Formally, an audio-object is defined by (id, I, v) where
• id is an object identifier which is unique
• I, interval is described by \(<s, e> where s is the start time and e is the end time. Start time and end time satisfy \(e-s>0\)
• \(v\) is a finite set of tag:value pairs, i.e., \(v=\{<a_1;v_1><a_2;v_2>...<a_i;v_i>...<a_n;v_n>\}\) for a particular \(i\) where \(a_i\) is the tag and \(v_i\) is its value.

For example, an audio object’s \(v\) is defined as \(<Player:wayne gretzky>\), where the tag is “Player” and the value is “wayne gretzky.”

To facilitate query, we store objects’ intervals, tags and values into a database. Now the question is whether we can use these self-explanatory tags during the database schema design as attributes. The answer is no. This is because for the schema design, we need to know all tags (attributes) \textit{a priori}. However, there are two difficulties in defining all attributes in advance. First, descriptions of an object are added incrementally to the system, not at one time. In other words, new attributes for the object may be created which do not exist as attributes in the existing schema. Second, new objects are created which may have attributes entirely different from the existing schema’s attributes. Therefore, we explicitly store this self-explanatory tag along with values. Thus, each audio object has its own schema, rather than a common schema for all objects.

3.2 Metadata Acquisition and Annotation

An ontology is a model that defines a set of representational terms (concepts). These concepts collectively provide an abstract view of a domain. Further, these concepts are used as self-explanatory tags and values of an audio object for that particular domain. In other words, these concepts are served as metadata. Note that in the ontology there are numerous links between concepts such as IS-A, Part-Of, and so on. For example, a concept such as “National_Football_League” (NFL) (a specialization) would be located under a concept such as Football. Further, in the ontology some concepts can be instantiated, that is, a
representation of a specific instance of the concept is produced. Thus “Player:wayne gretzky” refers to a particular player.

To facilitate annotation of concepts with audio objects, concepts come from either domain dependent ontology or generic ontology such as CYC [LGu90], WordNet [Mil95], or Sensus [Sens]. We choose the former, domain dependent ontology. This is because in the latter case, objects may be annotated with more concepts which do not properly describe the objects. Further, users may get irrelevant objects which violate the principle of customization. Throughout this thesis proposal, the ontology described is in the sports domain. This ontology is usually obtained from generic sports terminology and domain experts [CLA]. In Fig. 2, a very small portion of the ontology is shown. The ontology is described by a directed acyclic graph (DAG). Each node in the DAG represents a concept. Arcs represent association between concepts. For example, “NBA” is a subconcept of “Basketball,” the association between them is IS-A.

In general, each concept contains tag, value(s) and a synonymous list. Note also that this tag or label name is unique in the ontology. In addition, the synonymous list of a concept contains vocabulary for which the concept is matched with user requests. Each interior node in the DAG is divided into two categories: exhaustive (EX) and non-exhaustive (NEX). If a concept is entirely partitioned into a set of sub-concepts, then we call this concept exhaustive (EX). Then this concept is not annotated into any audio object, but sub-concept(s) may be annotated to that object. On the contrary, when a concept cannot entirely be partitioned among sub-concepts, it is termed non-exhaustive (NEX). For example, in Fig. 1 the concepts "Professional" and "Non_Professional" are EX, and the concept "Football" is NEX. It is important to note that the interior node also contains some additional information. It contains a flag which denotes whether the concept
Figure 2. A Small Portion of An Ontology for Sports Domain

qualifies another concept. Hence, when the flag is set, the concept is used as an adjective for other concepts (e.g., for “professional basketball” the concept professional (adjective) qualifies concept basketball).

Leaf nodes in the DAG are classified into the two categories of no binding (NB) and run-time binding (RB). For NB, the concept’s value is empty. In other words, no instantiation is required. In contrast, for RB, the value is instantiated on the fly during annotation or user query. It also contains additional information such as how many words should be fetched during instantiation. For example, user requests “Give me news about player Wayne Gretzky.” The RB concept, “Player” is selected, due to the match of “player” with this leaf concept’s synonymous list. Since the concept is RB, its value is instantiated with “wayne gretzky.” Hence, the audio objects which contain “Player:wayne gretzky” as metadata in the database are retrieved. In contrast, user requests “Give me news about national hockey league.” NB concept,
“NHL” is selected due to the match of “national hockey league” with the concept’s synonymous list. Since the concept is NB, its value is not instantiated. Further, the objects which contain “NHL” as metadata in the database are retrieved.

