Foundations of Multimedia Information Systems *

Sherry Marcus
Office of Research and Development
MS # 4133
Washington, DC 20505.
E-mail: marcus@nego.umiacs.umd.edu

V.S. Subrahmanian
Institute for Advanced Computer Studies
Institute for Systems Research
Department of Computer Science
University of Maryland
College Park
Maryland 20742.
E-mail: vs@cs.umd.edu

Abstract

Though numerous multimedia systems exist in the commercial market today, relatively little work has been done on developing the mathematical foundations of multimedia technology. We attempt to take some initial steps towards the development of a theoretical basis for multimedia information system. To do so, we develop the notion of a structured multimedia database system. We begin by defining a mathematical model of a media-instance. A media-instance may be thought of as "glue" residing on top of a specific physical media-representation (such as video, audio, documents, etc.) Using this "glue", it is possible to define a general purpose logical query language to query multimedia data. This glue consists of a set of "states" (e.g. video frames, audio tracks, etc.) and "features", together with relationships between states and/or features. A structured multimedia database system imposes a certain mathematical structure on the set of features/states. Using this notion of a structure, we are able to define indexing structures for processing queries, methods to relax queries when answers do not exist to those queries, as well as sound, complete and terminating procedures to answer such queries (and their relaxations, when appropriate). We show how a media-presentation can be generated by processing a sequence of queries, and furthermore we show when these queries are extended to include *constraints*, then these queries can not only generate presentations, but also generate temporal synchronization properties and spatial layout properties for such presentations. We describe the architecture of a prototype multimedia database system based on the principles described in this paper.

---

*Author for Correspondence: V.S. Subrahmanian. This research was supported by the Army Research Office under grant DAAL-03-92-G-0225, by the Air Force Office of Scientific Research under grant F49620-93-1-0065, by ARPA/Rome Labs contract Nr. F30602-93-C-0241 (Order Nr. A716), and by an NSF Young Investigator award IRI-93-57756.

1 Introduction

Though there has been a good deal of work in recent years on multimedia, there has been relatively little work on multimedia information systems. In [22], the authors have developed a theoretical framework for multimedia database systems. They show how, given a set of media sources, each of which contains information represented in a way that is (possibly) unique to that medium, it is possible to define general-purpose access structures that represent the relevant "features" of the data represented in that medium. In simple terms, any media source (e.g. video) has an affiliated set of possible features. An instance of the media source (e.g. a single video clip) possesses some subset (possibly empty) of these features. Thus, a feature may be "Taurus" or "Mustang". The features associated with a media source may have properties of two types – those that are independent of any single media-instance, and those that are dependent upon a particular media-instance. Thus, for instance, the property `price(taurus,15k)` is true and is independent of any single video-clip. In contrast, the feature `color(taurus,blue)` may depend upon a particular video-clip. Marcus and Subrahmanian [22] develop a general scheme that, given a set of media-sources, and a set of instances of those media-sources, builds additional data structures "on top" of the physical representation of data in that medium. This physical representation allows for the definition of suitable query languages, and database query processing algorithms.

In this paper, we extend on our previous work in the following way: earlier, the set of features was a "flat" set. However, in many cases, there are inherent "feature-subfeature" relationships that exist. For instance, the feature `odometer` (of the Taurus) is certainly a subfeature of the feature `dashboard` (of the Taurus). When certain features/properties (and or relationships between them) are desired in a query, there maybe no forthcoming answer. However, it may be reasonable to substitute a subfeature for a feature – for example, returning a photograph of Clinton's face may be an appropriate substitute for a request for a full-length photograph of Bill Clinton. In other words, the structure of the set of features may allow us to provide "better" answers than we would have been otherwise able to provide.

In this paper, we define a certain mathematical structure on the set of features/states. Using this notion of a structure, we are able to define *methodologies for relaxing queries* when the original query does not have an answer. Finding answers to a query then boils down to finding a solution satisfying the query, or if that fails, to relax the query, and then attempt to solve the relaxed query. An optimal answer to a query is, informally speaking, a solution to a (possibly) relaxed version of the query – without relaxing the query any more than absolutely necessary. We are then able to define indexing structures for processing queries and finding optimal answers to queries, as well as sound, complete and terminating procedures to answer such queries.

More importantly, we show how, given a sequence of queries, we can generate a *media presentation* from the sequence of queries. Intuitively, in a media-presentation, each of the participating media are in a given state, and this state changes over time. For instance, in a simple audio-video multimedia system, a media presentation would consist of a sequence of audio-video clips.

We extend this notion of generation of media presentations to handle *constrained* queries.

The idea of constraints in logic based languages was originally due to Jaffar and lassez [17]. Intuitively, constrained queries, contain, in addition to the logical constraints, certain extra constraints (over different possible domains). These extra constraints, for example, may range over time-points and be used to achieve some temporal relationships in the presentation (e.g. "There must be a delay of at most 0.1 microsecond between successive video-clips"), or, alternatively, be used to achieve certain spatial layouts or "overlays" in the resulting display (e.g. the all document windows must be above all video windows, but may not use the lower half of the screen. We show how our framework can express all the multimedia presentation composition operations described as useful by Weiss et. al. [29].

Finally, we present a brief report on the status of a prototype multimedia information system that we are developing at the University of Maryland. Readers who wish to see this system may do so by visiting the University of Maryland – we hope to make the system available via anonymous ftp sometime in the mid-1995.

2 Motivating Example

2.1 General Idea

Suppose we consider a (futuristic) multimedia database that has a "built-in" multimedia presentation. When the user invokes this presentation, an audio-visual demonstration comes up on the system, explaining various aspects of some topic of interest (we will consider cars in this example). Thus, the presentation may explain the history of automobiles, current models, and future models currently being designed. However, there is one difference from this being a plain "movie." The user may interrupt the presentation at any given point in time, and may request additional information on some object(s) of interest. Requests may include:

1. The user may click on a particular car and ask
   (a) When was this car first produced?,
   (b) What does the previous model of this car look like?
   (c) Is there a picture of any of the European plants where this car is produced?
   (d) What documentation is available on the fuel injection system of this car?
2. The user may stop the presentation and ask questions such as:
   (a) Is there a large car available for under 12,000 dollars?
   (b) What do cars that satisfy the previous question look like?
   (c) Is it possible to produce a 5 minute audio-video clip of this car being driven?
   (d) Is it possible to produce an audio-video of this car being driven for 5 minutes when it has 60,000 miles on it?
   (e) Is it possible to zoom in and view the dashboard of the car?
   (f) Is it possible to get some information on the gauge in the top left corner of the dashboard?

Responding to the above queries in the obvious, expected manner, requires multimedia access to video databases, audio databases, and document databases (and possibly other databases as well). These clips must be indexed in such a way that queries which ask for a picture of such-and-such car be satisfiable. Thus, methods to identify and store such features need to be developed - in this paper, we will show how storing features is possible, though we will not discuss how such features can be automatically identified.

2.2 A Specific (Small) Multimedia Car Database System

In this section, we describe a very small multimedia database system containing information about two cars - the Ford Taurus and the Ford Mustang. Let us suppose that we have video information, audio information, and document data on the Taurus and the Mustang. Furthermore, assume that we have the following data available in these media.

1. **Video:** We have nine video-frames $v1, ..., v9$. Frame $v1$ shows a whole Ford Taurus, while $v2$ and $v3$ show the front. $v4$ is a video- frame of the Taurus' interior, $v5$ shows the dashboard of the Taurus, while $v6$ shows the fuel gauge of the Taurus. Frame $v7$ shows the Mustang, while frame $v8$ shows the interior of the Mustang. Lastly, frame $v9$ is a picture of the odometer in the Mustang.
2. **Audio:** There is one audio-frame $a1$ describing the overall features of the Taurus.
3. **Documents:** There are three documents $d1, d2, d3$ – document $d1$ describes various aspects of the Taurus' dashboard. Document $d2$ describes the Mustang, and finally, document $d3$ describes the odometer of the Mustang.

At this stage, we are not concerned with precisely how image information is represented on the `video` medium, nor are we concerned with how `audio` information is represented on an audio-medium. What we will do in this paper is to build some additional structure on top of each of these media. This additional structure will provide the "glue" that presents a unified view of the different media, making it possible to pose queries that retrieve the desired information from the different media, and present them in a uniform and coherent way to the end-user. The rest of this paper gives:

- a formal description of this "glue" (cf. Section 3),
- a formal language for querying a multimedia system that is "glued" together using the above notions (cf. Section 4),
- a formal description of an "answer" to a query, and "optimal" acceptable answer to a query when no correct answer exists.
- a set of algorithms for processing queries expressed in the above mentioned query language.

We will use this toy domain involving the Taurus and the Mustang to illustrate the formal definitions introduced in this paper.

3 Structured Multimedia Systems, Formalized

In this section, we will formally define the "glue" that allows us to link together multiple bodies of data on different media. A media-instance (formally defined below) consists of the actual data represented in the medium, together with a certain 8-tuple which constitutes the desired glue.

Definition 1 A *media-instance* is an 8-tuple

$$(\mathcal{S}, \mathbf{fe}, \mathtt{ATR}, \lambda, \Re, \mathcal{F}, \mathbf{Var}_1, \mathbf{Var}_2)$$

where:

- $\mathcal{S}$ is a set of objects called *states*, and
- $\mathbf{fe}$ is a set of objects called *features*, and
- $\mathtt{ATR}$ is a set of objects called *attribute values*, and
- $\lambda : \mathcal{S} \rightarrow 2^{\mathbf{fe}}$ is a map from states to sets of features, and
- $\Re$ is a set of relations on $\mathbf{fe}^i \times \mathcal{S}$ for $i \geq 0$, and
- $\mathcal{F}$ is a set of relations on $\mathcal{S}$, and
- $\mathbf{Var}_1$ is a set of objects, called *variables*, ranging over $\mathcal{S}$, and
- $\mathbf{Var}_2$ is a set of variables ranging over $\mathbf{fe}$.

Note that this 8-tuple may be thought of as residing "on top" of a given physical representation of a body of data in a given medium. Thus, for instance, if data is stored on CD-Rom, then there is a 8-tuple of the above form associated with the CD-Rom. Furthermore, each state $s \in S$ is physically represented on the CD-Rom using whatever electronic representation and/or compression scheme is being used. Physical retrieval of a frame from a medium (such as CD-Rom) may be accomplished using whatever (pre-existing) retrieval mechanism is used to access frames on the CD-Rom.

Let us see how the Car Multimedia System, which has three participating media, can be thought of as three media-instances.

Example 1 (Car Example Revisited) Let us suppose that we have video information, audio information, and document data on two cars – the Ford Taurus and the Ford Mustang. Furthermore, let us assume that these two cars share the same dashboard, i.e the same dashboard is plugged into both cars.

1. **(Video Media-Instance)** This is the tuple

$$(\{v1, \ldots, v9\}, \mathbf{fe}^1, \mathtt{ATR}^1, \lambda^1, \Re_1, \Re_2, \mathbf{Var}_1, \mathbf{Var}_2)$$

where $\mathbf{fe}^1 = \{\mathtt{taurus}, \mathtt{front}, \mathtt{t\_interior}, \mathtt{dashboard}, \mathtt{fuel\_gauge}, \mathtt{mustang}, \mathtt{m\_interior},$

odometer}, $\Re_1 = \{$type, left, right, range, color, successor$\}$, and $\Re_2 = \{$earlier$\}$. λ^1 is the following map:

$$
\begin{aligned}
\lambda^1(v1) &= \{\texttt{taurus}\}.\\
\lambda^1(v2) &= \{\texttt{taurus}, \texttt{front}\}.\\
\lambda^1(v3) &= \{\texttt{taurus}, \texttt{front}\}.\\
\lambda^1(v4) &= \{\texttt{taurus}, \texttt{t\_interior}\}.\\
\lambda^1(v5) &= \{\texttt{taurus}, \texttt{t\_interior}, \texttt{dashboard}\}.\\
\lambda^1(v6) &= \{\texttt{taurus}, \texttt{t\_interior}, \texttt{dashboard}, \texttt{fuel\_gauge}\}.\\
\lambda^1(v7) &= \{\texttt{mustang}\}.\\
\lambda^1(v8) &= \{\texttt{mustang}, \texttt{m\_interior}\}.\\
\lambda^1(v9) &= \{\texttt{mustang}, \texttt{m\_interior}, \texttt{odometer}\}.
\end{aligned}
$$

Intuitively, when we say that $\lambda^1(v2) = \{$taurus, front$\}$, this means that the video frame v2 possesses two features, viz. taurus and front. Likewise, the statement $\lambda^1(v9) = \{$mustang, m_interior, odometer$\}$ indicates that video-frame $v9$ possesses three features – mustang, m_interior, and odometer reflecting the fact that this constitutes a picture of the Mustang's odometer. The relations in $\Re_1$ are relations between features. We assume that the following tuples are contained in these relations:

```
type(taurus,midsize,S)          type(mustang,compact,S)
type(tempo,compact,S)           left(odometer,fuel_gauge,v5).
range(odometer,0,150000,S)      range(fuel_gauge(0,10,S)
color(taurus,red,v1)            color(front,red,v2)
color(front,red,v3)             color(t_interior,green,v4)
color(mustang,black,v7)         color(m_interior,black,v8).
```

Likewise, the relation earlier in $\Re_2$ is an inter-state relation; for example we may know that v1 was an earlier shot than v2, in which case the tuple earlier(v1,v2) is present. The *attribute values* present in ATR for this media-instance is the set midsize, compact, red, green, black as well as all integers.

2. **(Audio Media-Instance)** This is the tuple

$$(\{a1\}, \mathbf{fe}^2, \mathtt{ATR}^1, \lambda^2, \Re_1, \Re_2, \mathbf{Var}_1, \mathbf{Var}_2)$$

where $\mathbf{fe}^2 = \{$taurus$\}$ and $\lambda^2(a1) = \{$taurus$\}$.

3. **(Document Media-Instance)** This is the tuple

$$(\{d1, d2, d3\}, \mathbf{fe}^3, \mathtt{ATR}^1, \lambda^3, \Re_1, \Re_2, \mathbf{Var}_1, \mathbf{Var}_2)$$

where $\mathbf{fe}^3 = \{$taurus, t_interior, dashboard, mustang, m_interior, odometer$\}$, and the map λ^3 is defined to be:

$$
\begin{aligned}
\lambda^3(d1) &= \{\texttt{dashboard}\}.\\
\lambda^3(d2) &= \{\texttt{mustang}\}.\\
\lambda^3(d3) &= \{\texttt{mustang}, \texttt{m\_interior}, \texttt{odometer}\}.
\end{aligned}
$$

The above definition ignores the relationship between different features. For example, the Taurus' interior is a part of the Taurus, i.e. `t_interior` is a sub-feature of `taurus`. The following definition captures this intuition.

Definition 2 A *structured multimedia database system,* SMDS, is a quadruple $(\{\mathcal{M}_1, \ldots, \mathcal{M}_n\}, \leq, \mathtt{RPL}, \mathtt{SUBST})$ where:

- $\mathcal{M}_i = (\mathcal{S}^i, \mathbf{fe}^i, \mathtt{ATR}^i, \lambda^i, \Re^i, \mathcal{F}^i, \mathbf{Var}_1^i, \mathbf{Var}_2^i)$ is a media-instance, and
- $\leq$ is a partial ordering on the set $\bigcup_{i=1}^n \mathbf{fe}^i$, and
- $\mathtt{RPL} : \bigcup_{i=1}^n \mathbf{fe}^i \to 2^{\bigcup_{i=1}^n \mathbf{fe}^i}$ such that $f_1 \in \mathtt{RPL}(f_2)$ implies that $f_1 \leq f_2$. Thus, RPL is a map that associates with each feature f, a set of features that are "below" f according to the $\leq$-ordering on features.
- SUBST is a map from $\bigcup_{i=1}^n \mathtt{ATR}^i$ to $2^{\bigcup_{i=1}^n \mathtt{ATR}^i}$.

Intuitively, suppose f_1, f_2 are features. When $f_1 \leq f_2$, then this means (intuitively) that f_1 is deemed to be a subfeature of f_2. The map RPL is used (as we shall see later) to determine what constitutes an "acceptable" answer to a query. For example, if we want a video-state depicting a car `dashboard`, and if no such video-frame exists, then an alternative answer may be a picture of the `odometer` which happens to be a "subfeature" of the `dashboard`. If this is the desired behavior, then `odometer` should be in the set RPL(`dashboard`). The map SUBST is used to determine what attribute values may be considered to be appropriate replacements for other attribute values, i.e. if `red` $\in$ SUBST(`green`), then `green` is deemed to be an appropriate attribute value to substitute for `red`. This may be useful because an end-user may request a picture of a red Taurus; if no such picture is available, and `red` $\in$ SUBST(`green`), then the system may present the user with a picture of a `green` Taurus instead. Till we explicitly state otherwise, we will assume, for ease of presentation, that for all attribute constants a, $\mathtt{SUBST}(a) = 2^{\bigcup_{i=1}^n \mathtt{ATR}^i}$, i.e. any attribute constant can be substituted for any other attribute constant without restriction. Later (Section 7.1), we will remove this assumption.

Example 2 (Car Example Revisited) The structured multimedia database system associated with the car example, as formalized earlier, is the quadruple ({`video`, `audio`, `document`}, $\leq$, RPL, SUBST) where the $\leq$ ordering is the reflexive transitive closure of the following $\leq$ pairs: `fuel_gauge` $\leq$ `dashboard`, `dashboard` $\leq$ `t_interior`, `t_interior` $\leq$ `taurus`, `front` $\leq$ `taurus`, `odometer` $\leq$ `m_interior`, and `m_interior` $\leq$ `mustang`. Note that in general, the ordering on the set of features must be provided by the designer of the multimedia system and must correspond to the intuition that $f_1 \leq f_2$ means that f_1 is a "subfeature" of f_2.

An example of the replacement map RPL is given by:

$$
\begin{aligned}
\mathtt{RPL(taurus)} &= \{\mathtt{front}\}. \\
\mathtt{RPL(front)} &= \emptyset. \\
\mathtt{RPL(t\_interior)} &= \emptyset.
\end{aligned}
$$

$$\begin{array}{rcl}
\texttt{RPL(dashboard)} & = & \{\texttt{odometer}, \texttt{fuel_gauge}\}. \\
\texttt{RPL(fuel_gauge)} & = & \emptyset. \\
\texttt{RPL(mustang)} & = & \emptyset. \\
\texttt{RPL(m_interior)} & = & \emptyset. \\
\texttt{RPL(odometer)} & = & \emptyset.
\end{array}$$

For instance, the statement `RPL(taurus)` = {`front`} says that if we are looking for a particular type of frame depicting the `taurus` and if no such frame exists, then finding a frame of the same type depicting the feature `front` serves as an acceptable alternative to query.

4 Query Language

In this section, we develop a query language to express queries addressed to a structured multimedia system $\texttt{SMDS} = (\{\mathcal{M}_1, \ldots, \mathcal{M}_n\}, \leq, \texttt{RPL}, \texttt{SUBST})$ where

$$\mathcal{M}_i = (\mathbf{ST}^i, \mathbf{fe}^i, \texttt{ATR}^i, \lambda^i, \Re^i, \mathcal{F}^i, \mathbf{Var}_1^i, \mathbf{Var}_2^i).$$

We will develop a logical language to express queries. This language will be generated by the following set of non-logical symbols:

1. **Constant Symbols:**
   (a) Each $f \in \mathbf{fe}^i$ for $1 \leq i \leq n$ is a constant symbol in the query language. (For convenience, we will often refer to these as "feature-constants.")
   (b) Each $s \in \mathbf{ST}^i$ for $1 \leq i \leq n$ is a constant symbol in the query language. (For convenience, we will often refer to these as "state-constants.")
   (c) Each integer $1 \leq i \leq n$ is a constant symbol.
   (d) For each medium $\mathcal{M}_i$, $\mathcal{M}_i$ is a constant symbol. (Thus for instance, if $\mathcal{M}_1 =$ `video`, then `video` is a constant symbol, etc.)
   (e) A finite set of symbols called *attribute-constants*. (Intuitively, these are constants such as `red`, `blue`, `midsize`, `2`, `3`, etc. that are neither features, nor states, but reflect attribute-values.)
2. **Function Symbols:** `flist` is a binary function symbol in the query language.
3. **Variable Symbols:** We assume that we have an infinite set of logical variables $V_1, \ldots, V_i, \ldots$.
4. **Predicate Symbols:** The language contains
   (a) a binary predicate symbol `frametype`,
   (b) a binary predicate symbol, $\in$,
   (c) for each inter-state relation $R \in \Re^i$ of arity j, it contains a j-ary predicate symbol R^*.

