Multimedia Database Systems *

Sherry Marcus
Mathematical Sciences Institute
Cornell University
Ithaca, NY 14853.
E-mail: marcus@msicedar.cit.cornell.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 there are now numerous examples of multimedia systems in the commercial market, these systems have been developed primarily on a case-by-case basis. The large-scale development of such systems requires a principled characterization of multimedia systems which is independent of any single application. It requires a unified query language framework to access these different structures in a variety of ways. It requires algorithms that are provably correct in processing such queries and whose efficiency can be appropriately evaluated. In this paper, we develop a framework for characterizing multimedia information systems which builds on top of the implementations of individual media, and provides a logical query language that integrates such diverse media. We develop indexing structures and algorithms to process such queries and show that these algorithms are sound and complete and relatively efficient (polynomial-time). We show that the generation of media-events (i.e. generating different states of the different media concurrently) can be viewed as a query processing problem, and that synchronization can be viewed as constraint solving. This observation allows us to introduce the notion of a media presentation as a sequence of media-events that satisfy a sequence of queries. We believe this paper represents a first step towards the development of multimedia theory.

1 Introduction

Though numerous multimedia systems exist in today's booming software market, relatively little work has been done in addressing the following questions:

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

- What are multimedia database systems and how can they be formally/mathematically defined so that they are independent of any specific application domain ?
- Can indexing structures for multimedia database systems be defined in a similar uniform, domain-independent manner ?
- Is it possible to uniformly define both query languages and access methods based on these indexing structures ?
- Is it possible to uniformly define the notion of an update in multimedia database systems and to efficiently accomplish such updates using the above-mentioned indexing structures ?
- What constitutes a multimedia presentation and can this be formally/mathematically defined so that it is independent of any specific application domain ?

In this paper, we develop a set of initial solutions to all the above questions. We provide a formal theoretical framework within which the above questions can be expressed and answered.

The basic concepts characterizing a multimedia system are the following: first, we define the important concept of a **media-instance**. Intuitively, a media-instance (e.g. an instance of video) consists of a body of information (e.g. a set of video-clips) represented using some storage mechanism (e.g. a quadtree, or an R-tree or a bitmap) in some storage medium (e.g. video-tape), together with some functions and/or relations (e.g. next minute of video, or who appears in the video) expressing various aspects, features and/or properties of this media-instance. We show that media-instances can be used to represent a wide variety of data including documents, photographs, geographic information systems, bitmaps, object-oriented databases, and logic programs, to name a few.

Based on the notion of a media-instance, we define a **multimedia system** to be a set of such media-instances. Intuitively, the concatenation of the states of the different media instances in the multimedia system is a snapshot of the global state of the system at a given point in time. Thus, for instance, a multimedia system (at time t) may consist of a snapshot of a particular video-tape, a snapshot of a particular audio-tape, and segments of affiliated (electronic) documentation. In Section 4, we develop a **logical query language** that can be used to express queries requiring multimedia accesses. We show how various "intuitive" queries can be expressed within this language. Subsequently, we define an **indexing structure** to store multimedia systems. The elegant feature of our indexing structure is that it is completely independent of the type of medium being used – in particular, if we are given a *pre-existing* representation/implementation of some information in some medium, our method shows how various interesting aspects (called "features") of this information can be represented, and efficiently accessed. We show how queries expressed in our logical query language can be efficiently executed using this indexing structure.

Section 5 introduces the important notion of a media presentation based on the notion of a **media-event**. Intuitively, a media-event reflects the global state of the different media at a fixed point in time. For example, if, at time t, we have a picture of George Bush on the screen (i.e. video medium) and an audio-tape of George Bush saying X, then this is a media-event with the video-state being "George Bush" and the audio-state being "George Bush saying X." A **media presentation** is a sequence of media-events. Intuitively, a media-presentation shows how the states of different media-instances change over time. One of the key results in this paper is that any query generates a set of media-events (i.e. those media-events that satisfy the query). Consequently, the problem of specifying a media-presentation can be achieved by specifying a sequence of queries. In other words,

$$\textbf{Generation of Media Events} \quad = \quad \textbf{Query Processing}.$$

Finally each media-event (i.e. a global state of the system) must be "on" for a certain period of time (e.g. the audio clip of Bush giving a speech must be "on" when the video shows him speaking). Furthermore, the next media-event must come on immediately upon the completion of the current media-event. We show that this process of synchronizing media-events to achieve a deadline may be viewed as a constraint solving problem, i.e.

$$\textbf{Synchronization} \quad = \quad \textbf{Constraint Solving}.$$

2 Basic Ideas Underlying the Framework

In this section, we will articulate the basic ideas behind our proposed multimedia information system architecture. For now, we will view a media-source as some, as yet unspecified, representation of information. Exactly how this information is stored physically, or represented conceptually, is completely independent of our framework, thus allowing our framework to be interface with most existing media that we know of.

Suppose $\mathcal{M}$ is a medium and this medium has several "states" representing different bodies of knowledge expressed in that medium – associated with this data is a set of "features" – these capture the salient aspects and objects of importance in that data. In addition, there is logically specified information describing relationships and/or properties between features occurring in a given state. These relationships between features are encoded as a logic program. Last, but not least, when a given medium can assume a multiplicity of states, we assume that there is a corpus of state-transition functions that allow us to smoothly move from one state to another. These are encoded as "inter-state" relationships, specifying relations existing between states taken as a whole. As the implementation of these inter-state transition functions is dependent on the medium, we will assume that there is an existing implementation of these transition functions. As we make no assumptions on this implementation, this poses no restrictions. Figure 1 shows the overall architecture for multimedia information systems.