Several concepts of the ontology can be candidate for the annotation of an audio object. However, some of these may be sub-concepts of others. In this case, we can deploy more specific concepts for annotation and discard their corresponding generalized concepts. It is important to note that when the query is requested by the generalized concepts, the audio object is still accessible. This is because the user requests first match with concepts of ontology, rather than straight search into the database.

Leaf concepts and NEX concepts of the ontology only serve as metadata for audio objects. In other words, leaf concepts and NEX concepts are only stored into the database. For NB, only tags and, for RB, both tags and values of concepts are stored in the database. Further, for NEX concepts, only tags are stored. This is because RB concepts are leaf nodes and NEX concepts are interior nodes. No EX concepts are stored as metadata in the database. This is because leaf concepts represent specific concepts, whereas EX concepts (interior nodes) represent more generalized concepts. Since the ontology depicts the association of different concepts, generalized concepts for the EX case may be expressed as a collection of specific concepts by traversing the DAG (for more details see Sec. 3.4). Furthermore, by storing only specific concepts in the database as metadata we save space and avoid redundancy in the schema design.

The success of customization depends upon the annotation process. Rather than describing each segment independently, which was strongly criticized by Smith et al., any arbitrary sequence of contiguous segments is described. These sequences of contiguous segments form an object. The process of creating relationships between a sequence of segments (object) and descriptions (here concepts) is known as stratification. Note that stratification can occur at the user interface level--someone is sitting at a console.
creating relationships. The stratification process can also be repeated as many times as necessary. Hence, end-users are able to select objects of interest with ever-increasing precision. It is important to note that here we assume end-users cannot do annotation; however, a domain expert can do annotation through a user interface.

Fig. 3 provides a schematic representation of how stratification is used. Audio is partitioned into several segments. Note that the lengths of segments are not equal. For example, the lengths of the first and second segments are 10- and 4-time unit respectively. For annotation, all available concepts in the ontology are shown to the user through an interface. The user then selects contiguous segments as an object and annotates the object with a set of specific concepts by pointing some concepts from the interface. Note that EX concepts are not used for annotation. Recall that the EX concept is more generalized concept. However, the NEX concept may be used for annotation. Further, if the RB concept is selected, its value is instantiated as a user input. However, it is possible that the user may want to describe an object with some new concepts that do not exist in the ontology. Then new concepts may be inserted into the ontology. In this situation, the user inputs the tag, value, a synonymous list, and associations with other concepts. This process allows the concepts to grow dynamically in the ontology.

The following example illustrates the annotation process (see Fig. 3). Objects $O_1$ (0-14) and $O_2$ (10-30) are defined and annotated with “NHL” and “Player:wayne gretzky” respectively. Since $O_1$ and $O_2$ overlap with each other, the intersected segment (10-14) of $O_1$ and $O_2$ contains an audio segment related to “NHL” and “Player:wayne gretzky.” Similarly, $O_5$ and $O_6$ are annotated with “NBA” and “Player:dennis rodman” respectively. Note that $O_5$ (45-70) contains $O_6$ (45-55). Hence if the query is “Give me national basketball association news which covers player Dennis Rodman,” $O_5$ is the result. In addition, $O_3$, $O_4$ and $O_7$ are annotated with some leaf concepts of the ontology. Note that one object ($O_7$) may be annotated with more than one concept. Further, one concept “NBA” is associated with more than one object.
Without loss of generality, we can assume an audio object cannot be associated with more than one NB concept. This is because NB concepts are disjoined. For example, when an object is associated with the NB concept “NBA,” it cannot be associated with another NB concept, “NHL.” Further, any audio segment cannot be associated with more than one NB concept. However, it is possible that a segment (audio object) may be associated with more than one RB concept. In addition, an NB concept can coexist with several RB concepts for a particular segment (object). For example, an object is associated with the NB type concept, “NHL,” and the RB concept, “Player:wayne gretzky.”

![Diagram](image)

**Figure 3. Annotation Using Stratification**

### 3.3 User Profile Generation

The ontology facilitates the expression of explicit user preferences in a common and simple way. Users can select a concept(s) by browsing the ontology through a user interface. They can traverse a certain level of the ontology and choose specific concepts. We assume that users cannot go beyond a certain point
in the level of ontology. This is because more specific concepts reside at the bottom part of the ontology, which are determined by the domain expert. Moreover, it is hard for novice users to differentiate among these bottom level concepts. Hence, although the ontology categorizes audio objects into a fine grain but the user profiles can be expressed in a coarse grain. Furthermore, with this simple approach, it is difficult to track changes in a user's interest over time. In order to capture users' implicit feedback, we need an intelligent agent, which we are currently investigating.