(d) for each feature-state relation $\psi \in \Re_2^j$ of arity j, it contains a j-ary predicate symbol ψ^*.

As usual, a term is defined inductively as follows: (1) each constant symbol is a term, (2) each variable symbol is a term, and (3) if η is an n-ary function symbol, and $t_1, \ldots, t_n$ are terms, then $\eta(t_1, \ldots, t_n)$ is a term. A ground term is a variable-free term. If p is an n-ary predicate symbol, and $t_1, \ldots, t_n$ are (ground) terms, then $p(t_1, \ldots, t_n)$ is a (ground) atom. A *query* is an existentially closed conjunction of atoms, i.e. a statement of the form

$$(\exists)(A_1 \,\&\, \ldots, A_n).$$

Example 3 (Car Example Revisited) We now show certain examples of queries that an end-user interacting with the Car Multimedia System may wish to ask. We show how these (informal) queries may be formally expressed within our query language.

1. **(Unimedia Search)** "Is there a picture of a white Taurus ?" This query can be expressed as the formula:

   $$(\exists \mathtt{S})(\mathtt{frametype}(\mathtt{S}, \mathtt{video}) \,\&\, \mathtt{taurus} \in \mathtt{flist}(\mathtt{S}) \,\&\, \mathtt{color}(\mathtt{taurus}, \mathtt{white}, \mathtt{S})).$$

   It can easily be seen that the answer to this query is "no" as specified in the Car example. (Later, we will show how this query can be processed to derive this answer). However, if we ask for a picture of a red Taurus (the query looks identical to the above query except that "`white`" is replaced by "`red`"), the answer $\mathtt{S} = \mathtt{v1}$ should be returned.

   Note that when the answer $\mathtt{S} = \mathtt{v1}$ is returned to the user, what this means is that video-frame `v1` is "brought up" on the screen, i.e. it is piped to the appropriate output device. To bring up the video-frame `v1` requires a fetch operation to be executed on the video-medium. The precise method of fetching `v1` physically depends upon the lower-level storage scheme used, and we will not address that part explicitly.

2. **(Multimedia Search)** "Is there an audio description as well as a picture of a midsized car ?" This can be expressed as the query

   $$(\exists \mathtt{S}_1, \mathtt{S}_2, \mathtt{S}_3, \mathtt{C})(\mathtt{frametype}(\mathtt{S}_1, \mathtt{audio}) \,\&\, \mathtt{frametype}(\mathtt{S}_2, \mathtt{video}) \,\&\, \mathtt{C} \in \mathtt{flist}(\mathtt{S}_1) \,\&$$

   $$\mathtt{C} \in \mathtt{flist}(\mathtt{S}_2) \,\&\, \mathtt{type}(\mathtt{C}, \mathtt{midsize}, \mathtt{S}_3)).$$

   One answer to this query is: $\{\mathtt{S}_1 = \mathtt{a1}, \mathtt{S}_2 = \mathtt{v1}, \mathtt{C} = \mathtt{taurus}, \mathtt{S}_3 = \mathtt{S}_1\}$. Intuitively, this means that audio-frame `a1` and video frame `v1` should be "brought up" synchronously. An interesting point is in order here – note that an answer which is identical to the above, except that $\mathtt{S}_3 = \mathtt{v2}$ could also be returned (because the atom `type(taurus,midsize,-)` is true irrespective of what the third argument is instantiated to. Unless otherwise stated, we will assume that the only states that are "brought up" with respect to a query are those that occur as an argument to an atom of the form `frametype(-,-)` in the query. In the above query, $\mathtt{S}_3$ does not occur as an argument to a `frametype(-,-)` atom; hence, $\mathtt{S}_3$ is not brought up on a physical output device.

3. "Is there a picture of the interior of the Mustang ?" This can be expressed as the query

$$(\exists \mathtt{S})(\mathtt{frametype(S,video)} \,\&\, \mathtt{mustang} \in \mathtt{flist(S)} \,\&\, \mathtt{m\_interior} \in \mathtt{flist(S)}).$$

   This query can be answered in essentially the same way as in the last two examples.

The above example contains relatively simple queries. Part of the reason for the simplicity of the queries asked above is that there are very few interesting relationships expressed in the Car example. Below, we expand the Car example so as to be able to ask more interesting queries.

Example 4 (Car Example Revisited) Suppose we expand the Car example by adding to $\Re_1$ (the list of feature-state relationships), the following relations and their associated tuples:

```
engine(taurus,v6,S).          engine(mustang,v8,S).
engine(tempo,v4,S).           price(taurus, 1993, 15k).
price(mustang,1993,18k).      price(tempo,1993,12k).
airbags(taurus,1,S).          airbags(mustang,2,S).
airbags(tempo,0,S).
```

The designer of the Car multimedia system may wish to define a predicate called `diff` which takes two arguments `Car1` and `Car2` and returns a list of differences between `Car1` and `Car2` (as far as certain designated predicates are concerned). This predicate, `diff`, is a derived predicate defined as follows:

$$\begin{array}{rcl}
\mathtt{diff(Car1,Car2,L,S)} & \leftarrow & \mathtt{diff_engine(Car1,Car2,L1,S)} \,\& \\
 & & \mathtt{diff_price(Car1,Car2,L2,S)} \,\& \\
 & & \mathtt{diff_airbags(Car1,Car2,L3,S)} \,\& \\
 & & \mathtt{append(L1,L2,L4)} \,\& \\
 & & \mathtt{append(L4,L3,L).} \\
\mathtt{diff_engine(Car1,Car2,[enginedif(E1,E2)],S)} & \leftarrow & \mathtt{engine(Car1,E1,S)} \,\& \\
 & & \mathtt{engine(Car2,E2,S)} \,\&\, \mathtt{E}_1 \neq \mathtt{E}_2. \\
\mathtt{diff_engine(Car1,Car2,[],S)} & \leftarrow & \mathtt{engine(Car1,E1,S)} \,\& \\
 & & \mathtt{engine(Car2,E2,S)} \,\&\, \mathtt{E}_1 = \mathtt{E}_2. \\
\mathtt{diff_price(Car1,Car2,[pricedif(P1,P2)],S)} & \leftarrow & \mathtt{price(Car1,P1,S)} \,\& \\
 & & \mathtt{price(Car2,P2,S)} \,\&\, \mathtt{P}_1 \neq \mathtt{P}_2. \\
\mathtt{diff_price(Car1,Car2,[],S)} & \leftarrow & \mathtt{price(Car1,P1,S)} \,\& \\
 & & \mathtt{price(Car2,P2,S)} \,\&\, \mathtt{P}_1 = \mathtt{P}_2. \\
\mathtt{diff_airbags(Car1,Car2,[airbagsdif(A1,A2)],S)} & \leftarrow & \mathtt{airbags(Car1,A1,S)} \\
 & & \mathtt{airbags(Car2,A2,S)} \,\&\, \mathtt{A1} \neq \mathtt{A2}. \\
\mathtt{diff_airbags(Car1,Car2,[],S)} & \leftarrow & \mathtt{airbags(Car1,A1,S)} \\
 & & \mathtt{airbags(Car2,A2,S)} \,\&\, \mathtt{A1} = \mathtt{A2}.
\end{array}$$

The predicate append is defined in the standard way.

The above definition of the predicate diff enables an end-user to be able to ask queries such as "What is the difference between the Ford Taurus and the Ford Mustang ?" This can be expressed as the query

$$(\exists \mathtt{L}, \mathtt{S})(\mathtt{diff}(\mathtt{taurus}, \mathtt{mustang}, \mathtt{L}, \mathtt{S}).$$

The answer to this query consists of

$$\mathtt{L} \quad = \quad [\mathtt{enginedif}(\mathtt{v6}, \mathtt{v8}), \mathtt{pricedif}(\mathtt{15k}, \mathtt{18k}), /\mathtt{g}(1, 2)].$$

(In general, instead of defining the predicate diff explicitly in terms of differences between various components of it, we can do this implicitly, though for the purposes of this paper, we do not do so.)

We now define the important concept of a media-event.

Definition 3 A *media-event* w.r.t. SMDS is an n-tuple, $(s_1, \ldots, s_n)$ where $s_i \in \mathbf{ST}^i$, i.e. a media-event is obtained by picking, from each media-instance $\mathcal{M}_i$, a specific state.

Intuitively, a media-event is just a snapshot of a medium at a given point in time. Thus, for instance, if we are considering an audio-video multimedia system, a media-event consists of a pair (a, v) representing an audio-state a and a video-state v. The idea is that if (a, v) is such a media-event, then, at the point in time at which this event occurs, the audio-medium is in state a, and the video-medium is in state v.

Example 5 Suppose we return to the car multimedia example and assume that each of the three participating media is expanded by the addition of an artificial new state nothing which, intuitively, means that nothing is done in that medium if its current state is nothing. A media event may now be:

$$\mathbf{me_1} \quad = \quad (\mathtt{v1}, \mathtt{a1}, \mathtt{nothing}).$$

This denotes the global state where video frame v1 and audio frame a1 are "turned on" simultaneously, and no document is being displayed.

At any given point in time, a structured multimedia system, SMDS, has an associated media-event – it is entirely possible that all media are in the state nothing, but this is a perfectly legitimate media event. Thus, any media-event satisfies certain formulas in our query language, and falsifies others. This informal notion of satisfaction binds together the query language we have defined with the global state of the multimedia system (at a given point in time). For instance, the media-event $\mathbf{me_1}$ above (intuitively) satisfies the formula

$$(\exists \mathtt{S}_1, \mathtt{S}_2)(\mathtt{frametype}(\mathtt{S}_1, \mathtt{video}) \,\&\, \mathtt{frametype}(\mathtt{S}_2, \mathtt{audio}) \,\&\, \mathtt{taurus} \in \mathtt{flist}(\mathtt{S}_1) \,\&\, \mathtt{taurus} \in \mathtt{flist}(\mathtt{S}_2)).$$

The reason is that this query asks whether it is possible to find a video-state and an audio state with the feature taurus in both of them. The answer is "yes" and one possible solution is obtained by setting $\mathtt{S}_1 = \mathtt{v1}$, $\mathtt{S}_2 = \mathtt{a1}$, which (intuitively) corresponds to the media-event $\mathbf{me_1}$. Thus, (certain) media-events may be viewed as answers to queries. Below, we formalize this intuition and show how we can define a general notion of what it means for a media-event to satisfy a formulas.

Definition 4 Suppose $\mathbf{me} = (s_1, \ldots, s_n)$ is a media event w.r.t. the multimedia system $\mathtt{SMDS} = \{\mathcal{M}_1, \ldots, \mathcal{M}_n\}$ as specified above, and suppose F is a formula. Then we say that $\mathbf{me}$ *satisfies* F (or $\mathbf{me}$ makes F true), denoted $\mathbf{me} \models F$ as follows:

1. if $F = \mathtt{frametype}(a, b)$ is a ground atom, then $\mathbf{me} \models F$ iff $a = s_i$ for some $1 \leq i \leq n$ and the frametype of $\mathcal{M}_i$ is b. (Recall, from the definition of the frame data structure, that associated with each $\mathcal{M}_i$ is a string called $\mathcal{M}_i$'s `frametype`.)
2. if $F = (c \in \mathtt{flist}(b))$, and there exists an $1 \leq i \leq n$ such that c is a feature in $\mathbf{fe}^i$ and $b = s_i$, then $\mathbf{me} \models F$ iff $c \in \lambda^i(s_i)$.
3. if $F = \phi^*(t_1, \ldots, t_n, s)$ and for some $1 \leq i \leq n$, $t_1, \ldots, t_n \in \mathbf{fe}^i$ and $s \in \mathbf{ST}^i$, then $\mathbf{me} \models F$ iff $(t_1, \ldots, t_n, s) \in \phi \in \Re_2^i$.
4. if $F = (G \,\&\, H)$, then $\mathbf{me} \models F$ iff $\mathbf{me} \models G$ and $\mathbf{me} \models H$.
5. if $F = (\exists x)F$ and x is a state (resp. feature) variable, then $\mathbf{me} \models F$ iff $\mathbf{me} \models F[x/t]$ where $F[x/t]$ denotes the replacement of all free occurrences of x in F by t where t is a state (resp. feature) constant[1].

If F cannot be generated using the inductive definition specified above, then it is the case that $\mathbf{me} \not\models F$.

Definition 5 A *multimedia specification* is a sequence of queries $Q_1, Q_2, \ldots$ to `SMDS`.

The intuitive idea behind a multimedia specification is that any query defines a set of "acceptable" media-events, viz. those media-events which make the query true. If the goal of a media specification is to generate a sequence of states satisfying certain conditions (i.e. queries), then we can satisfy this desideratum by generating any sequence of media events which satisfies these queries.

Definition 6 Suppose $\mathbf{me}_0 = (s_1, \ldots, s_n)$ is the initial state of a multimedia-system, i.e. this is an initial media-event at time 0. Suppose $Q_1, Q_2, \ldots$ is a multimedia specification. A *multimedia presentation* is a sequence of media-events $\mathbf{me}_1, \ldots, \mathbf{me}_i, \ldots$ such that media-event $\mathbf{me}_i$ satisfies the query Q_i.

The intuitive idea behind a multimedia presentation is that at time 0, the initial media-event is $(s_1, \ldots, s_n)$. At time 1, in response to query Q_1, a new media-event, $\mathbf{me}_1$, which satisfies Q_1 is generated. At time 2, in response to query Q_2, a new media-event, $\mathbf{me}_2$, in response to query Q_2 is generated. This process continues over a period of time in this way. The following example illustrates the notion of a multimedia-presentation generated by a multimedia specification.

Example 6 (Car Example Revisited) Let us consider a very simple multimedia presentation that is generated by the following queries:

[1]The notion of a "free" variable is the standard one, cf. Shoenfield [25]).

1. $\mathtt{Q}_1 = (\exists S_1, S_2)(\mathtt{frametype}(S_1, audio)\,\&\,\mathtt{frametype}(S_2, video)\,\&\,\{taurus\} = \mathtt{flist}(S_1)$ $\&\,\{taurus\} = \mathtt{flist}(S_2)$. This query is satisfied by the media event

$$\mathbf{me_1} = (\mathtt{v1}, \mathtt{a1}, \mathtt{nothing})$$

because $\mathtt{S}_1 = \mathtt{a1}, \mathtt{S}_2 = \mathtt{a2}$ is a solution to the above query.

2. $\mathtt{Q}_2 = (\exists S_1)\,\&\,\mathtt{frametype}(S_1, video)\,\&\,\mathtt{flist}(S_1) = \{taurus, front\}$. There are two possible answers to this query, corresponding to the media-events

$$\mathbf{me_2^1} = (\mathtt{v2}, \mathtt{nothing}, \mathtt{nothing})$$

$$\mathbf{me_2^2} = (\mathtt{v3}, \mathtt{nothing}, \mathtt{nothing})$$

Thus, the multimedia specification (Q_1, Q_2) generates one of two possible multimedia presentations – either $\mathbf{me_1}, \mathbf{me_2^1}$ or $\mathbf{me_1}, \mathbf{me_2^2}$.

5 Formal User Request Language

In this section, we show how certain requests that the user may wish to make can be expressed using the formal query language described in [22]. Suppose, at a given point in time, that $(s_1, \ldots, s_n)$ is the current media-event. Suppose now that there is an algorithm which, given a user interrupt, will identify the feature (called F) in state s_i that the user is referring to. This may require signal processing and/or statistical pattern recognition techniques which are beyond the scope of this paper, but which are studied in other papers (e.g. Niblack et. al. [23], Gong et. al. [11], and Gupta et. al. [14]). The situation we are dealing with is one where the current media event is $(s_1, \ldots, s_n)$, and the user wishes to obtain further details on one or more features of the current media-event.

Query I. Suppose the user wishes to find all states (irrespective of the medium involved) in which feature F occurs. This can be expressed as the query $\mathtt{info}(F)$ defined as:

$$(\exists S) F \in \mathtt{flist}(S).$$

This is a relatively "vague" query in that it asks for a list of all states in which F is a feature. The system may respond with a menu of possible answers (in different media such as audio, video, document, etc.) that satisfy the given query.

Query II. Suppose the user wishes to ask a more specific query where s/he is only interested in information about F on a particular medium. This can be expressed as the query $\mathtt{info}(F, M)$ defined as:

$$(\exists S)\mathtt{frametype}(S, M)\,\&\,F \in \mathtt{flist}(S).$$

Thus, when the user asks the query $\mathtt{info(mustang, audio)}$ s/he is asking for audio-clips containing information about the Mustang. These may include sound clips reflecting engine noise of the Mustang, as well as an audio sales-pitch for the Mustang.

Query III. The user wants to ask about all frames that possess feature F and that possess a certain property $p(-,-,\ldots,-)$. This can be encoded as the query $\mathtt{info}(\mathtt{F},\mathtt{p}(-,-,\ldots,-)) =$

$$(\exists \mathtt{S})(\mathtt{F} \in \mathtt{flist}(\mathtt{S})\ \&\ \mathtt{p}(-,-,\ldots,-,\mathtt{S}).$$

Thus, for instance, the user may ask for all frames containing a red Mustang. Here, $p(-,-,\ldots,-)$ is the atom `color(mustang,red)`.

It is easy to see that the property $p(-,-,\ldots,-)$ can be easily replaced with a conjunction of atoms.

Query IV. Suppose the user wishes to ask a *change* query – intuitively, such queries ask for a media-instance that is just like the current instance, except for a few changes. For example, the user of the CAR multimedia system may want a car just like a Mustang (which s/he is currently viewing) except that the desired car must have a higher fuel efficiency. Note that the Mustang must have been generated as a response to a previous query (i.e. a previous set of criteria), so what is really being asked for is an object which satisfies those same criteria, with some modifications articulated in the user's change request. This kind of request can be represented as the query $\mathtt{change}(OldQ, Add, Del)$ where Add and Del are sets of atoms. The answer to this query is obtained by deleting, from $OldQ$, the atoms in Del, and adding in the atoms in Add.

Formally, if we have the query $OldQ = (\exists)(\mathtt{A}_1\ \&\ \ldots\ \&\ \mathtt{A}_n)$ and if $Del \subseteq \{\mathtt{A}_1,\ldots,\mathtt{A}_n\}$ and if Add is a set of atoms, then

$$\mathtt{change}(OldQ, Add, Del) = (\exists)(\mathtt{B}_1\ \&\ \ldots\ \&\ \mathtt{B}_m)$$

where $\{\mathtt{B}_1,\ldots,\mathtt{B}_m\} = (\{\mathtt{A}_1,\ldots,\mathtt{A}_n\} - Del) \cup Add$.

Thus, for example, if the current query is "Find me a picture of a red Taurus" expressed as:

$$OldQ = (\exists \mathtt{S})(\mathtt{frametype}(\mathtt{S},\mathtt{video})\ \&\ \mathtt{taurus} \in \mathtt{flist}(\mathtt{S})\ \&\ \mathtt{color}(\mathtt{taurus},\mathtt{red},\mathtt{S})$$

then the change query which asks for a white Taurus can be specified as

$$change(OldQ, \{\mathtt{color}(\mathtt{taurus},\mathtt{white},\mathtt{S})\}, \{\mathtt{color}(\mathtt{taurus},\mathtt{red},\mathtt{S})\}).$$

At this stage, the user may wish to recall the motivating example in Section 2.1, and see how the queries specified there may be expressed in our language.