3.4 Processing Information Selection Requests (Database Queries in SQL)

We now show how SQL can be generated from users' requested queries expressed in plain natural language. In the current state of speech technology if we want conversion of speech to text in a speaker independent way we are required to restrict the vocabulary. The basic intuition for the generation of SQL is: first, users requests are matched with concepts of the ontology. Second, the SQL is generated using the more specific concepts of these matched concepts. Recall that audio objects are only annotated with more specific concepts. Further, virtually by residing on top of database, the ontology facilitates users flexibility and abstraction to ask queries into the system. This abstraction and flexibility cannot be achieved by simple keyword search. For example, an audio segment(s) is annotated with concept, “NHL” only; however, due to association of concepts in ontology (see Fig. 2), we can retrieve this segment(s) by using concepts “NHL”, “Hockey”, “Professional”, and so on in the user requests.

The SQL generation is done by a module, which we call the SQL generator (SG). The SG generates SQL by writing boolean conditions in the “where” clause. Then another module, which we call the query rewriter (QR), rewrites the query to achieve optimization. Note that we do not address any natural language issue which is beyond the scope of this research.

To describe the user selection requests, the sample database is described with the following self-explanatory schema:  

Audio_News (Id, Time_Start, Time_End, Meta_News, ...). Note that the Meta_News attribute corresponds to \{<a_1:v_1>,<a_2:v_2>, ..., <a_n:v_n>\}. Further, start time (s) and end time (e) map into
Time_Start and Time_End respectively. For the remainder of the thesis proposal, we assume Meta_News to be expressed as a set-valued attribute that is a feature of object-relational DBMS (SQL3) [CDNA’97], and that the SQL is written accordingly. However, in a pure relational case, the multi-valued attribute is implemented as a separate table for the purpose of normalization. Then SQL is expressed as a relational join.

To generate SQL, we need a grammar to parse user requests as a means of identifying concepts. Note that our goal here is not to check whether the sentence is syntactically correct. As a sample demonstration, we present a few rules in BCNF:

(1) \(<Sentence> = <A:V> [<Conjunctive_Word>|<Disjunctive_Word>|<Difference_Word><A:V>]^*

(2) \(<Sentence>=<A:V>(ADJ)<A:V>
    [<Conjunctive_Word>|<Disjunctive_Word>|<Difference_Word><A:V>]^*

(3) \(<Conjunctive_Word>=<Where>

(4) \(<Disjunctive_Word>=<Or>

(5) \(<Difference_Word>=<Except>

(6) \(<A> = <Tag of concept from the ontology>

(7) \(<V> = <Value of the concept>

... 

Production rule (1) and (2) specify that a sentence is composed of one or more concepts. However, in production rule (2), one concept qualifies the other concept (e.g., "Professional (adjective) basketball"). Note that the ADJ operator ensures that concepts appear adjacently in the sentence. Production rule (2) is described elaborately in Sec. 3.5. It is important to note that Conjunctive_Word, Disjunctive_Word, and Difference_Word serve as a signal for the SG that the sentence contains more than one concept.

**SQL Generation**

SQL is generated at the client site, submitted over the network to the server, and executed at the server. The intervals are then fetched over the network from the server to the client. The basic intuition for
the generation of query in SQL is as follows: tokens are generated from the user’s requested text after stemming. Using the production rules, tokens are associated with concepts when the synonymous lists of these concepts are matched with tokens. We call each of these associated concepts a QConcept. Among these QConcepts, some concepts are the sub-concept of other concepts in the ontology. In this case, we discard parent concepts. Recall that specific concepts are annotated with audio objects.

Now we discuss how each QConcept is mapped into the “where” clause of SQL. Note that by putting the boolean condition in the “where” clause, we are able to retrieve relevant audio objects. First, the QConcept is located in the DAG of the ontology by matching the user’s request to the concept's synonymous list. Next, we check whether the concept is a leaf concept. If it is already a leaf concept, we determine this concept’s category. If it is an NB type, its tag is only added in the “where” clause. For an RB type, the value is instantiated and tag:value pair is added to the “where” clause. However, if this QConcept is a non-leaf concept, all of its children concepts (or sub-concepts) are generated using depth/breadth first search (DFS/BFS). Among NB types, when children concepts become leaf concepts, these leaf concepts’ tags are added to the “where” clause connected via boolean “or” condition. Moreover for RB types, after instantiation of values, tag:value pairs are added to the “where” clause. Note that if, during the traversal of the graph, a certain concept is found to be NEX, its tag is also added to the “where” clause. However, no instantiation is required. This is because this concept is represented by an interior node and, hence, it cannot be an RB type. One important observation is that all concepts appearing in the SQL for a particular QConcept are expressed in disjunctive normal form. The following example illustrates the above notion: the user request, "Please give me news about the national football league." Using production rule (1), the sentence is parsed and “NFL” turns out to be the QConcept which is itself a leaf concept with an NB type. Hence, the SG generates SQL using only “NFL”:

```
SELECT Time_Start, Time_End
FROM Audio_News
WHERE "NFL" IN Meta_News
```
Let us consider the user request, “Please give me news about football.” Note that the concept “Football” has four leaf concepts (“NFL,” “CFL,” “College_Football,” and “HighSchool_Football”). Hence, these four NB type concepts’ tags are represented as boolean "or" condition in the “where” clause of SQL. Since the concept “Football” is NEX, the concept “Football” is also added to the “where” clause. In SQL:

```
SELECT Time_Start, Time_End
FROM Audio_News
WHERE "NFL" IN Meta_News OR "CFL" IN Meta_News
     OR "College_Football" IN Meta_News
     OR "HighSchool_Football" IN Meta_News OR "Football" IN Meta_News
```

**Disjunctive Query**

Although all leaf concepts of a particular QConcept are expressed in pure disjunctive form, Disjunctive_Word triggers the SG explicitly, indicating that the QConcepts are connected by boolean “or.” Further, if the user request contains more than one QConcept without any Conjuncuve_Word, Disjunctive_Word, or Difference_Word, we assume that QConcepts are connected in the “where” clause by boolean “or.” Here we demonstrate how RB concepts participate in SQL generation for the disjunctive query. For example, the user requests “Give me news related to team Lakers or player Dennis Rodman.” RB concepts “Team” and “Player” are selected from the ontology. After instantiation of values, “Team:lakers” and “Player:dennis rodman” are added to the “where” clause connected via “or.” In SQL:

```
SELECT Time_Start, Time_End
FROM Audio_News
WHERE "Team:lakers" IN Meta_News OR "Player:dennis rodman" IN Meta_News
```
Conjunctive Query (CQ)

To support conjunctive query (CQ), we assume that some triggering words (e.g., “where” in users’ requests) signal that the query is associated with a set of QConcepts and that these QConcepts are connected via boolean “and.” In CQ, without loss of generality, we assume users want to play audio segments in which two different QConcepts are present simultaneously. Two QConcepts simply cannot be expressed as the boolean “and” condition in the “where” clause. This is because in this case, only objects associated with two QConcepts simultaneously are only retrieved. However, there may be some overlapped objects which associate either one of the two QConcepts. Thus, we miss these intersected segments by simply writing the two QConcepts as boolean “and” in the “where” clause. We need to identify this intersected segment(s). Therefore, the process requires the checking of temporal intersect among selected objects by two QConcepts.

An audio object is essentially a temporal interval. This is because the object is described by start and end time. Temporal interval operators are therefore applicable to audio objects. There are 13 ways any two audio objects (equivalent to two intervals) can be related [AJ83]. These 13 can be expressed in terms of 7 relationships; however, 6 of the 13 represent no more than the simple inverse of others [LG93]. The temporal operators are as follows:

- **A Overlaps B**: Returns true if object B starts while object A is still active (the complementary operator is OverlappedBy).
- **A Contains B**: Returns true if B starts after A and ends before A (the complementary operator is During).
- **A Starts B**: Returns true if A and B start with the same time and A ends before B (the complementary operator is StartedBy).
- **A Finishes B**: Returns true if B starts before A and A and B end with the same time (the complementary operator is FinishedBy).
- **A Equals B**: Returns true if two sequences A and B are identical.
• **A Before B**: Returns true if A happens before B (the complementary operator is *After*).

• **A Meets B**: Returns true if B starts with next segment after A has ended (the complementary operator is *MetBy*).

Since “A Before B” and “A Meets B” are not relevant to the CQ query, we do not consider them here as temporal operators. Hence, to find intersected segments of two sets of intervals, selected by two QConcepts, we use Allen’s 9 temporal operators.