Example 7 (User Requests of Section 2.1) Consider an audio-video-document multimedia system, and suppose our current media-event is $(\mathtt{v}_1,\mathtt{nothing},\mathtt{nothing})$ and furthermore, suppose that the only feature in $\mathtt{v}_1$ is `taurus` and that this Taurus is red in color. we go through the commands issued in by the user in Section 2.1 one by one. The user clicks on the Taurus in this example.

1. *When was this car first produced?* This corresponds to the query $(\exists \mathtt{Y},\mathtt{S})\,\mathtt{prod\_year}(\mathtt{taurus},\mathtt{Y},\mathtt{S})$. Here, `prod_year` is a predicate specifying the year in which a particular car was first produced.

2. *What does the previous model of this car look like?* This can be viewed as a change query. Suppose the initial query had been $\texttt{OldQ} = (\exists \texttt{S})(\texttt{frametype(S, video)} \mathbin{\&} \texttt{taurus} \in \texttt{flist(S)} \mathbin{\&} \texttt{model(taurus, 3, S)})$. The change consists of $Add = \texttt{model(taurus, 2, S)}$ and $Del = \texttt{model(taurus, 3, S)}$. The new query is $\texttt{change(OldQ, Add, Del)}$.

3. *Is there a picture of any of the European plants where this car is produced?* This can be expressed as the query $(\exists \texttt{S})(\texttt{frametype(S, video)} \mathbin{\&} \texttt{factory} \in \texttt{flist(S)} \mathbin{\&} \texttt{in(factory, europe, S)} \mathbin{\&} \texttt{produces(factory, taurus, S)})$.

4. What documentation is available on the fuel injection system of this car? This can be expressed as the query $(\exists \texttt{S})(\texttt{frametype(S, document)} \mathbin{\&} \texttt{taurus} \in \texttt{flist(S)} \mathbin{\&} \texttt{fuel\_inj\_sys} \in \texttt{flist(S)})$.

When the user stops the presentation and interjects with new requests, all these new requests may be viewed as change queries. Let us suppose that the current media-event is generated by the first query below.

1. *What large cars are available for under 12,000 dollars?* This corresponds to the query $(\exists \texttt{C})(\texttt{type(C, large, S)} \mathbin{\&} \texttt{price(C, P, S)} \mathbin{\&} \texttt{P} \leq \texttt{12000})$.

2. *What do cars that satisfy the previous question look like?* This corresponds to the change query with OldQ as above, $Del = \emptyset$ and $Add = \{\texttt{frametype(S, video)}, \texttt{C} \in \texttt{flist(S)}\}$.

3. Is it possible to produce a 5 minute audio-video clip of this car being driven? This corresponds to the change query where OldQ is as in the preceding item, $Del = \emptyset$ and $Add = \{\texttt{frametype}(\texttt{S}_1, \texttt{audio}), \texttt{C} \in \texttt{flist}(\texttt{S}_1), \texttt{is\_driven}(\texttt{C}, \texttt{S}_1), \texttt{duration}(\texttt{S}_1, 5), \texttt{duration}(\texttt{S}, 5)\}$.

4. *Is it possible to produce an audio-video of this car being driven for 5 minutes when it has 60,000 miles on it?* This corresponds to the change query with OldQ as above, $Del = \emptyset$ and $Add = \{\texttt{mileage}(\texttt{C}, 60000, \texttt{S}_1)\}$.

5. *Is it possible to zoom in and view the dashboard of the car?* This corresponds to the change query with OldQ as above, $Del = \emptyset$ and $Add = \texttt{frametype}(\texttt{S}_2, \texttt{video}), \texttt{dashboard} \in \texttt{flist}(\texttt{S}_2)\}$. Note that the change query thus specified may have two video displays "on" simultaneously. There are a number of ways to handle this. One possibility is that the answer to $\texttt{S}_2$ is shown, in preference to the instantiation of S. Alternatively, the video frame corresponding to $\texttt{S}_2$ can be popped up on a different window.

6. *Is it possible to get some information on the gauge in the top left corner of the dashboard?* This corresponds to the change query with OldQ as above, $Del = \emptyset$ and $Add = \{\texttt{frametype}(\texttt{S}_3, \texttt{X}), \texttt{G} \in \texttt{flist}(\texttt{S}_3), \texttt{at}(\texttt{G}, \texttt{top}, \texttt{S}_2), \texttt{at}(\texttt{G}, \texttt{left}, \texttt{S}_2), \texttt{gauge}(\texttt{G}, \texttt{S}_3)\}$.

In sections 4 and 5, we have developed a language to express various kinds of queries and user requests; however, we have not developed data structures to store media-instances, nor have we developed algorithms to process queries and user requests. In the next section (i.e., Section 6), we define indexing structures to store media-instances, and subsequently, we show (Section 7) how to use these structures to process queries and user requests efficiently.

6 Access Structures

Suppose $\mathtt{SMDS} = (\{\mathcal{M}_1, \ldots, \mathcal{M}_n\}, \leq, \mathtt{RPL}, \mathtt{SUBST})$ where: $\mathcal{M}_i = (\mathcal{S}^i, \mathbf{fe}^i, \mathtt{ATR}^i, \lambda^i, \Re^i, \mathcal{F}^i, \mathbf{Var}_1^i, \mathbf{Var}_2^i)$ is a media-instance. Then we can associate the data structure shown in Figure 1 with SMDS.

Let us return to the car example involving the Ford Taurus and the Ford Mustang. As described earlier, this multimedia system has eight features in all, and these are ordered as specified in Example 2. For each of these eight features, we have a node of type `featurenode` with the `name` field of the node being the feature name. The top half of Figure 2 shows the graph of feature nodes, representing the $\leq$ ordering on features.

Let us consider the featurenode whose `name` field is `taurus`. The `children` field of this node is a pointer to a list of two nodes of type `node1`. The first (resp. second) of these two nodes has, in its `element` field, a pointer to the `featurenode` containing, as its `name` field, the string "Front" (resp. "t_interior").

The `statelist` of this node contains a pointer, denoted PTR1 in Figure 2 which points to a list of nodes of type `node2`. Each node is this list contains, in its `state` field, a pointer to the representation of that state in the medium on which it is stored. For example, if the state is a video-state, then this pointer points to the location/address on the video-tape/disc where that frame is stored. In this particular example, PTR1 points to a list of two nodes of type `node2` – the first of these has, in its `state` field, a pointer to the location at which video frame $v1$ is stored, while the second node has, in its `state` field, a pointer to the location at which the audio-frame $a1$ is stored.

The `replacelist` associated with this node contains a pointer to a list of nodes of type `node3`. Each of these nodes has, in its `feat` field, a pointer to a node of type `featurenode`. The idea is that if "Front" is pointed to by a node in the `replacelist` of `taurus`, then this means that if a user wishes to see a state (e.g. a video state) of the Taurus, and if a state with the desired properties is not present in the `statelist` associated with `taurus`, then an acceptable alternative is a state (with desired properties) in the `statelist` associated with the feature `front`. For instance, we may wish to see documentation on the Taurus. There may no state in the `statelist` associated with `taurus` that has, as its `frametype`, the string "document." In such a case, the presence of the feature `front` in `taurus`'s `replacelist` indicates that the user is willing to accept documentation on the `front` feature in lieu of that (unavailable) documentation on the `taurus`.

In general, given an SMDS $(\{\mathcal{M}_1, \ldots, \mathcal{M}_n\}, \leq, \mathtt{RPL}, \mathtt{SUBST})$, the associated access structure is defined as follows:

1. For each state $s \in \bigcup_{i=1}^{n} \mathcal{S}^i$, there is a pointer to a node of type `statenode`. Intuitively, if P_s is the pointer associated with state s, then P_s.`rep` is a pointer to the physical location of state s on the appropriate medium. Thus, for example, if s is an audio-state, then P_s.`rep` may be a pointer to a specific track on the audio-tape. P_s.`flist` points to a list of features possessed by state s.

2. For each feature $f \in \bigcup_{i=1}^{n} \mathcal{F}^i$, there is a pointer P_f to a node of type `featurenode`. P_f.`name` is simply a string representing the name of the feature. P_f.`children` points

```
type featurenode = record of  /* nodes in feature graph */
     name : string;  /* nameof feature */
     children: ^node1; /* points to a list of pointers to the children */
     statelist: ^node2; /* points to a list of pointers to states that*/
                            /* contain this feature */
     replacelist: ^node3; /* points to a list of descendants whose */
                          /* associated states can be deemed to have the */
                          /* the feature associated with this node */
     end record;


type node1 record of
     element: ^featurenode; /* points to a child of a featurenode */
     next :   ^node1; /* points to next child */
     end record;

type node2 record of
     state: ^statenode; /* pointer to the list associated with a state */
     link:  ^node2;  /* next node */
     end record;

type node3 record of
     feat: ^featurenode; /* pointer to a node that can be deemed to have */
                         /* the feature associated with the current node */
     link1: ^node3;
     end record;

type statenode record of
     rep:  ^framerep;
     flist: ^node4;
     end record

type node4 record of
     f : ^featurenode;
     link2: ^node4;
     end record:
```

Figure 1: Data Structure for Structure Multimedia Database Systems

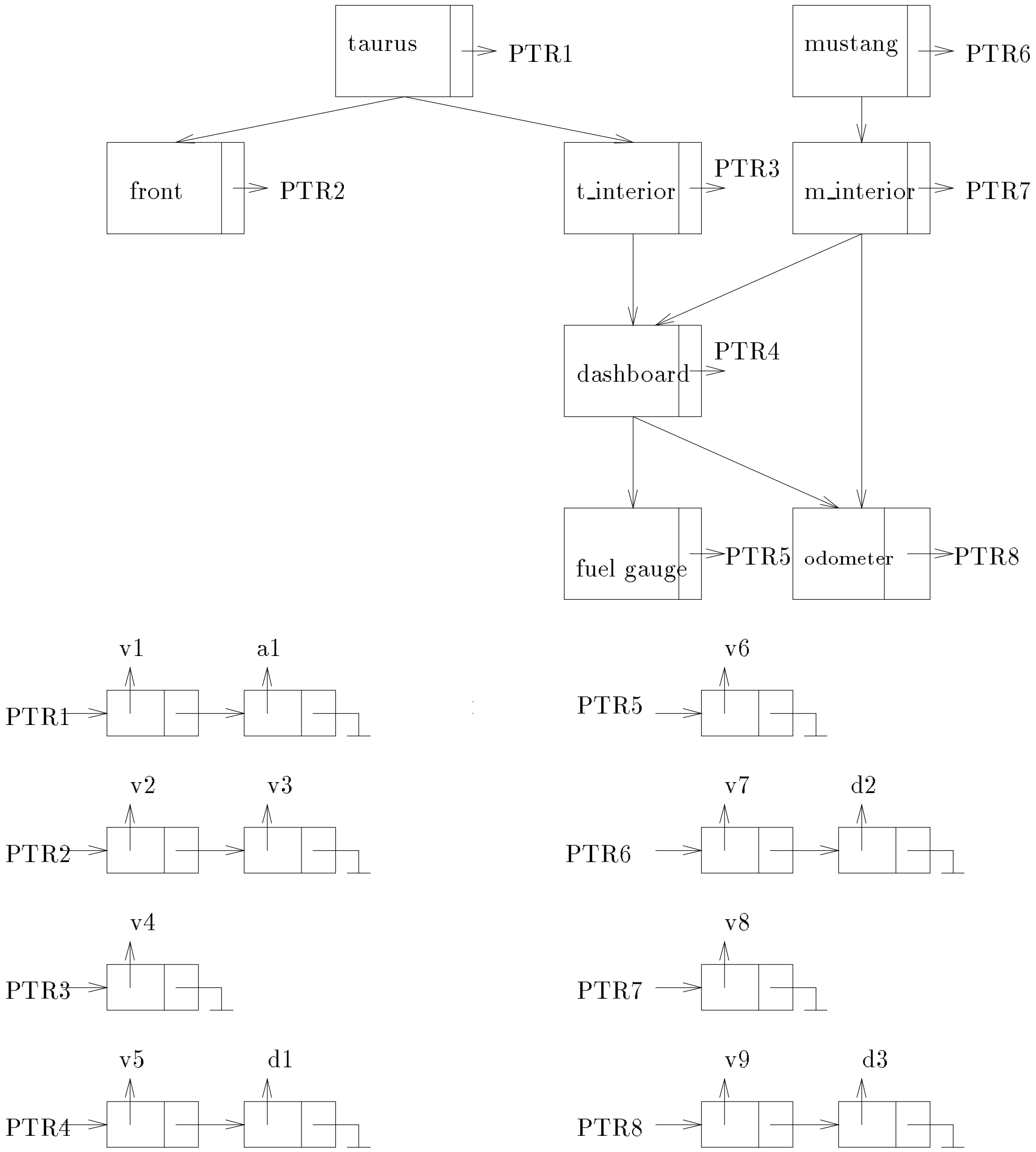

Figure 2: Indexing Structure for the Car Multimedia Database System

to a list L of pointers to `featurenodes` – L contains a pointer to feature f' iff:

- $f' \leq f$, and
- there is no f'' such that $f' < f'' < f$.

P_f.`statelist` points to a list of pointers to states that possess feature f. P_f.`replacelist` is a pointer to a list of `featurenodes` – f' is pointed to by a pointer in P_f.`replacelist` iff $f' \in \mathtt{RPL}(f)$.

The above definition specifies how, given any SMDS, it is possible to set up indexing structures to access information in that SMDS.

7 Answering Queries

In this section, we will first formalize the notion of an *answer* to a query. Then we will define the notion of an *optimal* answer. Finally, we will show how such optimal answers can be computed.

7.1 What's an Answer ?

Suppose Q is a query in our query language. Suppose c_1, c_2 are two constant symbols of the same type (i.e. both are feature symbols, or both are state symbols, or both are attribute symbols); it is not necessary that $c1$ and $c2$ be distinct. By $Q[c1/c2]$, we denote the query which is just like Q except that all occurrences of $c1$ in Q are replaced by $c2$. Thus, for example, if

$$Q \quad = \quad (\exists \mathtt{S})(\mathtt{p}_1(\mathtt{a},\mathtt{b},\mathtt{S})\,\&\,\mathtt{p}_2(\mathtt{a},\mathtt{b},\mathtt{c},\mathtt{S}))$$

then

$$Q[\mathtt{a}/\mathtt{b}] \quad = \quad (\exists \mathtt{S})(\mathtt{p}_1(\mathtt{b},\mathtt{b},\mathtt{S})\,\&\,\mathtt{p}_2(\mathtt{b},\mathtt{b},\mathtt{c},\mathtt{S})).$$

Let us consider a structured multimedia database system $\mathtt{SMDS} = (\{\mathcal{M}_1, \ldots, \mathcal{M}_n\}, \leq, \mathtt{RPL}, \mathtt{SUBST})$. This SMDS generates a query language as described in Section 4. We now define an ordering, $\preceq$, on queries: $Q_1 \preceq Q_2$ iff there exists an attribute symbol a occurring in Q_2 and an attribute symbol $b \in \mathtt{SUBST}(a)$ such that $Q_1 = Q_2[a/b]$.

It is easy to see that $\preceq$ is a pre-ordering, i.e. it is reflexive and transitive.

This ordering, $\preceq$ is useful in query processing for the following reasons: consider a query Q of the form $(\exists \mathtt{S})(\mathtt{p}_1(\vec{\mathtt{t}}_1,\mathtt{S})\,\&\ldots\&\,\mathtt{p}_n(\vec{\mathtt{t}}_n,\mathtt{S}))$. This query asks for a state in which the properties $\mathtt{p}_1(\vec{\mathtt{t}}_1), \ldots \mathtt{p}_n(\vec{\mathtt{t}}_n)$ hold. However, suppose no such state exists. Then we may consider returning an answer to a slightly "weaker" query, viz. that obtained by replacing an attribute a_1 in Q, by another attribute resp. a_2 such that answers w.r.t. a_2 are deemed to be acceptable answers w.r.t. a_1, i.e. $a_2 \in \mathtt{SUBST}(a_1)$. To see how this works, let us reconsider the car multimedia database system.

Example 8 (Car Example Revisited) Suppose we consider the query

$$Q = (\exists \mathtt{S})(\mathtt{frametype}(\mathtt{S},\mathtt{video})\,\&\,\mathtt{taurus} \in \mathtt{flist}(\mathtt{S})\,\&\,\mathtt{color}(\mathtt{taurus},\mathtt{white},\mathtt{S})).$$

This query asks for a picture of a white Taurus. However, no such picture is available, but a picture of a red Taurus is certainly available (viz. `v1`). Thus, the query $Q^{\star} = Q[\mathtt{white}/\mathtt{red}]$ is true w.r.t. the car multimedia database system. It is easy to see that $Q^{\star} \preceq Q$.

However, consider a slightly different query.

Example 9 (Car Example Revisited) Suppose an end-user wishes for documentation on the interior of of the Ford Taurus. Thus, the query being posed is:

$$Q = (\exists \mathtt{S})(\mathtt{frametype}(\mathtt{S},\mathtt{document})\,\&\,\mathtt{taurus} \in \mathtt{flist}(\mathtt{S})\,\&\,\mathtt{t\_interior} \in \mathtt{flist}(\mathtt{S})).$$

No documentation is available on the Ford Taurus' interior. Hence, there is no state `S` that can cause query Q to become true. Furthermore, there no attributes mentioned in this query; hence, substitution of attributes is not feasible either. However, documentation (document $d1$, in particular) is indeed available on the Taurus' `dashboard`. Had `dashboard` been a member of `RPL(t_interior)`, then $\mathtt{S} = d1$ could have been returned as an answer to the query $Q' \preceq Q$ where

$$Q' = (\exists \mathtt{S})(\mathtt{frametype}(\mathtt{S},\mathtt{document})\,\&\,\mathtt{dashboard} \in \mathtt{flist}(\mathtt{S})).$$

This is because $\mathtt{dashboard} \in \mathtt{RPL}(\mathtt{t\_interior})$ means that `dashboard` is an acceptable alternative (in queries) to `t_interior`.

The above example appears to indicate that the ordering $\preceq$ does not completely capture the right notion of an answer. We now extend the definition of $\preceq$ to handle this intuition.

Definition 7 Suppose Q_1 and Q_2 are queries. We say that Q_1 is a *relaxation* of Q_2, denoted $Q_1 \sqsubseteq Q_2$ iff there exist features $f_1, f_2 \in \bigcup_{i=1}^{n} \mathbf{fe}^i$ such that

- $Q_1 \preceq Q_2[f_1/f_2]$ and
- $f_2 \in \mathtt{RPL}(f_1)$.

If we examine the preceding example, we will observe that $Q' \sqsubseteq Q$, i.e. Q' is an acceptable weaker alternative to the query Q.

We are now in a position to formalize the notion of an *answer* to a query requesting a state that satisfies the query, and then define the notion of an *optimal* answer. Given a query Q, we use $\mathsf{DOWN}(Q)$ to denote the set of all queries Q' such that $Q' \sqsubseteq Q$.