There are three alternative ways we can identify relevant segments for the temporal operators in client-server architecture. These three algorithms are conventional method (CM), semi-join method (SJ), and self-join method (SJM). Here, besides query processing cost, communication cost is taken into account. This is because our goal is to develop an algorithm which minimizes query response time by considering communication cost. We propose an efficient algorithm (SJM) that outperforms the other two algorithms in terms of response time. We describe each of the three algorithms one by one:

**Conventional Method (CM)**

In the conventional method, we generate and execute SQL and collect intervals (time_start, time_end) of the selected objects separately for each QConcept. Then, we apply temporal operator(s) over two collected interval sets at the client site. Finally, the segments are fetched from the stored audio based on the results of temporal operator(s). For example, the user requests “Give me hockey news where player Wayne Gretzky played.” Two QConcepts, “Hockey” and “Player:wayne gretzky” are selected. The following two SQLs are issued and executed.

```
SELECT Time_Start, Time_End
FROM Audio_News
WHERE “NHL” IN Meta_News OR “MinorLeague_Hockey” IN Meta_News
    OR “College_Hockey” IN Meta_News OR “Hockey” IN Meta_News
```
SELECT Time_Start, Time_End
FROM Audio_News
WHERE “Player:wayne_gretzky” in Meta_News

After getting the two interval sets of two SQL statements, we apply temporal operations on these interval sets to get the final interval set. Using this final interval set, the segments of stored audio are played.

It is important to note that this method may incur high response time. This is because here intervals associated with at least one QConcept are transferred over the network; however, it may happen that only a few intervals intersect. In this case, we may look for alternatives which can reduce unnecessary interval transfer over the network. Note that communication cost is proportional to the amount of data transfer. Hence, by avoiding unnecessary interval transfers, we can reduce communication cost.

**Semi-Join Method (SM)**

The semi-join method is similar to the frequently used semi-join technique in a distributed database environment to join two relations. Here, the order of SQL execution for QConcepts is important. Further, it uses a UDF at server to execute temporal operations. First, an SQL statement for a QConcept that satisfies the least number of intervals between the two QConcepts is issued and executed. Next, SQL is issued for the second QConcept for each interval of the first SQL’s result. In other words, intervals of first SQL result are passed as arguments of the UDF in the second SQL one by one. Therefore, in the second SQL’s result, intervals that intersect and satisfy the second QConcept are transmitted; however, in the transfer of the first SQL result, intervals that satisfy only the first QConcept are transmitted. The drawbacks of this approach depend on the cost parameter availability. In other words, the number of intervals selected by each QConcept should be known a priori. Further, a SQL statement for the second QConcept may be executed several times. This is because the same SQL may be invoked for different intervals of the result of the first SQL. Let us assume QConcept, “Hockey” returns least number of intervals. Hence the first SQL is:
SELECT Time_Start, Time_End
FROM Audio_News
WHERE “NHL” IN Meta_News OR “MinorLeague_Hockey” IN Meta_News
   OR “College_Hockey” IN Meta_News OR “Hockey” IN Meta_News

After gathering intervals of first SQL at client site, the second SQL for QConcept, “Player:wayne gretzky” is issued. In SQL:

SELECT Time_Start, Time_End
FROM Audio_News
WHERE “Player:wayne_gretzky” IN Meta_News
   AND Temporal_UDF (Time_Start, Time_End, Par_Time_Start, Par_Time_End)

Temporal_UDF is a user defined function that implements temporal operation between two intervals provided as arguments. Here, the first two arguments (interval) satisfy the second QConcept, and the last two arguments are replaced by each interval of the first SQL’s result one by one.

**Self-Join Method (SJM)**

The self-join method (SJM) uses the notion of join and a user defined function (UDF). It is the most optimized form when a few number of intervals intersect with each other among the intervals selected by two QConcepts. In this case, like SM, the UDF stored at server simply implements the temporal operator(s); however, the last two arguments of the UDF do not come from the client. Note that the number of relations used in the “where” clause of SQL is equal to the number of QConcepts. Although, all QConcepts use the same relation but each QConcept has its own copy.