Definition 8 (Answer) Suppose $Q = (\exists \mathtt{S}_1 \cdots \mathtt{S}_r)(\exists \mathtt{F}_1 \cdots \mathtt{F}_m)(\mathtt{p}_1^*(\vec{\mathtt{t}}_1)\,\&\cdots\&\,\mathtt{p}_k^*(\vec{\mathtt{t}}_k))$ is a query where $\mathtt{S}_1,\ldots,\mathtt{S}_r$ is a list of all state-variables occurring in Q and $\mathtt{F}_1,\ldots,\mathtt{F}_m$ is a list of all feature-variables occurring in Q. An *answer to* Q w.r.t. a structured multimedia database system $\mathtt{SMDS} = (\{\mathcal{M}_1, \ldots, \mathcal{M}_n\}, \leq, \mathtt{RPL})$ is a substitution of the form

$$\theta \quad = \quad \{\mathtt{S}_1 = \mathtt{s}_1, \ldots, \mathtt{S}_\mathtt{r} = \mathtt{s}_\mathtt{r}, \mathtt{F}_1 = \mathtt{f}_1, \ldots, \mathtt{F}_\mathtt{m} = \mathtt{f}_\mathtt{m}\}$$

where $s_1, \ldots, s_r$ are state-constants and $f_1, \ldots, f_m$ are feature-constants such that for some query $Q' \sqsubseteq Q$, $Q'\theta$ is true in $\mathtt{SMDS}$.

Intuitively, an answer to a query Q is any substitution that causes the original query (or a weaker, but acceptable query) to be true w.r.t. the structured multimedia database system. Let us return to the example of the multimedia car database system to understand the concept of an "answer".

Example 10 (Car Example Revisited) Let Q be the query

$$(\exists \mathtt{S})(\mathtt{frametype}(\mathtt{S}, \mathtt{video}) \,\&\, \mathtt{taurus} \in \mathtt{flist}(\mathtt{S}) \,\&\, \mathtt{color}(\mathtt{taurus}, \mathtt{white}, \mathtt{S})).$$

There is no way of instantiating the variables in Q so that this query is true because the Taurus shown in frame $v1$ is red in color, not white. The weaker query $Q' =$

$$(\exists \mathtt{S})(\mathtt{frametype}(\mathtt{S}, \mathtt{video}) \,\&\, \mathtt{taurus} \in \mathtt{flist}(\mathtt{S}) \,\&\, \mathtt{color}(\mathtt{taurus}, \mathtt{red}, \mathtt{S})).$$

is certainly satisfiable with $\mathtt{S} = \mathtt{v1}$; notice that $Q' \sqsubseteq Q$. This means that answers to Q' are considered to be acceptable answers to Q; hence, the substitution $\mathtt{S} = \mathtt{v1}$ is an answer to the original query Q.

Consider now the query $Q^\sharp =$

$$(\exists \mathtt{S})(\mathtt{frametype}(\mathtt{S}, \mathtt{video}) \,\&\, \mathtt{front} \in \mathtt{flist}(\mathtt{S}) \,\&\, \mathtt{color}(\mathtt{front}, \mathtt{red}, \mathtt{S})).$$

It is easy to see that $Q^\sharp \sqsubseteq Q$ because $Q^\sharp = Q[\mathtt{taurus/front}, \mathtt{white/red}]$. Hence, $\mathtt{S} = \mathtt{v2}$ is also an answer to the original query Q.

The reader will notice that $Q^\sharp = Q'[\mathtt{taurus/front}]$, i.e. $Q^\sharp \sqsubseteq Q' \sqsubseteq Q$. Hence, Q' is, intuitively, "closer" to the original query Q, and therefore, the answer $\mathtt{S} = \mathtt{v1}$ should be preferred to the answer $\mathtt{S} = \mathtt{v2}$.

The following two definitions capture the above intuition.

Definition 9 Suppose $\mathcal{Q}$ is a set of queries. Query $Q \in \mathcal{Q}$ is said to be $\sqsubseteq$-maximal iff for all $Q' \in \mathcal{Q}$, $Q \sqsubseteq Q'$ implies that$Q' \sqsubseteq Q$.

Note that a non-empty set, $\mathcal{Q}$, of queries may contain one or more $\sqsubseteq$-maximal queries and that maximal elements are not necessarily unique.

Definition 10 Using the same notation as in Definition 8, we say that

$$\theta \quad = \quad \{\mathtt{S}_1 = \mathtt{s}_1, \ldots, \mathtt{S}_\mathtt{r} = \mathtt{s}_\mathtt{r}, \mathtt{F}_1 = \mathtt{f}_1, \ldots, \mathtt{F}_\mathtt{m} = \mathtt{f}_\mathtt{m}\}$$

is an *optimal answer* to query Q iff for some $\sqsubseteq$-maximal query $Q' \in \mathsf{DOWN}(Q)$, $Q'\theta$ is true in $\mathtt{SMDS}$.

In the preceding example, this means that $\mathtt{S} = \mathtt{v1}$ is an optimal answer to the query Q rather than $\mathtt{S} = \mathtt{v2}$.

7.2 Processing Queries

In the preceding section, we have defined what constitutes an optimal answer to a query. In this section, we devise algorithms that use the indexing structure defined earlier to answer queries. As queries are defined inductively, our query processing algorithms themselves will be defined inductively as well.

7.2.1 Membership Queries

Suppose we consider a ground atom of the form $\mathtt{t} \in \mathtt{flist(s)}$ where t is a feature-constant and s is a state-constant. As the query is ground, the answer is either yes or no. The algorithm below shows how such a query may be answered. It uses a function, SORTDOWN(t) which returns a list of features – this list is obtained by topologically sorting [20] (in descending order) the partially ordered set $(\mathtt{RPL}(t), \leq)$. For now, we assume $\mathtt{SORTDOWN}(t)$ is nondeterministic in the sense that it will nondeterministically return some valid topologically sorted set of features in $\mathtt{RPL}(t)$.

Example 11 (Car Example Revisited) Consider the feature `m_interior` in the car multimedia system. Suppose $\mathtt{RPL(m\_interior)} = \{\mathtt{dashboard}, \mathtt{odometer}, \mathtt{fuel\_gauge}\}$. Then, SORTDOWN(`m_interior`) could be either of the two sequences: `dashboard, odometer, fuel_gauge` or the sequence `dashboard, fuel_gauge,odometer`. Note that the feature `fuel_gauge` can never precede `dashboard`.

```
proc ground_in(t:string; s:↑statenode):boolean;
   if ground_in1(t,s) then return "true" and halt
   else
      while SORTDOWN(t)≠ ∅ do
      begin
      Let v be the first element in SORTDOWN(t);
      if ground_in1(v, s) then return "true" and halt
      else SORTDOWN(t):= SORTDOWN(t)−{v}.
      end
      if SORTDOWN(t)= ∅ then halt and return "false."
end proc.

proc ground_in1(t:string; s:↑statenode):boolean;
   found := false; ptr := s.flist;
      while (not(found) & ptr ≠ NIL) do
         if ((ptr.f).name = t) then found := true
         else ptr := ptr.link2 ;
      return found.
end proc.
```

It is easy to see that algorithm ground_in1 above is linear in the length of `flist(s)`. The algorithm ground_in is linear in $m \times \|\mathtt{flist}(s)\|$ where $m = \|\mathtt{SORTDOWN}(t)\|$.

Suppose we now consider non-ground atoms of the form $(\exists)(\mathtt{t} \in \mathtt{flist(s)})$ where either one, or both, of t, s are non-ground.

(Case 1: s ground, t non-ground) In this case, all that needs to be done is to check if $s.\mathtt{flist}$ is empty.That is, we see if there are any features associated with this state. If there are none, then there is no solution to the existential query "$(\exists t) t \in \mathtt{flist}(s)$." Otherwise, simply return $((s.\mathtt{flist}).f).name$ as the value of t. Thus, this kind of query can be answered in constant time; the only processing required is to find the first feature associated with state s , if such a feature exists.

(Case 2: s non-ground, t ground) This case is more interesting. t is a feature. Suppose PTR points to the `featurenode` associated with t. If PTR.`statelist` is non-NIL, then return (PTR.`statelist`).`state`. That is, we return the state which has the desired feature. If PTR.`statelist` is NIL,i.e. there are no states which has this feature, then check, one by one, (that is, in the order in which SORTDOWN(t) is enumerated) for each $f \in$ SORTDOWN(t), whether PTR(f).`statelist` is non-NIL where PTR(f) is the pointer associated with the `featurenode` f. If such an f exists satisfying this property, then let f_0 denote the first such f in SORTDOWN(t), return "true" and the substitution

$$\mathtt{S} = (\mathtt{PTR}(\mathtt{f}_0).\mathtt{statelist}).\mathtt{state}$$

otherwise, return "false."

Thus, this kind of query can be answered in time $O(\sum_{v \in \mathrm{SORTDOWN(t)}}(k_v))$ where k_v is the length of the list PTR(v).`statelist`.

(Case 3: s non-ground, t non-ground) In this case, find the first `featurenode` (in a topological sorting of all features in the multimedia system) which has a non-empty "statelist" field. If no such `featurenode` is present, then no answer exists to the query "$(\exists s, t)\ t \in \mathtt{flist}(s)$."as there are no states present with any feature in them. Otherwise, let PTR be a pointer to the first such featurenode. Return the solution

$$\mathrm{t = PTR;\ \ s = PTR.statelist.rep).}$$

Thus, this kind of query can be answered in constant time.

The following result shows that the above algorithms are sound, complete, and guaranteed to terminate.

Theorem 1 Suppose Q is a query of the form $(\exists)(f \in \mathtt{flist}(s))$. Then:

1. **(Soundness)** If the above algorithms return a substitution θ, then θ is an optimal answer to query Q.
2. **(Completeness)** Suppose θ is an answer to Q (i.e. $Q'\theta$ is true in `SMDS` for some $Q' \sqsubseteq Q$). Then there exists a topological sorting $t_1, \ldots, t_r$ of $\mathtt{RPL}(t)$ and a $\sqsubseteq$-maximal query $Q^\star \in \mathsf{DOWN}(Q)$ such that:
   (a) $Q' \sqsubseteq Q^\star \sqsubseteq Q$, and
   (b) $Q^\star\sigma$ is true in `SMDS` and
   (c) σ is returned by the above algorithms.

3. **(Failure)** The above algorithms will return "false" iff there is no answer to query Q.

Proof. (1) Case 0: As $t \in \texttt{flist}(s)$ where s and t are constants is a ground query the algorithm ground_in returns an answer of "true" or "false".

Case 1: Let $(\exists t) t \in \texttt{flist}(s) = Q$ where s is a ground and t is not ground. If $\texttt{flist}(s)$ is empty, the algorithm returns "false" as an answer to Q since there's nothing in that state to be found. Otherwise, we return the name of the first feature found in the featurelist. This query just asks for any arbitrary feature in state s which is what the algorithm returns.

Case 2: This is where the notion of optimal answer comes into play. The query $Q = (\exists s) t \in flist(s)$ where t is ground and s is non ground. If statelist(t) is not empty, then we return the value of the first state on the list. If it's empty, then our algorithm returns the answer $s = s_0$ in the following way: suppose $\texttt{SORTDOWN}(t) = t_1, \ldots, t_r$. Then s_0 must be the first state associated with the statelist of t_j for some $1 \leq j \leq r$. Furthermore, for all $j < i$, the statelists associated with t_j must be empty.

It is now easy to verify that $s = s_0$ is an optimal answer (there may be others as well). To see this, suppose $\theta = \{s = s_0\}$ is not an optimal answer via the witness $Q' = Q[t/t_i], 1 \leq i \leq r$. Then there is a query $Q^\star$ such that $Q' \sqsubset Q^\star$. $Q^\star$ must be of the form $Q[t/t_j]$ for some $1 \leq j \leq r$. Furthermore, as t_i is the first item in the enumeration $t_1, \ldots, t_r$ of a topological sorting of $\texttt{RPL}(t)$ that has a nonempty associated statelist, it must be the case that $j \geq i$. Furthermore, $Q^\star = Q'[t_i/t_j]$. By the assumption, as $Q' \sqsubset Q^\star$, and as $Q^\star = Q'[t_i/t_j]$, it must be true that $t_j \leq t_i$. But then, by the definition of topological sorting, $j \leq i$, which is possible only if $i = j$, i.e. $t_i = t_j$. But this contradicts the assumption that $Q' \sqsubset Q^\star$.

Case 3: The query $Q = (\exists s, t) s, t \in flist(s)$ where s and t are non ground. The answer returned by the algorithm is of the form $\{s = s_0, t = t_0\}$. t_0 is the first feature returned by $\texttt{SORTDOWN}(t)$ whose statelist is not empty. It is now easy to verify that $\{s = s_0, t = t_0\}$ is an optimal answer.

(2) As before, there are four cases to consider, depending upon whether s, t are ground or not.

(Case 0: s ground, t ground) In this case, suppose Q' is true for some $Q' \in \mathsf{DOWN}(Q)$. θ must be empty as Q is variable-free. If $t \in \lambda^i(s)$, then $t \in \texttt{flist}(s)$ and the answer "true" is returned immediately on the first call to ground_in by algorithm "ground." Otherwise, Q' is obtained from from Q by substituting some $t' \in \texttt{RPL}(t)$ for t in Q. Let $X = \{t^\star \mid t' \leq t^\star \,\&\, t^\star \in \texttt{RPL}(t) \,\&\, t^\star \in \texttt{flist}(s)\}$. If $X = \{t'\}$, then the answer "true" (via witness Q') is an optimal answer; otherwise, let $t^\flat$ be any maximal element of X. Then "true" is an optimal answer for query Q (via witness $Q^\star = Q[t/t^\flat]$. To see that there is a topological sorting of $\texttt{RPL}(t)$ that causes the answer "true" to be returned (via witness $Q^\star$), consider $Y = \{t^\flat \in \texttt{RPL}(t) - \{t\} \mid t^\star < t^\sharp\}$. Then it is easy to see that there is a topological sorting of $\texttt{RPL}(t)$ of the form:

$$\langle \text{topological sorting of Y} \rangle, t^\star, \text{topological sorting of } \texttt{RPL}(t^\flat) - (Y \,\cup\, \{t, t^s tar\}) \rangle.$$

If this were the enumeration generated by $\texttt{SORTDOWN}(t)$, then our algorithm will return the answer "true," with $Q^\star$ as the witness.

(Case 1: s ground, t nonground) Follows immediately from Case (1) of the algorithm.

(Case 2: s non-ground, t ground) In this case, suppose Q' is true for some $Q' \in \mathsf{DOWN}(Q)$. If $t \in \lambda^i(s_0)$ for some s_0, then $t \in \mathtt{flist}(s_0)$ and the answer $\{s = s_0\}$ is returned immediately. Otherwise, Q' is obtained from from Q by substituting some $t' \in \mathtt{RPL}(t)$ for t in Q. Let $X = \{t^\star \mid t' \leq t^\star \,\&\, t^\star \in \mathtt{RPL}(t) \,\&$ there exists at least one state s_1 such that $t^\star \in \mathtt{flist}(s_1)\}$. The rest of the proof now follows using the same technique as in Case 0.

(Case 3: s non-ground, t non-ground) Immediate from the above cases.

(3) Immediate. □

7.3 Property Queries

In this section, we consider queries of the form $(\exists)p^*(t_1, \ldots, t_n, s)$ to a structured multimedia system $\mathtt{SMDS} = (\{\mathcal{M}_1, \ldots, \mathcal{M}_n\}, \leq, \mathtt{RPL}, \mathtt{SUBST})$. Suppose p is a relation in the media-instance $\mathcal{M}_i$. There are two cases to consider, depending upon whether s is ground, or whether s is non-ground.

(Case 1: s ground) If s is ground, we can associate with Q, a tree called a *query tree*, denoted $\mathtt{QT}(Q)$, as follows:

1. the root of $\mathtt{QT}(Q)$ is labeled with Q.
2. If N is a node in $\mathtt{QT}(Q)$ and if N is labeled with the query Q_N, then:
   (a) N is a leaf-node if $Q_N = p^*(t_1^N, \ldots, t_n^N, s)$ and if $p(t_1^N, \ldots, t_n^N, s)\theta$ is a ground instance of $p(t_1, \ldots, t_n, s)$ that is true in media-instance $\mathcal{M}_i$ where $\mathcal{M}_i$ is the media-instance in which p is defined. In this case, θ is an answer to Q and N is called a "success" node.
   (b) Otherwise, for each $\sqsubseteq$-maximal element $Q^\sharp \in \mathsf{DOWN}(Q_N) - \{Q_N\}$, node N has a child labeled with $Q^\sharp$. (If $\mathsf{DOWN}(Q_N) - \{Q_N\} = \emptyset$, then N is a "failure" node.

(Case 2: s non-ground) In this case, let $\mathtt{POSS} = \{s' \mid \exists i$ such that $s \in \mathcal{S}^i$ and $p(t_1', \ldots, t_n', s') \in \Re_1^i$ for some n-tuple $(t_1', \ldots, t_n')\}$. Then construct a tree called the `non-ground query tree` associated with Q, denoted $\mathtt{NGQT}(Q)$, as follows.

1. (Level 0, i.e. Root) The root of $\mathtt{NGQT}(Q)$ is labeled with Q.
2. (Level 1 nodes) For each $s_i' \in \mathtt{POSS}(Q)$, the root has a child N labeled with the query $Q_N = p^*(t_1, \ldots, t_n, s_i')$. Note that each query associated with node N has a last argument that is ground (and hence, Case 1 may be applied here).
3. The subtree rooted at node N is $\mathtt{QT}(Q_N)$.
4. Suppose N' is a leaf node in the subtree, $\mathtt{QT}(Q_N)$, whose root (residing at level 1 in $\mathtt{NGQT}(Q)$) is labeled with Q_N.
   (a) Suppose N' is a success node in $\mathtt{QT}(Q_N)$ labeled with the substitution θ and the query $p(-, \ldots, -, s_N)$. Then N' is a *success node of* $\mathtt{NGQT}(Q)$ and is labeled with the substitution $\theta \cup \{s = s_N\}$.

(b) Suppose N' is a failure node in $\mathtt{QT}(Q_N)$. Then N' is a failure node in $\mathtt{NGQT}(Q)$.

The following theorem shows that $\mathtt{NGQT}(Q)$ accurately captures the notion of an answer.

Theorem 2 Suppose Q is a query of the form $(\exists)p^*(t_1,\ldots,t_n,s)$. Then:

1. **(Soundness)** If $\mathtt{QT}(Q)$ (resp. $\mathtt{NGQT}(Q)$) contains a success node labeled with θ, then θ is an optimal answer to the query Q.
2. **(Completeness)** If Q has an answer, then there exists a node N in $\mathtt{QT}(Q)$ (resp. $\mathtt{NGQT}(Q)$) that is an optimal answer to query Q.
3. **(Failure)** There is no answer to query Q iff $\mathtt{QT}(Q)$ (resp. $\mathtt{NGQT}(Q)$) contains no success node.

Proof. **(1)** Suppose θ is the label of the success node Q_N in $\mathtt{QT}(Q)$. Then $Q_N\theta$ is true in $\mathtt{SMDS}$. We need to show that Q_N is a $\sqsubseteq$-maximal query in $\mathsf{DOWN}(Q)$ that is true in $\mathtt{SMDS}$ and then we would have proven optimality of θ by definition.

Suppose $Q = Q_0,\ldots,Q_j = Q_N$ is the path from the root of $\mathtt{QT}(Q)$ to Q_N (where $j \geq 0$). By definition, none of the Q_i's, $i < j$, are true in $\mathtt{SMDS}$ (for otherwise, these would have been success nodes in $\mathtt{QT}(Q)$). Hence, again by construction, there is no query Q' in $\mathsf{DOWN}(Q)$ such that Q' is true in $\mathtt{SMDS}$ and such that $Q \sqsubset Q'$ (were this the case, then $Q_i = Q'$ for some $i < j$). Thus, Q_N is a $\sqsubseteq$-maximal query in $\mathsf{DOWN}(Q)$ that is true in $\mathtt{SMDS}$, and we are done.

(2) Suppose θ is an answer to Q, i.e. there is a query $Q_N \in \mathsf{DOWN}(Q)$ such that $Q_N\theta$ is true in $\mathtt{SMDS}$. By definition, there is a $\sqsubseteq$-maximal query Q_M that is true in $\mathtt{SMDS}$ and such that $Q_N \sqsubseteq Q_M$. Furthermore, there exists a sequence of queries $Q_1,\ldots,Q_j$ such that:

$$Q = Q_1 \sqsubset Q_2 \sqsubset \cdots \sqsubset Q_{j-1} \sqsubset Q_j = Q_M$$

and such that $\{Q_1,\ldots,Q_j\} \subseteq \mathsf{DOWN}(Q)$ and such that none of $Q_1,\ldots,Q_{j-1}$ is true in $\mathtt{SMDS}$ and such that for all $Q' \in \mathsf{DOWN}(Q)$, there is no $1 \leq r \leq j-1$ such that $Q_r \sqsubset Q' \sqsubset Q_{r+1}$. Then $Q = Q_1,\ldots,Q_j = Q_M$ is a path in the tree $\mathtt{QT}(Q)$ and hence, M is a success node in $\mathtt{QT}(Q)$. This completes the proof.

(3) Immediate from (1) and (2) above.

The analogous proofs for the case when s is non-ground (i.e. when $\mathtt{NGQT}(Q)$ is considered instead of $\mathtt{QT}(Q)$) follows immediately from the above results and the construction of $\mathtt{NGQT}(Q)$. □

7.4 Other Queries

The other types of predicates involved in an atomic query can be answered by simply consulting the logic program. For instance, queries of the form $(\exists N,S)\mathtt{frametype}(N,S)$

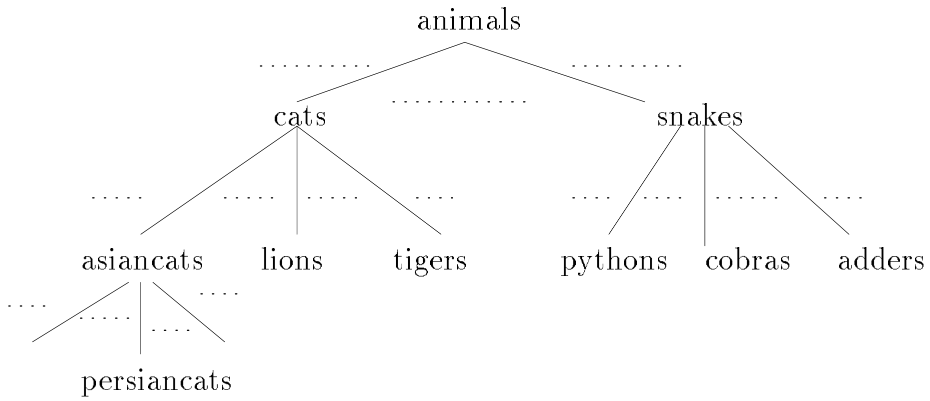

Figure 3: An Animal SMDS

can be handled easily enough because the binary relation `frametype` is stored as a set of unit clauses in the logic program representation. Similarly, queries involving feature-state relations can be computed using the logic program too. Queries involving inter-state relations can be solved by recourse to the existing implementation of those operations. As described earlier, inter-state relationships are domain dependent, and we envisage that the implementation of these relationships will be done in a domain specific manner. Answers to conjunctive queries are considered in Section 9.

All four types of queries specified in the section on the user-request language can be handled within our query processing framework. It is important to note that for change queries (type IV queries), incremental algorithms for computing only the relevant changes need to be devised.

8 Closed SMDSs

Thus far, we have assumed that given a feature f, $\mathtt{RPL}(f)$ is allowed to be any set of features. However, this flexibility may not always be desirable. It can be argued (and we will do so below) that in some situations, the designer of a specific `SMDS` should be forced to consider imposing some restrictions on his/her selection of the function `RPL`. Consider the set of features shown (together with the $\leq$ ordering on them) in Figure 3.

Example 12 Suppose we consider an `SMDS` consisting of the features shown in Figure 3. Then, surely, the feature `python` should not be considered an adequate replacement for the feature `persiancats`.

Suppose now that `cats` is considered an adequate replacement for `persiancats`. What this means is that if a user wishes for information (on one or more media) about`persiancats`, and if the desired type of information is not available, then the system may try to find the desired information w.r.t. cats. From the diagram, it is clear that the feature asiancats is "closer" (intuitively) to the feature persiancats than the feature cats. Thus, if the user says cats is an appropriate replacement for persiancats, then surely asiancats should also be considered an appropriate replacement for persiancats ?

These intuitions motivate the need for imposing axioms on SMDS's.

Definition 11 Suppose $(\{\mathcal{M}_1, \ldots, \mathcal{M}_n\}, \leq, \mathtt{RPL}, \mathtt{SUBST})$ s a structured multimedia database system. This SMDS is said to be *closed* iff the following three conditions hold:

1. $(\forall a, b \in \bigcup_{i=1}^n \mathbf{fe}^i) a \in \mathtt{RPL}(b) \Rightarrow a \leq b$, and
2. $(\forall a, b, c \in \bigcup_{i=1}^n \mathbf{fe}^i) c \leq b \leq a \;\&\; c \in \mathtt{RPL}(a) \Rightarrow c \in \mathtt{RPL}(b) \;\&\; b \in \mathtt{RPL}(a)$.
3. $(\forall s \in \bigcup_{i=1}^n \mathbf{ST}^i)(\forall a, b, c \in \bigcup_{i=1}^n \mathbf{fe}^i) c \leq b \leq a \;\&\; c \in \bigcup_{j=1}^n \lambda^j(s) \;\&\; a \in \bigcup_{j=1}^n \lambda^j(s) \Rightarrow b \in \bigcup_{j=1}^n \lambda^j(s)$. (We assume that if λ^j is not defined on s, then the function call $\lambda^j(s)$ returns the empty set.)

What the third part of the above definition says is that if c is a subfeature of b and b is a subfeature of a, and state s possesses both features a and c, then state s must also possess the "intermediate" feature b. In the context of the Animal example, if we have a picture which is labeled with both cats and persiancats, then this picture must also be labeled with asiancats. Note that this labeling need to be made explicit – it can be assumed to be implicit.

Suppose we consider the set of features $\bigcup_{i=1}^n \mathbf{fe}^i$ associated with an SMDS and consider a single feature f therein. In non-closed SMDS's, finding an appropriate replacement for f may involve searching "laterally" across the feature-graph. On the other hand, in the case of closed SMDS's, the search is restricted to the set of features "below" f. As we shall show later in this section, this restriction, coupled with the "tree" restriction defined below, will guarantee a more compact storage scheme for such closed SMDS's.

Definition 12 A *tree-closed* SMDS (or TC-SMDS, for short) is a closed SMDS such that the set of features in $\bigcup_{i=1}^n \mathbf{fe}^i$ can be represented as a tree[2].

As an example, the Animal SMDS shown in Figure 3 is a tree-SMDS (though, as we have not articulated the RPL relation, we cannot discuss, at this point, whether it is closed). In the rest of this section, we will proceed as follows: section 8.1 defines a new indexing structure for TC-SMDS's and proves that this indexing structure represents a savings (in space) over the indexing structure in the non-closed case. Section 8.2 defines query processing algorithms to operate upon this indexing structure. The price to be paid is that these algorithms, may, in the worst-case, be less efficient than in the case where the indexing structure for the non-closed case is used.

[2] As usual, we assume that all partially ordered sets can be represented as Hasse diagrams, and thus, the poset $(\bigcup_{i=1}^n \mathbf{fe}^i, \leq)$ is tree-closed iff the corresponding Hasse diagram is a tree.

8.1 Indexing Structure for TC-SMDS's

The only difference between the indexing structure for TC-SMDS's and plain SMDS's is in the statelists associated with featurenodes. In the case of arbitrary SMDS's, if state s possesses feature f, then state s occurs in the `statelist` associated with the `featurenode` associated with f. In the case of TC-SMDS's, this situation may not be the case, and we may choose not to store s in the `statelist` associated with the `featurenode` associated with f. Rather, we store (a pointer to) s in the statelist associated with the lowest feature node(s) below f (let us say this featurenode is associated with feature f') which is both in f's replace list, and which occurs in state s. Before formally describing this, we give an illustrative example below.

Example 13 Consider the animal SMDS and consider the path:

$$\texttt{animals} \geq \texttt{cats} \geq \texttt{asiancats} \geq \texttt{persiancats}.$$

Suppose s_1 and s_2 are states (and w.l.o.g., we assume there is only one media-instance involved). Suppose $\lambda(s_1) = \{\texttt{cats}, \texttt{asiancats}, \texttt{persiancats}\}$ and $\lambda(s_2) = \{\texttt{animals}, \texttt{cats}, \texttt{asiancats}\}$. Then, s_1 is only stored in the statelist of `persiancats` and s_2 is only stored in the statelist of `asiancats`.

Suppose s is any state, and

$$f_0 \geq f_1 \geq f_2 \cdots \geq f_k$$

is a path (denoted $\wp$) in the feature tree. If $f_i \notin \lambda^j(s)$ for all $1 \leq j \leq n$, then s occurs in none of the statelists associated with the f_i's. Otherwise, let $\min_\wp(s)$ and $\max_\wp(s)$ denote the smallest and largest integers, ℓ and ℓ' respectively, such that

1. $f_\ell \in \lambda^j(s)$ for some j and
2. $f_{\ell'} \in \lambda^r(s)$ for some r.

Then, by the closure axiom, it follows that for all $\min_\wp(s) \leq w \leq \max_\wp(s)$, $f_w \in \lambda^v(s)$ for some v. In other words, all features "between" $\min_\wp(s)$ and $\max_\wp(s)$ are possessed by state s. In this case, state s will be stored only in the statelist associated with the feature $f_{\max_\wp(s)}$.

Sorted Statelists. It is imperative for the correct functioning of the algorithms described earlier on in the paper that for any given feature f, if the (unique) path, $\wp$, from the root of the feature tree to f is $f_0 \geq f_1 \geq \cdots \geq f_r = f$, then the statelist of f must be sorted in non-increasing order according to $\max_\wp(s)$.

Example 14 Suppose we return to the animal example, and suppose we have three video frames v_1, v_2, v_3 and one audio frame a_1. Suppose the maps λ^1 and λ^2 are as specified below:

$$\begin{aligned}
\lambda^1(v_1) &= \{\texttt{animals}, \texttt{cats}, \texttt{asiancats}, \texttt{lions}\} \\
\lambda^1(v_2) &= \{\texttt{asiancats}, \texttt{persiancats}\} \\
\lambda^1(v_3) &= \{\texttt{snakes}, \texttt{pythons}, \texttt{asiancats}, \texttt{persiancats}\} \\
\lambda^2(a_1) &= \{\texttt{pythons}\}
\end{aligned}$$

Furthermore, suppose the replacelist associated with feature f is the set of all features "below" f according to the $\leq$ ordering. Then the statelists of the features

$$\{\texttt{cats}, \texttt{asiancats}, \texttt{lions}, \texttt{persiancats}, \texttt{snakes}, \texttt{pythons}\}$$

is show below:

`cats`	empty
`asiancats`	$\boxed{\text{ptr to } v_1}$
`lions`	$\boxed{\text{ptr to } v_1}$
`persiancats`	$\boxed{\text{ptr to } v_2}$ followed by $\boxed{\text{ptr to } v_1}$
`snakes`	empty
`pythons`	$\boxed{\text{ptr to } v_3}$ followed by $\boxed{\text{ptr to } a_1}$

Let us examine v_1 which possesses the features `animals, cats, asiancats, lions`. If we examine the path in the feature tree (Figure 3) from `cats` to `persiancats`, then we find that the "lowest" feature along this path which is possessed by v_1 is `asiancats` – hence, v_1 is in the statelist associated with `asiancats`. Likewise, if we examine the path in the feature tree from `cats` to `lions`, then the "lowest" feature along this path which is possessed by v_1 is `lions` – hence, v_1 is in the statelist associated with `lions`. The reader can easily verify that other statelists can be constructed along the same lines.

The only question that remains is the ordering of items within a statelist. Consider the feature `pythons` and the two states $\mathtt{v_3}$ and $\mathtt{a_1}$. The "highest" feature on the path from the root to the node `pythons` which is possessed by $\mathtt{v_3}$ is `snakes`, while in the case of $\mathtt{a_1}$ it is `pythons` – as states associated with a given featurenode's statelist are arranged in non-increasing order according to the 'highest" feature on the path from the root to the node possessed by that state, $\mathtt{v_3}$ appears first in the statelist, followed by $\mathtt{a_1}$.

8.2 Query Processing in TC-SMDS's

The organization of information in the indexing structure for TC-SMDS's is such that the algorithms given in Section 7.2 correctly compute optimal answers to queries in all cases, except one. Intuitively, the reason for this is the following: the only algorithms that may possibly go wrong are those computing membership queries of the form $(\exists t) t \in \texttt{flist}(s)$. When s is ground, then the algorithms in Section 7.2.1 look through the featurelists associated with state s for an answer to the query. As these lists are unchanged in the definition of the indexing structure for TC-SMDS's, the correctness of these algorithms is unaffected.

In the case when both s and t are non-ground, then all that is required is to find any featurelist possessing a non-empty associated state list, and all answers in this case are optimal; hence, this part is unaffected as well.

The only complications that could possibly arise are when t is ground and s is non-ground. In this case, we would like to find a state that has feature f (or failing that, a feature that is in f's replace list and is in s, and such that no other feature in f's replacelist that is strictly "above" this feature, satisfies these criteria). The algorithm give in Section 7.2.1

may return incorrect answers when blindly applied to the new indexing structure in this one case. The following example shows what could happen.

Example 15 Let us revisit the animal example, and suppose we have two documents d_1, d_2. Suppose $\lambda(d_1) = \{\texttt{cats}, \texttt{asiancats}, \texttt{persiancats}\}$ and $\lambda(d_2) = \{\texttt{asiancats}\}$. Then, on the query $(\exists s)\texttt{cats} \in \texttt{flist}(s)$, then algorithm of Section 7.2.1 will return the (incorrect) answer $s = d_2$.

Thus, in order to correctly work with the indexing structure for TC-SMDS's, we only need to modify Case (2) of the algorithms in Section 7.2.1. This modification is described below.

Modification to Case (2) of Algorithm in Section 7.2.1. Given the query $(\exists t)\, t \in \texttt{flist}(s)$ with t ground and s non-ground, proceed as follows:

1. Input: a pointer, temp, pointing to the featurenode associated with t.
2. found := false;
3. If (temp.statelist) is nonempty then
   (a) temp1 := temp.statelist;
   (b) while (temp1 is non-NIL) do
   (c) check if there exists a node X in (temp1.state).flist such that (X.f)=temp; /* i.e. look through the featurelist associated with the state pointed to be temp1 and see if t is a feature occurring in this featurelist. If so, then the state designated by the pointer temp1 has the feature t. */
   (d) If yes, then set found := true, ANS to ((temp.statelist).state).rep and goto step i-4, otherwise set temp1 to (temp1.link) and return to step 3b.
4. If (found) then return ANS and halt.
5. Otherwise, find a maximal element in $\mathsf{DOWN}(t) - \{t\}$ and (recursively) call the above algorithm once for each maximal element in $\mathsf{DOWN}(t) - \{t\}$. If $\mathsf{DOWN}(t) - \{t\} = \emptyset$, then **halt** with failure.

The following example illustrates the working of the above algorithm on the query $(\exists s)\texttt{cats} \in \texttt{flist}(s)$ (w.r.t. the indexing structure described in Example 15). The reader will note immediately that there is only one optimal answer to this query, viz. d_1.

The above algorithm will first examine the statelist of the featurenode `cats`. This is empty, hence, a recursive call will be made to the algorithm with the `asiancats` being considered instead of `cats`. The statelist associated with `asiancats` contains the state d_2. The featurelist associated with d_2 must now be checked to see if (there is a pointer to) `cats` is in it. There is none, hence, d_2 is not an optimal answer. The next element examined is `persiancats`. The *first* element examined in the statelist is d_1. We now examine the featurelist associated with d_1 – a pointer to `cats` is contained therein; hence, d_1 is an optimal answer to the above query.

Note that the above algorithm can be made more efficient by replacing the test

$$\text{while (temp1 is non-NIL) do}$$

in Step 3b of the algorithm by the test

$$\text{while (temp1 is non-NIL) AND (rank(temp1.state)} \not< t\text{) do}$$

where the *rank* of a state is the set of all maximal elements in $\lambda(s)$. We say a set S of features satisfies the condition $S < t$ where t is a single feature if no element of S is larger than t w.r.t. the $\leq$ ordering on features. Thus, in the above "while" statement, rank(temp1.state) returns the maximal elements of the set of all features possessed by the state currently pointed to by temp1. The new test in the while-loop checks if a node has been encountered in the statelist of the current featurenode being looked at such that the state denoted by this node has no "bigger" feature than t – if this is the case, then, as the statelists of featurenodes are topologically ordered according to the $\leq$-ordering, this means that no statenode occuring after the current node in the statelist can possibly have t as a feature, and hence, there is no need to conduct this additional search. The following theorem now follows immediately from the above discussions.

Theorem 3 Suppose Q is a query of the form $(\exists f)(f \in \texttt{flist}(s))$ and SMDS is a TC-SMDS. Then the algorithms in section 7.2.1 when modified as specified above have the following properties:

1. **(Soundness)** If they return a substitution θ, then θ is an optimal answer to query Q.
2. **(Completeness)** Suppose θ is an answer to Q (i.e. $Q'\theta$ is true in SMDS for some $Q' \sqsubseteq Q$). Then there exists a topological sorting $t_1, \ldots, t_r$ of $\mathtt{RPL}(t)$ and a $\sqsubseteq$-maximal query $Q^\star \in \mathsf{DOWN}(Q)$ such that:
   (a) $Q' \sqsubseteq Q^\star \sqsubseteq Q$, and
   (b) $Q^\star\sigma$ is true in SMDS and
   (c) σ is returned by the above algorithms.
3. **(Failure)** The algorithms will return "false" iff there is no answer to query Q.

8.3 Discussion

Consider a TC-SMDS Γ – this is certainly an ordinary SMDS, and hence, we may use either of the two indexing structures proposed in this paper. In this section, we compare these two indexing structures and the algorithms associated with them in order to determine which is more efficient for closed SMDS's.

The first observation is that the indexing structure for TC-SMDS's can be obtained from the indexing structure for arbitrary SMDS's by simply deleting some elements from the statelists associated with featurenodes. Hence, the following proposition may be stated:

Proposition 1 Suppose f is any feature in a TC-`SMDS` Γ. Then the statelist of f according to the indexing structure for TC-`SMDS`'s is a subset of the statelist of f according to the indexing structure for arbitrary `SMDS`'s. $\Box$

As all other things are equal, this means that from the point of view of space, the indexing structure for TC-`SMDS`'s is more compact than that for arbitrary `SMDS`'s.

However, one rarely gets anything for nothing, and hence, there must be a "cost" associated with this additional compactness. What could this cost be? Two thoughts come to mind:

1. **(Loss of Expressive Power?)** TC-`SMDS`'s can only be used to represent certain `SMDS`'s not all. Though closed `SMDS`'s can be converted to TC-`SMDS`'s,there seems to be no obvious way to convert an `SMDS` that does not satisfy the "closure" properties into one that does.
2. **(Diminished Efficiency of Algorithms ?)** As we have observed in the preceding section, all algorithms that work on the original indexing structure for arbitrary `SMDS`'s continue to work in the case of TC-`SMDS`'s with no change – except in the case of membership queries of the form $(\exists s)t \in \texttt{flist}(s)$ where s is a variable and t is ground. First and foremost, we observe that for all membership queries other than ones where s is a variable and t is ground, the **algorithms for TC-`SMDS`'s are more efficient** – the reason for this is that the statelists are smaller in length and hence, it takes less time to examine them. On the other hand, in the one case where s is a variable and t is ground, the algorithms operating on TC-`SMDS`'s are less efficient. This is because they may need to examine the feature lists of states occurring in the statelists of featurenodes.