The basic idea of SJM is as follows: first, we select an object (O1) which is associated with one QConcept and an object (O2) which is associated with another QConcept and check whether these two objects satisfy the temporal operator(s) using UDF. If UDF returns true, these two objects’ intervals are transferred over the network. Next, the entire process is repeated for this O1, with a new object, O2, and so
on. This iteration is implemented using the notion of join. The UDF is served as a joining condition between relations. Further, this UDF is not an expensive operation and the objects’ intervals that intersect and satisfy at least one QConcept are transferred over the network. Therefore, by avoiding unnecessary interval transfers, communication cost is reduced. In SQL:

```sql
FROM Audio_News A1, Audio_News A2
WHERE "Player:wayne_gretzky" IN A1.Meta_News
```

Fig. 4 gives a visualization of the three algorithms for CQ in client-server environment.

---

Figure 4. Different Methods of Conjunctive Query in the Client-Server Environment
Difference Query

When users’ requests possess Difference_Word, the difference query is issued. Without loss of generality, let us assume difference query to be associated with two QConcepts. First, SQL is executed for each QConcept and the intervals of each SQL’s are collected separately. Next, result intervals are determined by subtracting the first SQL’s interval set from the second SQL’s interval set. Finally, stored audios of these result intervals are fetched and played. For example, the user requests “Give me hockey news except news related to player Wayne Gretzky.” Two SQLs are issued for two QConcepts (“Hockey” and “Player:wayne gretzky”). Note that the two SQLs are identical to SQLs that are shown in CM approach of CQ query.

3.5 Optimizations

In this section, we present novel optimization techniques that rewrite the SQL query generated by the SG with the help of knowledge that comes from the ontology. This reduces the query response time without sacrificing precision and recall. First, we present an optimization to write SQL when one concept qualifies the other concept. Next, we show how knowledge of EX helps achieve further optimization. Finally, we show a special case in which further optimization can be achieved for difference query.

Qualified Disjunctive Form (QDF)

When one concept qualifies the other concept (production rule (2)), the straightforward approach is SJM. In other words, we treat the query as a conjunctive query. Without loss of generality, here we assume the leaf concepts of QConcepts used for querying are NB type. Recall that NB type concepts are disjoined (see Sec. 3.2). Therefore, SG may generate the query by writing two QConcepts as boolean “and” in the “where” clause. However, further optimization is possible by taking the intersection of all leaf concepts of the two QConcepts. This optimized method is known as qualified disjunctive form (QDF). Hence, QR only considers intersected leaf concepts for these QConcepts. By discarding the non-intersected concepts, the QR reduces the number of boolean conditions in the “where” clause for these QConcepts. However, the
traditional query optimizer eliminates the redundant leaf concepts by means of transforming a redundant expression into an equivalent one using boolean algebra (see [JK84] for more details). By employing QDF, we leverage the work of a traditional query optimizer. Note that, for these QConcepts in QDF form, leaf concepts are expressed in pure disjunctive form. The following example illustrates the conversion into QDF: the query is "Give me professional basketball news." This sentence is parsed using the production rule (2). The intersected leaf concepts of two QConcepts (professional and basketball) are “NBA,” “CBA,” “USBL,” “WNBA,” and “ABL” (see Fig. 1). The SQL for without QDF is:

```sql
SELECT Time_Start, Time_End
FROM Audio_News
WHERE ("NBA" IN Meta_News OR "CBA" IN Meta_News

  OR "NFL" IN Meta_News OR "CFL" IN Meta_News

  OR "PGA" IN Meta_News OR ...)

AND ("NBA" IN Meta_News OR "CBA" IN Meta_News

  OR "College_Basketball" IN Meta_News ....)
```

The SQL for QDF is:

```sql
SELECT Time_Start, Time_End
FROM Audio_News
WHERE "NBA" IN Meta_News OR "CBA" IN Meta_News

  OR "USBL" IN Meta_News OR "WNBA" IN Meta_News

  OR "ABL" IN Meta_News
```

**Optimization for EX concepts**

The SG does not allow the EX concept to appear as a boolean condition in the “where” clause; however, it does allow the NEX concept to appear in the “where” clause. The EX concept is exhaustive, which implies that it can be expressed as a collection of leaf concepts. Therefore, it is adequate to keep only
leaf concepts of the EX concept in the “where” clause. Thus, due to the avoidance of one boolean condition, the query response time may be reduced.

Fig. 5 shows how the number of boolean conditions impacts query response time. The X axis represents the number of boolean conditions in the “where” clause that are expressed in disjunctive form, and the Y axis represents query response time in milliseconds. Note that we assume client and server are at the same machine in order to avoid the impact of communication cost on response time. The Audio_News table is populated with 94 objects. Most of the objects are associated with a single concept. We observe that the increasing number of boolean conditions in the “where” clause makes response time high for a fixed number of tuples.