8.4 An Improved Access Structure

In this section, we will define an access structure which improves on the previous access structures and facilitates processing many kinds of conjunctive queries. The improvement is in maintaining the statelists associated with feature nodes. In the case of both ordinary `SMDS`'s as well as TC-`SMDS`'s, each featurenode has a list of statenodes associated with it – for instance, in Figure 2, the featurenode `taurus` has two statenodes associated with it – viz. $v1$ and $a1$ and there is a pointer PTR1 pointing to this list of statenodes.

In large-scale applications, a large number of states may possess a given feature, and hence, the list of statenodes associated with a featurenode may be very long. It may be useful to break up this list into sub-lists, organized by the `frametype`. Thus, for instance, in the case of the Car Multimedia Database System, the list of statenodes in Figure 1 associated with the feature `taurus` and pointed to by PTR1 may be split up into two sub-lists: an *audio* sublist consisting just of $a1$ and a video-sublist consisting just of $a2$. The advantage of this scheme is that when looking for an audio-state containing the feature `taurus`, then only the audio sublist needs to be examined.

This principle can be extended to TC-`SMDS`'s as well in the obvious way, with the query processing algorithms modified appropriately.

9 Conjunctive Queries

The methods discussed thus far to handle conjunctive queries are very rudimentary. They solve each conjunct individually, and then intersect appropriately. This is a very poor solution because many of the solutions to the individual conjuncts may not satisfy the join conditions generated by solutions to the other conjuncts. Let us consider an example:

Example 16 Let us return to the car multimedia system and consider the query "Is there a picture of a green 1993 Ford Taurus ?" This can be expressed as the query

$$(\exists \mathtt{S})\mathtt{frametype}(\mathtt{S},\mathtt{video}) \,\&\, \mathtt{taurus} \in \mathtt{flist}(\mathtt{S}) \,\&\, \mathtt{color}(\mathtt{taurus},\mathtt{green},\mathtt{S}) \,\&\, \mathtt{model}(\mathtt{taurus},1993,\mathtt{S}).$$

In this query, the subquery $\mathtt{frametype}(\mathtt{S},\mathtt{video})$ may have a million answers (or at least as many answers as there are video frames). However, the subqueries $(\exists \mathtt{S})\mathtt{color}(\mathtt{taurus},\mathtt{green},\mathtt{S})$ and $(\exists \mathtt{S})\mathtt{model}(\mathtt{taurus},1993,\mathtt{S})$ may have a comparatively small number of answers. We may solve either of these first; w.l.o.g., let us solve the former. This may yield a set of answer $\sigma_1,\ldots,\sigma_n$ each of the form $\mathtt{S} = \langle$ some frame$\rangle$. For each of these σ_i's we can now check if the appropriate instance of the latter subquery, i.e. $(\exists \mathtt{S})\mathtt{model}(\mathtt{taurus},1993,\mathtt{S})$, is true. Those of $\sigma_1,\ldots,\sigma_n$ that do satisfy the instance of the latter subquery constitute the set of answers to the conjunctive query $(\exists \mathtt{S})\mathtt{color}(\mathtt{taurus},\mathtt{green},\mathtt{S}) \,\&\, \mathtt{model}(\mathtt{taurus},1993,\mathtt{S})$. At this stage, we need to determine which of these mow satisfy the (ground version of) the subquery $\mathtt{taurus} \in \mathtt{flist}(\mathtt{S})$ and then determine which of those substitutions still left satisfy the condition that `S` be of `frametype video`.

Algorithm: Our general approach to solving conjunctive queries of the form

$$(\exists \mathtt{V}_1,\ldots,\mathtt{V}_\mathtt{n})(\mathtt{A}_1 \,\&\, \ldots \,\&\, \mathtt{A}_\mathtt{m})$$

is the following:

1. Let X be the set of all A_i's that are relations – that is these are atoms that are different from $\mathtt{frametype}(-,-)$ and $LHS\,\{=,\in\}\,\mathtt{flist}(-)$.
2. Let us say that two elements of X are $\sim$-related to each other if they share a common variable symbol, and consider the transitive closure, $\sim^\star$ of this relation. Let $X/\sim^\star$ be the set of all $\sim^\star$-equivalence classes generated by this equivalence relation.
3. For each $\sim^\star$-equivalence class, $Y = \{B_1,\ldots,B_k\}$ of X, do the following:
   (a) Let $SOL(Y)$ be the set of all substitutions satisfying B_1. Set $i = 2$.
   (b) if $i > k$ then computation of $SOL(Y)$ has been completed (for the $\sim^\star$-equivalence class Y). If $SOL(Y')$ still remains to be computed for some $\sim^\star$-equivalence class Y', do so, else goto step 4.
   (c) Otherwise (i.e. if $i \le k$), set $SOLY(Y) = \{\sigma \mid \sigma$ is a solution of $B_i\gamma$ for some $\gamma \in SOL(Y)\}$, i.e. instantiate B_i in many different ways – one for each $\gamma \in SOL(Y)$ and solve each of the resulting atomic subqueries, and accumulate their answers and place these in $SOL(Y)$.

(d) $i := i + 1$. Goto Step 3b.

4. Let Λ be the cross product $\Pi_{Y \in X/\sim^\star} SOL(Y)$. Thus, if $Y_1, \ldots, Y_r$ are all the $\sim^\star$-equivalence classes of Y, then an element of Λ is an r-tuple of substitutions $(\sigma_1, \ldots, \sigma_r)$. As, in any given r-tuple in Λ, no two σ_i's share variables, the union of these substitutions is solvable.

5. Let $SOL = \{\sigma_1 \cup \cdots \cup \sigma_r \mid (\sigma_1, \ldots, \sigma_r) \in \Lambda\}$.

6. For each of the remaining atoms, check, for each $\theta \in SOL$, whether the remaining subquery is true (using the naive method) and if so, return the answer and **halt**.

If we return to Example 16, then the $\sim$ relation consists of:

$$\texttt{color(taurus, green, S)} \sim \texttt{model(taurus, 1993, S)}$$

and there is only one $\sim^\star$-equivalence class, viz. that consisting of the above two atoms.

On the other hand, suppose we consider a more complex query such as:

$$(\exists \texttt{S}_1, \texttt{S}_2, \texttt{S}_3, \texttt{S}_4)\texttt{frametype}(\texttt{S}_1, \texttt{video}) \,\&\, \texttt{frametype}(\texttt{S}_2, \texttt{video}) \,\&\, \texttt{frametype}(\texttt{S}_3, \texttt{video}) \,\&$$

$$\texttt{frametype}(\texttt{S}_4, \texttt{video}) \,\&\, \texttt{p}_1(\texttt{X}, \texttt{a}, \texttt{S}_1) \,\&\, \texttt{p}_2(\texttt{X}, \texttt{Y}, \texttt{S}_2) \,\&\, \texttt{p}_3(\texttt{Y}, \texttt{S}_3) \,\&\, \texttt{p}_4(\texttt{a}, \texttt{S}_4)$$

where $\texttt{a}$ is some constant symbol and $\texttt{X}, \texttt{Y}, \texttt{S}_1, \texttt{S}_2, \texttt{S}_3, \texttt{S}_4$ are all variable symbols. Then the $\sim^\star$-equivalence classes are:

$$\{\texttt{p}_1(\texttt{X}, \texttt{a}, \texttt{S}_1), \texttt{p}_2(\texttt{X}, \texttt{Y}, \texttt{S}_2), \texttt{p}_3(\texttt{Y}, \texttt{S}_3)\},\ \{\texttt{p}_4(\texttt{a}, \texttt{S}_4)\}.$$

10 Timed-Output Queries

In this section, we quickly show how the query language described above can be expanded to handle certain kinds of queries called *timed-output queries* (or TO-queries, for short).

First, let us re-examine the intuitions underlying an ordinary query. A query of the form

$$(\exists \texttt{S}_1, \texttt{S}_2, \ldots, \texttt{S}_\texttt{n})(\exists \texttt{X}_1 \ldots \texttt{X}_\texttt{m})(\texttt{frametype}(\texttt{S}_1, \texttt{type1}) \,\&\, \ldots \texttt{frametype}(\texttt{S}_\texttt{n}, \texttt{typen}) \,\&$$

$$\langle \texttt{conditions} \rangle)$$

asks for instantiations of the $\texttt{S}_\texttt{i}$'s (that range over frametypes, and the $\texttt{X}_\texttt{j}$'s that range over constants (either feature-constants or attribute constants). The idea is that if, say, we get an answer where the instantiations of the $\texttt{S}_\texttt{i}$'s are

$$\texttt{S}_1 = \texttt{s}_1, \ldots, \texttt{S}_\texttt{n} = \texttt{s}_\texttt{n}$$

then this means that each of the frames $\texttt{s}_1, \ldots, \texttt{s}_\texttt{n}$ must be "brought up" as part of the output. Thus, for instance, if $n = 2$, and $\texttt{S}_1$ and $\texttt{S}_2$ are of frametypes `video` and `audio`, respectively, then this says that the video-frame $\texttt{s}_1$ and the audio-frame $\texttt{s}_2$ must be output (starting at the same type) on their respective output devices.

However, the user may wish to see/hear only an initial segment of this output; or, s/he may want to hear the audio in its entirety, but only see the first couple of minutes of the video (e.g. if the user wants to start cooking while listening to the audio....). A *timed query* is a query where all state variables occurring in the query have an associated *time annotation.* Thus, in the case of the above query, we would annotate it thus:

$$(\exists \mathtt{S}_1 : \mathtt{t}_1, \mathtt{S}_2 : \mathtt{t}_2, \ldots, \mathtt{S}_n : \mathtt{t}_n)(\exists \mathtt{X}_1 \ldots \mathtt{X}_m)(\mathtt{frametype}(\mathtt{S}_1, \mathtt{type1}) \,\&\, \ldots \mathtt{frametype}(\mathtt{S}_n, \mathtt{typen}) \,\&\, \langle \mathit{conditions} \rangle))$$

where the $\mathtt{t_i}$'s are non-negative integers. Intuitively, if we consider the situation where $n = 2$, and $\mathtt{S}_1$ and $\mathtt{S}_2$ are of frametypes `video` and `audio`, respectively, and $\mathtt{t}_1 = 3$ and $\mathtt{t}_2 = 25$, and if

$$\mathtt{S}_1 = \mathtt{s}_1, \mathtt{S}_2 = \mathtt{s}_2$$

is a solution, then this says that the video-frame $\mathtt{s}_1$ must be on for 3 units of time, while the audio-frame must be on for 25 units of time. This is quite straightforward to implement. Note that these time-annotations do not change the meaning of the query; rather, they specify how the output is to presented to the user.

In future work, we will study in greater detail, the structure of these temporal annotations so as to be able to express more complex desiderata/constraints on the temporal presentation of the output frames.

11 Constrained Queries

In any query Q to an `SMDS`, certain variables range over *states* – in particular, any variable, `S`, that occurs in an atom of the form $\mathtt{frametype}(\mathtt{S}, -)$ is a state variable, ranging over states. When answering the query Q, the state variables typically get "instantiated" to some state-constants. The idea is that the physical objects (e.g. picture, sound-frame, video-frame, text file, etc.) represented by these named states are to be "brought up" or "output" on the appropriate output device. Thus, in the query

$$(\exists \mathrm{S}_1, \mathrm{S}_2)\mathrm{frametype}(\mathrm{S}_1, \mathrm{video}) \,\&\, \mathrm{frametype}(\mathrm{S}_2, \mathrm{document}) \,\&\, \mathrm{taurus} \in \mathrm{flist}(\mathrm{S}_1) \,\&\, \mathrm{taurus} \in \mathrm{flist}(\mathrm{S}_2)$$

a request is being made to find a video frame of the Taurus, and documentation concerning the Taurus. The substitution $\mathtt{S}_1 = \mathtt{v1}, \mathtt{S}_2 = \mathtt{d1}$ (w.r.t. our familiar car example) is an answer to this query. However, this answer may not always be satisfactory – for example:

1. (Temporal Relationships) the user may wish to watch the video frame `v1` first, followed by the document `d1`,or
2. (Spatial Relationships) the user may wish to see two windows – one with the document on it, and the other with the video on it; he may also wish to specify that the "top window" represents the document and that the bottom window contain the video, etc.

In this section, we present a somewhat more general picture of how state variables may be used to define constraints over different types of domains – we will start by showing how state variables may be used to define a broader class of timed-output queries, and then we will show how spatial relationships may be captured using such constraints as well. Finally, we will generalize this to an abstract formalism ranging over different constraint domains [17, 18, 28, 21, 4].

11.1 The Temporal Domain

Suppose we have a query Q and suppose θ is an answer substitution to query Q. We use θ_s to denote the restriction of θ to state variables – i.e. all equations in θ that are not of the form $\mathtt{S} = -$ are deleted from θ. To express *temporal relationships*, we may define a very simple temporal constraint language over the integers or the reals (as deemed appropriate). We do this as follows:

1. For each state variable `S`, we have two constraint-variables `start(S)` and `end(S)`.

2. The language contains the non-negative real numbers as constants.

3. The contains the operator symbol $+$.

4. The relations in the language are $=$ and $\leq$.

Constraints can now expressed easily in the form $t\langle \mathtt{OP} \rangle t'$ where t, t' are either constraint-variables or real numbers (or summations, thereof). Thus, for instance, the user may enhance the query

$(\exists \mathrm{S}_1, \mathrm{S}_2)\mathrm{frametype}(\mathrm{S}_1, \mathrm{video})\,\&\,\mathrm{frametype}(\mathrm{S}_2, \mathrm{document})\,\&\,\mathrm{taurus} \in \mathrm{flist}(\mathrm{S}_1)\,\&\,\mathrm{taurus} \in \mathrm{flist}(\mathrm{S}_2)$

above by applying to it, the constraint Ξ given as:

$$\mathtt{start}(\mathtt{S}_2) \geq \mathtt{end}(\mathtt{S}_1) + \mathtt{0.001}.$$

This would specify that he wishes to see the document only after seeing the video. Thus, in general, a conjunction of one or more constraint may be appended to queries to specify the temporal relationships between the outputs (on media or output devices) generated by a query. It is easy to see that all the queries in Section 10 can easily be expressed using such queries.

11.2 The Spatial Domain

Let us now consider the spatial domain. Suppose there is only one spatial output device (such as an ordinary video display screen). Then the output may need to be "laid out" on the screen which can be viewed as a two dimensional cartesian $(m \times n)$ fragment of the Cartesian plane. Relationships like `below, above, overlapping`, etc. may now be defined easily enough using methods defined by Sistla et. al.[26].

As in the case of temporal domains, *constraints* may be used to express spatial relationships between objects being displayed. Thus, for instance, in the case of the query $Q =$

$(\exists \mathrm{S}_1, \mathrm{S}_2)\mathrm{frametype}(\mathrm{S}_1, \mathrm{video})\,\&\,\mathrm{frametype}(\mathrm{S}_2, \mathrm{document})\,\&\,\mathrm{taurus} \in \mathrm{flist}(\mathrm{S}_1)\,\&\,\mathrm{taurus} \in \mathrm{flist}(\mathrm{S}_2)$

we may append the constraint Ξ' given by:

$$ABOVE(S_1, S_2)$$

to denote that $\mathtt{S_1}$ must be "above" $\mathtt{S_2}$. In a similar vein, if the user not only wants the video output to be above S_1, but also wants the associated window to be twice the area of that associated with the document output, then this too is an additional constraint Ξ'' given by

$$\mathtt{AREA(S_1)} = 2 \times (\mathtt{S_2})$$

where $\mathtt{AREA(S_1)} + \mathtt{AREA(S_2)} = n \times m$.

11.3 Generalization to Arbitrary Domains

In this section, we *abstract* from the above two examples involving temporal and spatial relationships between answer substitutions (when restricted to state variables).

We begin with a domain D of objects, called the domain of discourse. The function space generated by D consists of the set of all functions over D, as well as the set of all functionals over functions of D. A *constraint domain* defined by D, denoted Σ_D is a pair (F, R) where F is a subset of the function space generated by D, and R is a set of relations over F. The notion of constraint domain was originally due to Jaffar and Lassez [17]¿

Intuitively, in the case of the temporal domain, D may be thought of as the integers or the reals (depending upon whether we are interested in continuous or discrete time), F may be thought of as consisting of just one function, $+$, and R may be thought of as containing just the two relations $\leq$ and $=$.

In the case of the 2-dimensional spatial domain, D may be thought of as the set of pairs (x, y) for $x \in \{0, \ldots, m, \perp\}$, $y \in \{0, \ldots, n, \perp\}$. $\perp$ is a special symbol whose meaning will become clear soon. The set F may consist of $+$ and $\times$, and R may consist of relations such as $\leq, =, \mathtt{ABOVE}, \mathtt{BELOW}, \mathtt{LEFT}, \mathtt{RIGHT}$, etc.

A *state constraint language* based on the constraint domain Σ_D is denoted $\mathtt{CL}(\Sigma_D)$ and is defined as follows:

1. For each state variable `S`, we have a set of constraint-variable symbols called `state-related aspect variables` (SRA-variables, for short). SRA-variables range over D. For example, in the temporal domain discussed earlier, `start(S)`, `end(S)` are SRA-variables.
2. A *term* is either:
   (a) an SRA-variable or
   (b) a member of D, or
   (c) of the form $\mathtt{f(t_1, \ldots, t_n)}$ where $\mathtt{t_1, \ldots, t_n}$ are terms, and `f` is an n-ary function on D.
3. an *atomic constraint expression* is an expression of the from $R(t_1, \ldots, t_n)$ where R is an n-ary relation in Σ_D, and $t_1, \ldots, t_n$ are terms.

4. a *constraint expression* is a conjunction or disjunction of atomic constraint expressions.

Thus, in the temporal domain, $\mathtt{end(S_1)} + \mathtt{0.001}$ is an atomic constraint expression. Similarly, the expression $\mathtt{AREA(S_1)}$ is an atomic constraint expression in the spatial domain – `AREA(-)` always returns a pair of the form $(\langle\text{realnumber}\rangle, \perp)$ which denotes the real-number in the first argument.

The definition of a *query* may now be extended to a *generalized constraint query* as follows. Suppose $\Sigma_{D_1}, \ldots, \Sigma_{D_k}$ are constraint domains over $D_1, \ldots, D_k$, respectively. If Q is a query (as defined earlier) and if $\Xi_1, \ldots, \Xi_k$ are constraint expressions over $\Sigma_{D_1}, \ldots, \Sigma_{D_k}$, respectively, then

$$Q \,\&\, \Xi_1 \,\&\, \ldots \,\&\, \Xi_k$$

is a constrained query.

For example, suppose we wish to return to the query $Q =$

$$(\exists \mathrm{S}_1, \mathrm{S}_2)\mathrm{frametype}(\mathrm{S}_1, \mathrm{video}) \,\&\, \mathrm{frametype}(\mathrm{S}_2, \mathrm{document}) \,\&\, \mathrm{taurus} \in \mathrm{flist}(\mathrm{S}_1) \,\&\, \mathrm{taurus} \in \mathrm{flist}(\mathrm{S}_2)$$

that we considered earlier. Recall that $\mathtt{S_1} = \mathtt{v1}, \mathtt{S_2} = \mathtt{d1}$ is an answer to this query. Then:

1. (Temporal Relationships) If the user wishes to watch the video frame first, followed by the document, then he may express this as the query:

$$\mathtt{Q} \,\&\, \mathtt{start(S_2)} \leq \mathtt{end(S_1)} + \mathtt{0.001}.$$

2. (Spatial Relationships) If the user wishes to see two windows – one with the document on it, and the other with the video on it and with the video in the top window, and the text in the lower window, then he may specify this as:

$$\mathtt{Q} \,\&\, \mathtt{ABOVE(S_1, S_2)}.$$

Finally, suppose a query involves three state variables – $\mathtt{S_1, S_2, S_3}$ ranging over video, document and document, respectively, and that the user would like to see the video on top, while the two documents are below it as shown in Figure 4. Then this may be specified as the query:

$$\mathtt{Q} \,\&\, \mathtt{ABOVE(S_1, S_2)} \,\&\, \mathtt{ABOVE(S_1, S_3)} \,\&\, \mathtt{LEFT(S_2, S_3)}.$$

11.4 Constrained Presentations

Before concluding this section, we observe that just as we may wish to express constraints on queries that guarantee an acceptable temporal and/or spatial layout of a media-event that satisfies the query(ies), we may also wish to express constraints on *multimedia-presentations*. Expressing constraints on multimedia-presentations can be achieved by expressing constraints on multimedia-specifications, which, after all, are what we use to generate multimedia-presentations anyway.

Thus, we may define a *constrained multimedia-specification* as follows:

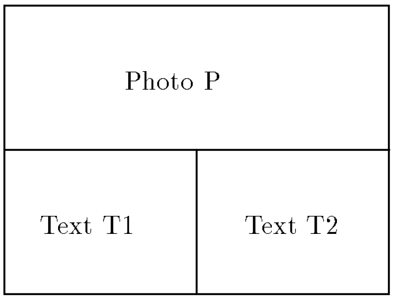

Figure 4: A Possible Spatial Arrangement

1. $[\langle name \rangle : Q_1, \ldots, Q_n] : \langle Constraint \rangle$ is a constrained multimedia-specification.
2. If Psi_i, $1 \leq i \leq m$ is either multimedia-specification or a constrained multimedia-specification, then the concatenation, $\Psi_1, \Psi_2, \ldots, \Psi_m$ is a constrained multimedia-specification.

The only important thing about constrained multimedia-specifications is the first clause above. Intuitively, it says that the *multimedia presentation* generated by the multimedia-specification $Q_1, \ldots, Q_n$ must satisfy the constraints. To see what this means, consider the following example.

Example 17 (Car Example Revisited) Let us reconsider the multimedia presentation of Example 6 which is generated by the following queries:

1. $\mathtt{Q}_1 = (\exists S_1, S_2)(\mathtt{frametype}(S_1, audio) \,\&\, \mathtt{frametype}(S_2, video) \,\&\, \{taurus\} = \mathtt{flist}(S_1)$ $\&\, \{taurus\} = \mathtt{flist}(S_2)$. This query is satisfied by the media event

$$\mathbf{me_1} = (\mathtt{v1}, \mathtt{a1}, \mathtt{nothing}).$$

2. $\mathtt{Q}_2 = (\exists S_1) \,\&\, \mathtt{frametype}(S_1, video) \,\&\, \mathtt{flist}(S_1) = \{taurus, front\}$. An answer to this query is the media-event

$$\mathbf{me_2^1} = (\mathtt{v2}, \mathtt{nothing}, \mathtt{nothing})$$

Thus, $\mathbf{me_1}, \mathbf{me_2^1}$ is a multimedia presentation that satisfies the multimedia specification (Q_1, Q_2). Let us "name" this multimedia specification $S1$. Then

$$[S1 : Q_1, Q_2] : \mathtt{start}(\mathtt{S}_1) = \,\&\, \mathtt{end}(\mathtt{S}_1) = 5$$

says that the multimedia-presentation $\mathbf{me_1}, \mathbf{me_2^1}$ generated by the multimedia specification (Q_1, Q_2) must satisfy the constraints: "Start at time 0 and end at time 5)." Suppose now that the durations of videos/audios involved in this multimedia presentation are as follows:

VIDEO/AUDIO	DURATION
$v1$	3
$a1$	1
$v2$	2

Then it is feasible to satisfy constraints in the above query.

On the other hand, suppose, in the preceding example, that the duration of $v2$ is 3 time units. Then there are exactly two ways of satisfying the query – either we playback the audio/video at a "faster" rate than originally envisaged, or we may cut a part of the video $v2$ off when the five time units are "up." For now, we make no commitment on how to do this. Weiss et. al. [29] have developed special operators that may be used to select one or more of these options. In Appendix A, we show how all the composition operators specified by Weiss et. al. [29] may be expressed within our query language, thus making it possible to handle "stretched" playback, "shrunk/fast-forwarded" playback, etc.

12 Implementation

The notion of *media-instances* described in this paper has led to a prototype implementation at the University of Maryland. The implementation is written in C and runs on SUN/Sparc workstation. At this point in time, the implementation can execute all the types of queries described in this paper (relaxed queries are not included in the current implementation, and support for them is being currently built in). In addition to the basic features and states that occur in multimedia database systems, the current implementation:

- **supports inference chaining** – "basic concepts" can be described using the features and states, while "derived concepts" can be defined in terms of the basic concepts. This process can be iterated many times.
- **supports reasoning with uncertainty and/or goodness measures.** For instance, certain pictures of Clinton may be "better" than others. Our system has the capability of finding pictures of the desired quality.

The current implementation is part of a much broader implementation effort to integrate multiple heterogeneous databases and multimedia databases [27, 28, 1, 21]. The overall systems architecture is shown in Figure 5. The end-user accesses a host of heterogeneous databases, data structures and multimedia databases using the mediator framework. Currently, our system can integrate the following types of data:

- pictorial data,
- video data,
- text data (e.g. USA Today newswires),

- relational data,
- spatial data (such as news-wires).

We briefly describe each part of the implementation. The entire implementation on the Sparc platform is about 35,000 lines of C code. A version for the 486-based PC platform has also been developed.

Mediator: The concept of a mediator is due to Gio Wiederhold [30, 31]. Intuitively, a mediator is a program that integrates information from multiple databases and/or multiple data structures. Till recently, most mediators were implemented as very large C programs. In [27, 21], we have proposed a uniform language to implement mediators. A mediator developer would specify a mediator for a specific application in this language.

Mediator Development Environment: In order to assist the mediator develop/author, we have provided (and in some cases, are in the process of providing) a number of tools that the mediator developer may interact with. These tools are interactive and assist the mediator author/developer in specifying the mediation rules/strategy.

Mediator Compiler: In [27, 21], we have shown how compilers for the mediator language may be implemented. The main advantage of such languages is that they access existing databases and software packages by using the functions and procedures defined already in those DBMSs, using a facility similar to the well-understood remote procedure call. We have developed a compiler for this language (though the process of optimizing the compiler and adding new facilities to it is a continuing, ongoing process). The mediator authored for a specific application will be processed by the mediator compiler.

Graphical/Voice User Interface: By the end of July 1994, we expect to complete development of a graphical user interface (under Windows for the PC platform, and Xwindows on the Sparc) so that the end-user of a mediated system developed using the mediator development environment may ask his/her queries by using a mouse and/or selecting items from a menu. A voice-based analog of this graphical user interface is just being addressed, in conjunction with Nynex Corp.

Using our system, we have defined an example multimedia database (slightly over 6 Mbytes of data) that contains: (1) pictorial data (photographs from the inauguration of Bill Clinton) in the form of GIF files, (2) text data obtained from electronic versions of the newspaper USA Today, and (3) relational data defining relations. Note that a wide variety of databases can be defined within our system, and that this particular database is only one example of a database that can be defined within our system. We will illustrate some features of our implementation using the above multimedia database example.

The relational database contains relations of the form: `spouse(X,Y)` – `X` is the spouse of `Y`, `rank(X,Y,From,To` saying that person `X` held rank `Y` from time `From` to time `To`.

We can now process queries of the form:

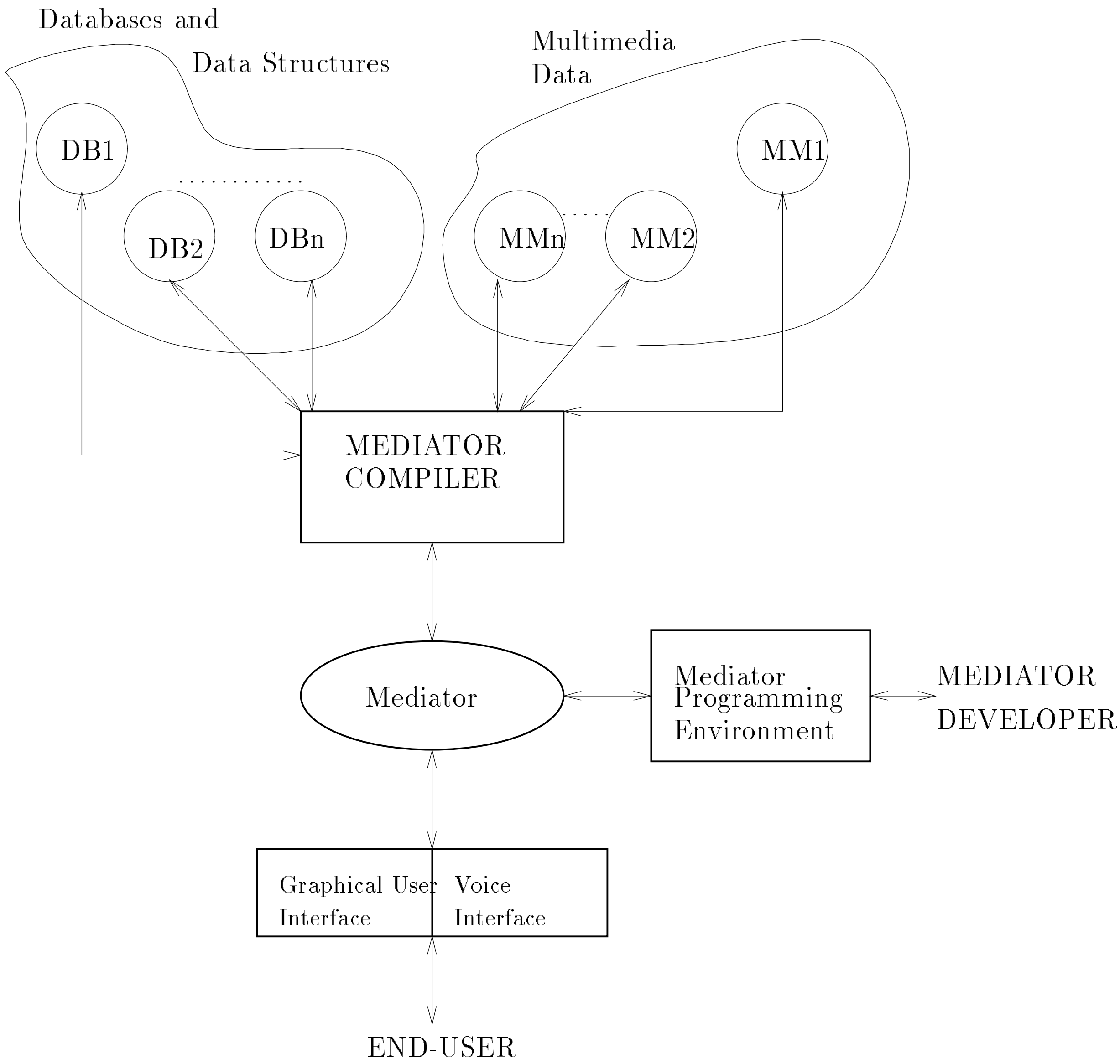

Figure 5: Overall System Architecture

1. *Find all pictures, with George Bush, of the spouse of a person whose taxes have been reported in news articles.* This corresponds to the query:

$$(\exists \text{Article}, \text{Spouse}, \text{Picture}, \text{Person}, \text{Spouse})\text{feature}(\text{"taxes"}, \text{Article})\&$$

$$\text{feature}(\text{Person}, \text{Article})\&\text{frametype}(\text{Article}, \text{"news"}) \,\&\, \text{frametype}(\text{Picture}, \text{"picture"}) \,\&$$

$$\text{spouse}(\text{Person}, \text{Spouse})\&\text{feature}(\text{"GeorgeBush"}, \text{Picture})\&\text{feature}(\text{Spouse}, \text{Picture}).$$

This query looks at textual news data (via the `frametype(Article,"news")` calls, at pictorial data (via the `feature` predicate), and `relational` data (via the `spouse` predicate). An answer to this query consists of the substitution

$$\{\text{Article} =' \text{a9}', \text{Picture} = \text{"INAU08.GIF"}, \text{Person} = \text{"HilaryClinton"}, \text{Spouse} = \text{"BillClinton"}\}.$$

Once this answer has been computed, the system asks the user if s/he wishes to see the picture stored in "INAU08.GIF". If the user says yes, then this picture is brought up on the screen. It then asks the user if s/he wishes to see the news article in file "a9". If the user says yes, that too is brought up on the screen. The entire query takes 360 milliseconds to execute.

2. *Find all pictures of Bill Clinton where the "quality" of the picture of Bill Clinton exceeds 75% and where he is pictured with a statue of Lincoln.* This can be expressed as follows:

$$(\exists \text{Picture})(\text{frametype}(\text{Picture}, \text{"picture"}) \,\&\, \text{feature}(\text{"BillClinton"}, \text{Picture}) : 0.75 \,\&$$

$$\text{feature}(\text{"LincolnStatue"}, \text{Picture})).$$

In this query, the atom `feature("Bill Clinton",Picture)` is *annotated* with the real number 0.75. Any picture in which Bill Clinton appears with over 75% "goodness" is considered to satisfy that annotated atom `feature("Bill Clinton",Picture):0.75`. When a `feature` atom is not annotated, then this implicitly represents an annotation of 1. There is a rich and well-understood theory of such annotations [7, 21, 19, 27] that the interested reader may pursue further if he so desires.

In the above query, the system quickly comes back with the answer substitution `Picture = 'INAU06.GIF')`. It then asks the user whether s/he would like to see the picture. If the user says "yes", then the picture is presented to the user on the screen, otherwise, the system asks the user if s/he would like to see another picture. The entire CPU time taken to process this query is 180 milliseconds.

3. *Find all pictures of the spouse of a US President with a UN Secretary General.* This can be expressed as follows:

$$(\exists \text{P1}, \text{P2}, \text{P3}, \text{T1}, \text{T2}, \text{T3}, \text{T4Picture})(\text{frametype}(\text{Picture}, \text{"picture"}) \,\&$$

$$\text{feature}(\text{P1}, \text{Picture}) \,\&\, \text{feature}(\text{P2}, \text{Picture}) \,\&$$

$$\text{spouse}(\text{P1}, \text{P3}) \,\&\, \text{rank}(\text{P3}, \text{"President"}, \text{T1}, \text{T2}) \,\&\, \text{rank}(\text{P2}, \text{"UNSecretaryGeneral"}, \text{T3}, \text{T4}) \,\&\,.$$

When executed, this query returned the answer substitution

$$\{\text{Picture} = \text{"perez.gif"}, \text{P1} = \text{"HilaryClinton"}, \text{P2} = \text{"PerezdeCuellar"}$$

P3 = "BillClinton"}.

The other variables are not shown above (as they are not important). The system then asks the user whether s/he would like to see the picture "perez.gif". If the user says "yes", then this picture is brought up on the screen; otherwise the system halts. The entire CPU time was 410 milliseconds.

13 Related Work

There is now a great deal of ongoing work on multimedia systems, both within the database community, as well as outside it. All of these works, without exception, deal with integration of *specific types of media data*; for example, there are systems that integrate certain compressed video-representation schemes with other compressed audio-representation schemes. However, we are aware of no single theoretical framework for integrating multimedia data that "abstracts" away the essential features of diverse media and data representations, making it possible, in principle, to integrate multimedia data without knowing in advance, what the structure of this data might be.

13.1 Grosky

Grosky's work [12, 13] is close, in spirit, to our work in many respects. Grosky [12] proposed a version of SQL that could be used to query pictorial databases using feature-based approaches. In [13], he describes *complex* features – intuitively, complex features have sub-features – in connection with our work, given a feature f, the set of features "below" f (w.r.t. the notion of below-ness defined by the ordering $\leq$ on features) may be thought of as the sub-features associated with f. Our work may be viewed as an extension of Grosky's work in the following ways: we can deal with diverse kinds of media data, our use of the ordering $\leq$ is to define a mathematically solid way of "relaxing" queries and define optimal answers, and our implementation can deal with many aspects (such as uncertainty, time, and access to multiple data structures and databases) that is novel. In fairness, Grosky's framework [12] can deal with integrating relational and pictorial information, though he does not address schema mismatches which have been addressed in our implementation of hybrid knowledge bases [21].

13.2 Berson, et. al.

Berson et. al. [6] study how to distribute multimedia objects (bodies of data) across a network, given that the multimedia objects occupy large bandwidths, while traditional disks have relatively small bandwidths. Our framework is complementary to theirs in the following way: in the definition of our field `framerep` in the type `statenode` (cf. Figure 1), `framerep` points to the physical location of the multimedia data. This data itself could be further broken up into chunks as suggested by Berson et. al. [6]. This would facilitate effective bandwidth utilization of this data when communicating information across the network.

13.3 Gibbs et. al.

Gibbs et. al. [10] study how stream-based temporal multimedia data may be modeled using object based methods. Our work is related to theirs upto a point, and then diverges. According to their framework, an "artifact" is an object produced in a specific medium; for instance, prints, TV-news-programs, and music recordings are all artifacts. "Media objects" are digital representations of artifacts. In terms of our framework, the set of media-objects in an `SMDS` is essentially the same as $\bigcup_{i=1}^{n} \mathbf{ST}^{i}$ using the notation developed in this paper. Their notion of a "media descriptor" is also identical to our notion of a `frametype`. Their notion of a "media element" is similar to our notion of a media-event. In their framework, a "timed stream" is a finite sequence of tuples (e_i, s_i, d_i) where e_i is a media element, s_i is its "start" time, and d_i is its duration. Gibbs et. al. study various issues related to time-streams. In contrast to Gibbs et. al.[10], we do *not* study different types of time streams – rather, we develop a query language for querying multimedia data and *show how this query language can be used to generate media-presentations (or time-streams in the Gibbs et. al. [10] terminology)* – something Gibbs et. al. do not address. Our work and Gibbs et. al.'s work may come together in the following way: an end-user gives a media-specification (sequence of queries) to an `SMDS` which generates a media-presentation using the techniques described in this paper. This media-presentation may then be viewed as a certain kind of timed-stream that may be delivered to the output devices using the techniques of Gibbs et. al. [10].

13.4 Weiss et. al.

Weiss et. al. [29] develop a set of operations that may be used to compose multiple aspects of multimedia presentations. They correctly point out that such operations form the "kernel" of basic operations needed in creating effective multimedia presentations. It is important that any language for multimedia databases should, likewise, satisfy these criteria.

To see how our work is related to that of Weiss et. al. [29], we observe that we have given a formal mathematical definition of a multimedia presentation and a multimedia specification (that generates a multimedia presentation). Using these definitions, it turns out that *each and every composition operator that Weiss et. al. specify [29, p.144] can be expressed as constrained queries* within our query language. This indicates that our query language has the required expressive power for composing multimedia data together to generate effective multimedia presentations. Appendix A contains, for each algebraic composition operation given by Weiss et. al., the multimedia specification in our language that generates this presentation.

13.5 Iino et. al.

Iino et. al. [16] study methods for spatial and temporal reasoning in multimedia systems. They propose a very elegant Petri Net model for object composition. Their work is related to one aspect of our work, viz. the "constraint" part of constrained queries defined in Section 11. In connection with spatio-temporal reasoning, we observe that there are numerous models of time, as well as numerous models of space. Iino et. al. use the well-known formulation of time due to Allen [2] to synchronize multimedia events. In contrast, by avoiding a commitment to any single model of time/temporal reasoning, and instead using generalized constraints in our query language, we are able, in principle, to express constraints over different models of time, and different models of space. Of course, we also provide numerous facilities not provided by Iino et. al. [16] such as query languages for multimedia data, indexing structures, formal soundness and completeness results, etc.