![Figure 5. Impact of Number of Boolean Conditions on Response time](image-url)
Optimization for Difference Queries

An ontology, in particular, helps to optimize a difference query when the second QConcept is the sub-concept of the first QConcept and all leaf concepts of these QConcepts are NB type. Rather than employing a general technique (fetching intervals using two SQLs for two QConcepts), we simply discard all leaf concepts of the second QConcept from the first QConcept before the generation of a single SQL. This is because these NB type leaf concepts of the two QConcepts are disjoined (see Sec 3.2). Hence, no common segment is shared among the NB type leaf concepts. As the first QConcept subsumes the second QConcept, a number of leaf concepts of the first QConcept are the leaf concepts of the second QConcept. Moreover, some of the leaf concepts of the first QConcept may not be part of the second QConcept. Therefore, any segment (object) associated with any of the leaf concepts of only the first QConcept (not part of the second QConcept) does not overlap with the segments (objects) associated with the leaf concepts of the second QConcept. Hence, this disjoined property guarantees that users do not get any unwanted segment due to this straightforward discarding of leaf concepts. In addition, communication cost is reduced because intervals of the second SQL’s result are not transferred over the network. The generation of SQL works in the following way: first, all leaf concepts of the first QConcept are added to a list. Next, all leaf concepts of the second QConcept are subtracted from this list. Finally, this list is used to generate an SQL. For example, the user requests “Give me hockey news except college hockey” or in SQL:

```
SELECT Time_Start, Time_End
FROM Audio_News
WHERE “NHL” IN Meta_News OR “MinorLeague_Hockey” IN Meta_News
   OR “Hockey” IN Meta_News
```

Note that QConcept, “Hockey” has three leaf concepts “NHL,” “MinorLeague_Hockey,” and “College_Hockey,” and “Hockey” is itself NEX concept. Since QConcept “Hockey” subsumes QConcept “College_Hockey,” leaf concepts “NHL” and “MinorLeague_Hockey” are only added to the where clause.
4 Research Plan and Experimental Implementation

In this section, we first summarize future work. Next, we present the current implementation status of PAC. We would like to extend this work in four directions: tools for segmentation and annotation, exploring the impact of the number of annotations/segment, addressing the time constraint query, and user-studies and their evaluation. A timetable for future work is shown in Fig. 6. Our initial focus will be on the development of tools for segmentation and annotation based on the ideas that we have described in Sec. 3.1 and Sec. 3.2. Next, we will concentrate on the study of the impact of the average number of annotations/segment on precision and recall for manual annotation. In parallel, we will develop an automatic technique which extracts highlighted sections of audio for time constraint query. Then, we will evaluate our technique. When

![Figure 6. Timetable for Future Work](image-url)
the study of the impact for manual annotation and its evaluation is complete, we will study the impact of the number of annotations per segment on precision and recall for automatic annotation. Finally, we would like to conduct a user-study to evaluate the use of ontology. Now, we present briefly technical details of each future direction except the first one.

**Impact of Number of Annotations/Segment for Manual Annotation**

The more we annotate the more the average number of annotations/segment raises. Recall that any arbitrary number of contiguous segments can be annotated incrementally due to stratification. An increase in the number of annotations/segment has two kinds of impact on precision and recall. First, the more we annotate, the higher the recall. This is because new segments will be retrieved as a result of new annotations. Second, we may get lower precision with a higher number of annotations. This is because the annotation process is tedious and time-consuming, a fact which contributes to error during annotation. In other words, some segments may be wrongly annotated due to a heavy cognitive mental load. The more we annotate, the more annotation error we may get. Therefore, we need to find an average number of annotations/segment which balances both considerations of precision and recall.

**Impact of Number of Annotations/Segment for Automatic Annotation**

We have pointed out that our annotation process is manual, which is time consuming and error-prone. Now, the question is what we can do to make annotation automatic. One way to make the annotation process automatic is to use the “wordspotting” technique [WB92]. Wordspotting is a particular application of automatic speech recognition in which the vocabulary of interest is relatively small, and it is the job of the recognizer to pick out only occurrences of words from this vocabulary in the speech to be recognized [JD95]. The output of a wordspotter is typically a list of keyword “hits,” labeled with each keyword’s start timings.
This automatic annotation process tremendously reduces human labor; however, it may still recognize falsely some concepts which can result in a deterioration of precision. In other words, we cannot eliminate this false identification, which is equivalent to human annotation error. We would like to extend our work on automatic annotation, and would like to observe further how manual and automatic annotation impact upon one another in terms of precision and recall.