13.6 Woelk and Kim

Woelk and Kim [32] have developed an object-oriented implementation of multimedia capabilities on top of the ORION object-oriented database system. A key feature of their work is that it is closely tied to an object-oriented implementation. In contrast, the query language defined in this paper can be implemented using the techniques described in this paper, or on a relational database management system (cf. Brink[8]) or on top of an object-oriented system. The important point is that our definition of an `SMDS` avoids any commitment to a single implementation paradigm. More importantly, our definition of a media-instance is very broad, acting as an abstraction of the various kinds of media that we are all familiar with. The definition of *media-presentation* is mathematically precise, depends solely upon media instances, and exists independently of any implementation. The correspondence of media-presentations and query sequences is a new aspect of our work. The utility of constraints for multimedia applications is another new and unique feature of our work, vis a vis that of Woelk and Kim [32]. The notion of a *structured* multimedia systems is unique to our paper – it is very important to note that *$a \leq b$ does not mean that a inherits properties of b* – to the contrary, a `dashboard` does not inherit most properties of `car`s even though it (usually) resides within one ! This notion of structure enables builders of such systems to formally articulate methods of relaxing queries – a flexibility that may be very useful in many applications. The user/multimedia system developer who wants queries to be rigid can simply take $\leq$ to be the identity relationship and `SUBST` to be empty.

In short, this paper presents a formal mathematical model of what constitutes a media-instance. The development of such a mathematical foundation is vitally necessary – it sets up the basic yardsticks against which the correctness of algorithms and implementations can be evaluated. To our knowledge, our paper represents the first effort in this direction, presenting the first definitions of soundness and completeness of multimedia computation algorithms and then establishing algorithms that are provably sound and complete. Last, but not least, the mathematical model presented in this paper is not sterile – it has been implemented in a working system.

13.7 Gupta et. al.

Gupta, Weymouth and Jain [14] have developed a model, called the VIMSYS model, for querying a pictorial database. VIMSYS is a hybrid of a functional model and an object oriented model. This model allows users to view image data in four different "planes" corresponding to different areas of interest. In contrast to the work of Jain and his group, our work attempts to develop *general purpose* methods of integrating multiple types of media data, not just image data. For instance, our system can integrate image and pictorial data, news-wires, relational databases including different relational systems and possible schema mismatches even within a system, spatial data structures (such as quadtrees) and numerical computations. Our work could benefit from the work of Jain et. al. [14] in the following way: their algorithms are highly optimized for the image domain. Thus for this domain, we could "hook" our mediator onto the system of Jain et. al., and access image databases through Jain et. al.'s program. Jain et. al.'s system could be accessed by our mediator framework and this would allow us to integrate their functionality with ours, accessing heterogeneous relational data, raw and/or structured text new-wires, pictorial data, spatial data, and vide-data.

13.8 Other Efforts

Oomoto and Tanaka [24] have defined a video-based object oriented data model, OVID. They take pieces of video, identify meaningful features in them and link these features. Our work deals with integrating multiple media and provides a unified query language and indexing structures to access the resulting integration. Hence, one such media-instance we could integrate is the OVID system, though our framework is general enough to integrate many other media (which OVID cannot). In a similar vein, Arman et. al [3] develop techniques to create large video databases by processing incoming video-data so as to identify features and set up access structures. Cardenas, et. al. [9] have developed a query language called PICQUERY+ for querying certain kinds of federated multimedia systems. In contrast to their work, our notion of a media-instance is very general and captures, as special cases, many structures (e.g. documents, audio, etc.) that their framework does not appear to capture. Hence, our framework can integrate far more diverse structures than that of [9].

14 Conclusions

There is now intense interest in multimedia systems. These interests span across vast areas in computer science including, but not limited to: computer networks, databases, distributed computing, data compression, document processing, user interfaces, computer graphics, pattern recognition and artificial intelligence. In the long run, we expect that intelligent problem-solving systems will access information stored in a variety of formats, on a wide variety of media. Our work focuses on the need for unified framework to reason across these multiple domains.

This paper makes the following contributions. First, we have formally specified the notion of a *media-instance* – intuitively, a media-instance consists of a set of "states" (e.g. video-clips, audio-tracks, etc. may be viewed as states), a set of "features" (i.e. objects occurring in those states), as well as properties of these features, and relationships between these features. The *key* contribution of this paper is in the definition of a *structured* multimedia database system. In this case, we have defined two importance concepts – that of *acceptability* of an alternative answer based on the functions `RPL` and `SUBST`. This is particularly useful because when a query is not satisfiable, we may not want the system to simply return the answer "no." Instead, the system should be more cooperative, returning an answer to a slightly weaker query. We have shown how such a notion of a "weaker" query can be expressed (as the `RPL` and `SUBST` functions in an `SMDS` induce the definition of the $\sqsubseteq$-ordering). We have developed query processing algorithms, and showed that they are sound, complete and terminating, and have discussed the complexity of these algorithms.

Furthermore, we have discussed how *constrained* queries (and sequences of constrained queries) can be used to generate multimedia presentations that satisfy various logical constraints, as well as various temporal and spatial constraints. These are very critical for real-world applications where an end-user of such a multimedia database system may not only want to generate a multimedia presentation that satisfies various logical requirements, but where the presentation itself adheres to certain "output" formats that the user desires. These output formats may include requests to show certain video-clips in slow motion, to synchronize certain audio-clips, video-clips and document data, and/or to layout certain outputs in windows that reflect a spatial arrangement the user feels comfortable with.

The theoretical algorithms and definitions provided in this paper are not sterile – an implementation based on this theory exists at the University of Maryland. This implementation is built on top of an already existing, successful theory[21, 27, 1] and implementation of mediators for integrating heterogeneous databases and data structures. That implementation has been successfully used for two large-scale real-world applications by organizations outside the University of Maryland [5, 15].

Future work must focus on two issues: conjunctive query optimization in multimedia databases needs to be addressed in greater detail. It is critically needed in any realistic multimedia database system. Second, the issue of updates to multimedia database systems needs to be carefully addressed.

References

[1] S. Adalı and V.S. Subrahmanian. (1993) Amalgamating Knowledge Bases, III: Distributed Mediators, accepted for publication in: Intl. Journal of Intelligent Cooperative Information Systems.

[2] J.F. Allen. (1983) *Maintaining Knowledge About Temporal Intervals*, Communications of the ACM, Vol. 26, No. 11, pps 832–843.

[3] F. Arman, A. Hsu and M. Chiu. (1993) *Image Processing on Compressed Data for Large Video Databases*, First ACM Intl. Conf. on Multimedia, pps 267–272.

[4] C. Bell, A. Nerode, R. Ng and V.S. Subrahmanian. (1992) *Mixed Integer Programming Methods for Computing Nonmonotonic Deductive Databases*, accepted for publication in Journal of the ACM.

[5] J. Benton and V.S. Subrahmanian. (1993) *Using Hybrid Knowledge Bases for Missile Siting Problems*, Proc. 1994 Conf. on Artificial Intelligence Applications, IEEE Computer Society.

[6] S. Berson, S. Ghandeharizadeh, R. Muntz and X. Ju. (1994) *Staggered Striping in Multimedia Information Systems*, Proc. 1994 ACM SIGMOD Conf. on Management of Data, pps 79–90.

[7] H. Blair and V.S. Subrahmanian. *Paraconsistent Logic Programming,* Theoretical Computer Science, Vol. 68, pp 135–154.

[8] A. Brink. (1994) M.S. Thesis, George Mason Univ., in preparation.

[9] A. F. Cardenas, I.T. Ieong, R. Barket, R. K. Taira and C.M. Breant. (1993) *The Knowledge-Based Object-Oriented PICQUERY+ Language*, IEEE Trans. on Knowledge and Data Engineering, 5, 4, pps 644–657.

[10] S. Gibbs, C. Breiteneder and D. Tsichritzis. (1994) *Data Modeling of Time-Based Media*, Proc. 1994 ACM SIGMOD Conf. on Management of Data, pps 91–102.

[11] Y. Gong, H. Zhang, H.C. Chuan and M. Sakauchi. (1994) *An Image Database System with Content Capturing and Fast Image Indexing Abilities*, Proc. 1994 Intl. Conf. on Multimedia Computing and Systems, pps 121-130, IEEE Press.

[12] W. Grosky. (1984) *Toward a Data Model for Integrated Pictorial Databases*, Comp. Vision, Graphics, and Image Proc., 25, pps 371–382.

[13] W. Grosky. (1994) *Multimedia Information Systems*, IEEE Multimedia, Vol. 1, 1, pps 12–24.

[14] A. Gupta, T. Weymouth and R. Jain. (1991) *Semantic Queries with Pictures: The VIMSYS Model*, Proc. 1991 Intl. Conf. on Very Large Databases, Barcelona, Spain, pps 69–79.

[15] J. Horst, E. Kent, H. Rifky and V.S. Subrahmanian. (1994) *Hybrid Knowledge Bases for Real-Time Robotic Reasoning*, Proc. IVth Intl. Workshop on Pattern Recognition in Practice (eds. E. Gelsema and L. Kanal), N. Holland/Elsevier.

[16] M. Iino, Y.F. Day and A. Ghafoor. (1994) *An Object-Oriented Model for Spatio-Temporal Synchronization of Multimedia Information*, Proc. 1994 Intl. Conf. on Multimedia Computing and Systems, pps 110-120, IEEE Press.

[17] J. Jaffar and J.L. Lassez. (1987) *Constraint Logic Programming*, Proc. ACM Principles of Programming Languages, pp 111–119.

[18] J. Jaffar, S. Michaylov, P. Stuckey, and R. Yap. (1992) *The CLP($\Re$) Language and System*, ACM Trans. on Programming Languages and Systems.

[19] M. Kifer and V.S. Subrahmanian. (1992) *Theory of Generalized Annotated Logic Programming and its Applications,* Journal of Logic Programming, 12, 4, pps 335–368.

[20] D. E. Knuth. (1968) *The Art of Computer Programming, Vol. I: Fundamental Algorithms*, Addison-Wesley.

[21] J. Lu, A. Nerode, and V.S. Subrahmanian. (1992) *Hybrid Knowledge Bases*, submitted to IEEE Trans. on Knowledge and Data Engineering, March 1993, revised Jan. 1994.

[22] S. Marcus and V.S. Subrahmanian. (1993) *Multimedia Database Systems*, submitted for publication.

[23] W. Niblack, et. al. (1993) *The QBIC Project: Querying Images by Content Using Color, Texture and Shape*, IBM Research Report, Feb. 1993.

[24] E. Oomoto and K. Tanaka. (1993) *OVID: Design and Implementation of a Video-Object Database System*, IEEE Trans. on Knowledge and Data Engineering, 5, 4, pps 629–643.

[25] J. Shoenfield. (1967) *Mathematical Logic*, Addison Wesley.

[26] A. P. Sistla, C. T. Yu, and R. Haddad. (1994) *Reasoning About Spatial Relationships in Picture Retrieval Systems*, to appear in Proc. 1994 Intl. Conf. on Very Large Databases, Santiago, Chile, Aug. 1994.

[27] V.S. Subrahmanian. (1992) *Amalgamating Knowledge Bases*, Univ. of Maryland Tech. Report CS-TR-2949, Aug. 1992. Accepted for publication in ACM Transactions on Database Systems.

[28] V.S. Subrahmanian. (1993) *Hybrid Knowledge Bases for Intelligent Reasoning Systems*, Invited Address, Proc. 8th Italian Conf. on Logic Programming (ed. D. Sacca), pps 3–17, Gizzeria, Italy, June 1993.

[29] R. Weiss, A. Duda and D.K. Gifford. (1994) *Content-Based Access to Algebraic Video*, Proc. 1994 Intl. Conf. on Multimedia Computing and Systems, pps 140–151, IEEE Press.

[30] G. Wiederhold. (1993) *Intelligent Integration of Information*, Proc. 1993 ACM SIGMOD Conf. on Management of Data, pps 434–437.

[31] G Wiederhold, S. Jajodia, and W. Litwin. Integrating temporal data in a heterogeneous environment. In *Temporal Databases*. Benjamin/Cummings, Jan 1993.

[32] D. Woelk and W. Kim. (1987) *Multimedia Information Management in an Object-Oriented Database System*, Proc. 13th Conf. on Very Large Databases, pps 319–329.

Appendix A: Expressing the Algebraic Operations of Weiss et. al. as Multimedia Specifications

Operation 1 *Concatenation: $E_1 \circ E_2$ defines the presentation where E_2 follows E_1.*

Suppose E_1 is generated by the multimedia specification $\mathcal{S}_1$ and E_2 is generated by the multimedia specification $\mathcal{S}_2$. Then $E_1 \circ E_2$ is generated by the multimedia specification $\mathcal{S}_1\mathcal{S}_2$.

Operation 2 *Union: $E_1 \cup E_2$ defines the presentation where E_2 follows E_1 and common footage is not repeated.*

Suppose E_1 is generated by the multimedia specification $\mathcal{S}_1 = Q_1^1, \ldots, Q_n^1$ and E_2 is generated by the multimedia specification $\mathcal{S}_2 = Q_1^2, \ldots, Q_m^2$. Let us define a new multimedia specification $\mathcal{S}_3 = Q_1^3, \ldots, Q_{n+m}^3$ defined as $Q_i^3 = Q_i^1$ if $i \leq n$ and Q_i^2 otherwise. $E_1 \cup E_2$ is generated by the following multimedia specification $\mathcal{S} = Q_1, Q_2, \ldots, Q_k$ where:

1. $Q_1 = Q_1^1$, and
2. $Q_{j+1} = Q_i^3$ where i is the smallest integer such that $Q_i^3 \notin \{Q_1, \ldots, Q_J\}$.

Operation 3 *Intersection: $E_1 \cap E_2$ defines the presentation where only common footage of E_1 and E_2 is played.*

Suppose E_1 is generated by the multimedia specification $\mathcal{S}_1 = Q_1^1, \ldots, Q_n^1$ and E_2 is generated by the multimedia specification $\mathcal{S}_2 = Q_1^2, \ldots, Q_m^2$. $E_1 \cap E_2$ is generated by the multimedia specification $\mathcal{S} = Q_1, \ldots, Q_k$ where:

1. $Q_1 = Q_j^1$ where j is the smallest integer such that Q_j^1 is in $\{Q_1^2, \ldots, Q_m^2\}$, and
2. $Q_{i+1} = Q_w^1$ where w is the smallest integer such that $Q_w^1 \notin \{Q_1, \ldots, Q_i\}$ and Q_w^1 is in $\{Q_1^2, \ldots, Q_m^2\}$.

Operation 4 *Difference: $E_1 - E_2$ defines the presentation where only footage of E_1 that is not in E_2 is played.*

Suppose E_1 is generated by the multimedia specification $\mathcal{S}_1 = Q_1^1, \ldots, Q_n^1$ and E_2 is generated by the multimedia specification $\mathcal{S}_2 = Q_1^2, \ldots, Q_m^2$. $E_1 - E_2$ is generated by the multimedia specification $\mathcal{S} = Q_1, \ldots, Q_k$ where:

1. $Q_1 = Q_j^1$ where j is the smallest integer such that Q_j^1 is *not* in $\{Q_1^2, \ldots, Q_m^2\}$, and
2. $Q_{i+1} = Q_w^1$ where w is the smallest integer such that $Q_w^1 \notin \{Q_1, \ldots, Q_i\}$ and Q_w^1 is not in $\{Q_1^2, \ldots, Q_m^2\}$.

Operation 5 *Parallel: $E_1 \| E_2$ defines the presentation where E_1 and E_2 are played concurrently and start simultaneously.*

Suppose E_1 is generated by the multimedia specification $\mathcal{S}_1 = Q_1^1, \ldots, Q_n^1$ and E_2 is generated by the multimedia specification $\mathcal{S}_2 = Q_1^2, \ldots, Q_m^2$. Rename all existential variables in $\mathcal{S}_2$ so that for all $1 \leq i \leq min(n, m)$, Q_i^1 and Q_i^2 share no common variables. Then $E_1 \| E_2$ is generated by the multimedia specification $\mathcal{S} = Q_1, \ldots, Q_k$ where:

> $Q_i = Q_i^1 \,\&\, Q_i^2 \,\&\, \mathtt{start(S_1)} = \mathtt{start(S_2)} = \ldots = \mathtt{start(S_w)}$ where $\mathtt{S}_1^*, \ldots, \mathtt{S_w}$ are all the state-variables in either Q_i^1 or Q_i^2.

Operation 6 *Parallel-End: $E_1 \wr E_2$ defines the presentation where E_1 and E_2 are played concurrently and terminate simultaneously.*

This is identical to the "Parallel" composition operation except that the `start` variables are replaced by `end` variables.

Operation 7 *Conditional:* $(\mathtt{test})?E_1 : E_2 : \ldots E_k$ *defines the presentation where E_i is played if* `test` *evaluates to i.*

Suppose E_i is generated by the multimedia specification $\mathcal{S}_i = Q_1^i, \ldots, Q_{n_i}^i$. Then $(\mathtt{test})?E_1 : E_2 : \ldots E_k$ is generated by the multimedia specification $\mathcal{S} = Q_1, \ldots, Q_k$ where: $Q_i =$

$$\bigwedge_{j=1}^{k} \left((test = i)) \rightarrow Q_i^j \right).$$

Operation 8 *Loop:* $\mathtt{loop}E_1$ *time defines a repetition of video-expression E_1 for a duration of time.*

Suppose E_1 is generated by the multimedia specification $\mathcal{S}_1$. Then $[E_1] : time$ is the constrained multimedia specification that generates the above desired presentation.

Operation 9 *Stretch:* $\mathtt{stretch}E_1$ *factor. The duration of the presentation is equal to factor times duration of E_1.*

Suppose E_1 is generated by the multimedia specification $\mathcal{S}_1 = Q_1^1, Q_2^1, \ldots, Q_k^1$. Then $\mathtt{stretch}E_1 : factor$ may be generated by the media specification $\mathcal{S} = Q_1, Q_2, \ldots$ where $Q_i =$

$$Q_i^1 \,\&\, \mathtt{duration(S,D)} \,\&\, \mathtt{start(S,T)} \,\&\, \mathtt{end(S,T + D \times factor)}.$$

Here, we assume that Q_i^1 contains only one state-variable; if there are more, this will lead only to a longer conjunction.

Operation 10 *Limit:* $\mathtt{limit}E_1$ *time. The duration of the presentation is equal to the minimum of time and the duration of E_1.*

Suppose E_1 is generated by the multimedia specification $\mathcal{S}_1 = Q_1^1, Q_2^1, \ldots, Q_k^1$. Then $\texttt{limit} E_1 time$ may be generated by the media specification $\mathcal{S} = Q_1, Q_2, \ldots$ where $Q_i =$

$$Q_i^1 \,\&\, \texttt{duration(S, D)} \,\&\, \texttt{start(S, T)} \,\&\, \texttt{end(S, T + min(D, Time))}.$$

Operation 11 *Transition:* $\texttt{transition} E_1 E_2\, type\, time$ *Defines type transition between expressions E_1 and E_2; time defines the duration of the transition effect.*

Suppose E_1 is generated by the multimedia specification $\mathcal{S}_1 = Q_1^1, \ldots, Q_n^1$ and E_2 is generated by the multimedia specification $\mathcal{S}_2 = Q_1^2, \ldots, Q_m^2$. Suppose Q is a query that generates a multimedia presentation of the form specified in *type* – this could include things like "show a blank video" or "halt" etc. Then the $\texttt{transition} E_1 E_2$ *type time* operation can be expressed as the multimedia specification E_1, X, E_2 where X is given by:

$$Q \,\&\, \texttt{duration(Q, time)}.$$

Operation 12 *Contains:* $\texttt{contains} E_1$ *query defines the presentation that contains component expressions of E_1 that match query.*

Suppose E_1 is generated by the multimedia specification $\mathcal{S}_1 = Q_1^1, \ldots, Q_n^1$. Now rename the variables in *query* so that no variables are shared with any of the Q_j^i's. Then the operation $\texttt{contains} E_1$ *query* is captured by the multimedia specification $\mathcal{S} = Q_1, Q_2, \ldots, Q_n$ where

$$Q_j = (Q_j^1 \,\&\, query).$$