**Time Constraint Query**

Users may be interested in highlights of the news. For this, we need to identify highlight sections of the news. By analyzing the pitch of the recorded news we can identify highlight sections. This is because it is well known in the speech and linguistics communities that there are changes in pitch under different speaking conditions [HG92, Sil87]. For example, when a speaker introduces a new topic there will be an increased pitch range. Sub-topics and parenthetical comments are often associated with a compression of pitch range. Since pitch varies considerably between speakers, it is necessary to find an appropriate threshold for a particular speaker. We will investigate all these issues. Further, we can employ the technique of clustering to reduce the number of highlighted sections.

**User-Studies and Evaluation**

We will conduct a user-study of the initial domain dependent ontology used in PAC. This user-study will assess the effectiveness of the use of the ontology. The evaluation will be largely exploratory and designed as formative research to help in the refinement of the ontology. Much data will be collected through preliminary use of PAC. We plan to define effectiveness of the use of the ontology primarily in terms: ease of query (i.e., the level of abstraction required to satisfy user requests and the number of requests covered by a given ontology).
Experimental Implementation

The development of the PAC system is still in progress. However from its current state, we can report on the foundation of the choices that have been initially taken. The system follows client server architecture (see Fig. 7). The Server (a SUN Sparc Ultra 2 model with 188 MBytes of main memory) has an Informix Universal Server (IUS) [Inf97], which is an object relational database system. IUS is used to store the metadata of each audio object and other relevant information. The schema which we used is described in Sec. 3.4. After getting sports content from the *Los Angeles Times*, audio feeds are stored in the server. Total duration of stored audio is 5 hours and 94 audio objects are defined. Most of these objects are annotated with single specific concept. To access data from the remote database to the client, Remote Method Invocation (*RMI*--a part of Java API provided by Informix) is used. At the client side, user input interface facilitates user profile information acquisition and voice input. User explicit and implicit feedback is stored at the server. To recognize voice input and convert into text, we used a speech recognition engine, IBM Via Voice [IBM98] and Java Speech API [Jav99].

The SG and QR modules were written in Java as application, and resided in the client. After converting speech to text, the SG module generates SQL (see Sec. 3.4) which is further optimized using QR (see Sec. 3.5) and this SQL is sent to the database. After that, the scheduler module receives a set of intervals as a result of the execution of SQL statement and if possible concatenates these intervals. Next, according to concatenated intervals, the scheduler fetches corresponding stored audios, using HTTP protocol. Hence, when the scheduler starts playing the first interval’s audio, it strives to download the other intervals’ in the background in order to avoid any further delay related to fetch audio segments over the network.
5 Conclusion and Contribution

In this thesis proposal we have presented a model for user-customized information selection and delivery for audio. In this context we have addressed some key research issues: segmentation of audio, metadata generation, selection of audio information, and user profile generation. By demonstrating the implementation of a domain dependent ontology (sports), we have demonstrated that the ontology enables end-users to construct user profiles. We also have demonstrated that the ontology facilitates the generation of metadata. We have demonstrated that how the ontology can be used to generate information selection requests (in SQL) from user requests (in plain natural language). We also have demonstrated that the correlation of concepts in the ontology provides end-users richer forms of information with which to query the system rather than simple keyword search. Finally, we have discussed novel optimization techniques that improve query processing performance, utilizing the knowledge associated with the ontology.
We anticipate a number of contributions in this proposed thesis:

- In this framework we show how to use a domain dependent ontology in order to facilitate
  - metadata generation
  - expression of user interests
  - information selection requests in database query
  - novel optimization techniques that improve query performance.

- We propose a mechanism through which the ontology can be used to provide end-users richer forms of information for query into the system, rather than keyword search.

- For the query, we use the most widely used query language, SQL rather than any new query language constructed from scratch. We extend the query language to support Allen’s temporal operations, using user defined functions (UDF) as an extension of SQL.

- We propose an efficient algorithm for the temporal operations of conjunctive queries in client-server architecture.

- We propose a data model that supports the notion of arbitrary attributes, which can be attached for each audio object whenever necessary.

Fig. 8 gives a visualization of our contributions.
Figure 8. Anticipated Contributions
References