The ideas discussed thus far are studied in detail in Section 4 where we develop a query language to integrate information across these multiple media sources and express queries, and where we develop access structures to efficiently execute these queries.

All the aspects described thus far are independent of time and are relatively static. In real-life multimedia systems, time plays a critical role. For instance, a query pertaining to audio-information may need to be synchronized with a query pertaining to video-information, so that the *presentation* of the answers to these queries have a coherent audio-visual impact.

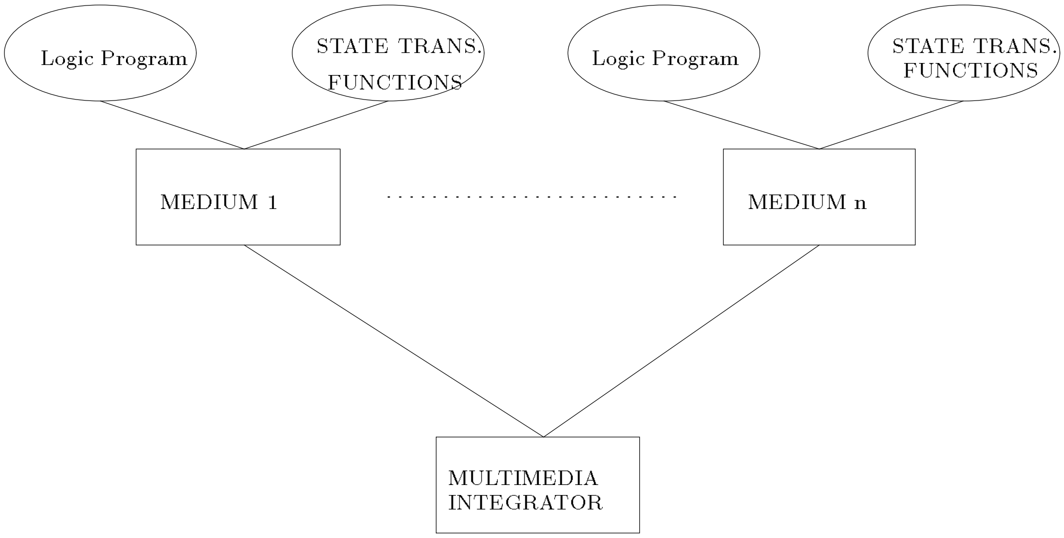


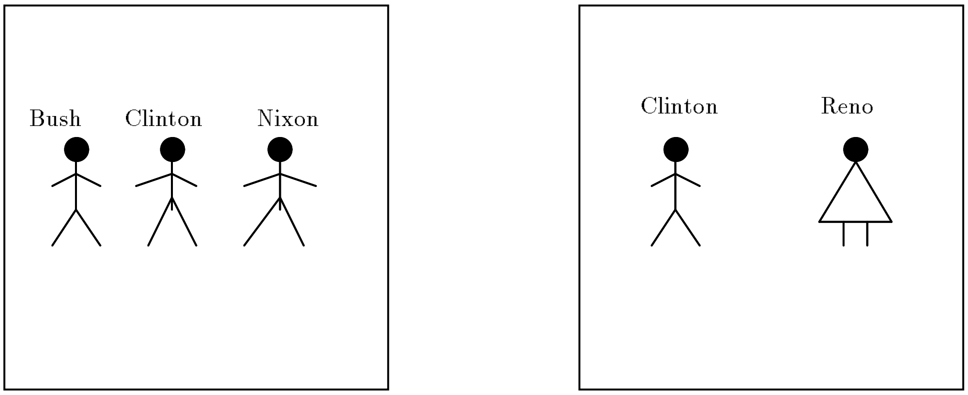
Figure 1: Multimedia Information System Architecture

Hence, the data structures used to represent information in the individual media (which so far, has been left completely unspecified) must satisfy certain efficiency requirements. We will show that by and large, these requirements can be clearly and concisely expressed as constraints over a given domain, and that based on the design criteria, index structures to organize information within a medium can be efficiently designed.

3 Media Instances

In this section, we formally define the notion of a media-instance, and show how it can be used to represent a wide variety of data stored on different kinds of media. Intuitively, a medium (such as `video`) may have data stored on it in many formats (e.g. `raster`, `bitmap`, `vhs_format`, `pal`, `secam`, etc.). Thus, raster is an example of an instance of the medium `video` because video information may be stored in raster format. However, in addition to just storing information, media instances, as defined below contain information on how to access and manipulate that information.

Definition 1 A *media-instance* is a 7-tuple $\mathbf{mi} = (\mathbf{ST}, \mathbf{fe}, \lambda, \Re, \mathcal{F}, \mathbf{Var}_1, \mathbf{Var}_2)$ where $\mathbf{ST}$ is a set of objects called *states*, $\mathbf{fe}$ is a set of objects called *features*, λ is a map from S to $2^{\mathbf{fe}}$, $\mathbf{Var}_1$ is a set of objects called *state variables* ranging over states, $\mathbf{Var}_2$ is a set of objects called *feature variables* ranging over features, $\Re$ is a set of *inter-state relations*, i.e. relations (of possibly different arities) on the set $\mathbf{ST}$, and $\mathcal{F}$ is a set of *feature-state relations*. Each relation in $\mathcal{F}$ is a subset of $\mathbf{fe}^i \times \mathbf{ST}$ where $i \geq 1$.



(a) (b)

Figure 2: Two Picture Frames

3.1 The Clinton Example

We will try to explain the intuitions underlying the definition of a media-instance by considering three media (video, audio and document) representing various political figures. This example will be a "running example" throughout the paper.

Example 1 (A Video-Domain) Consider bitmapped photographs of various high-ranking US government officials shown in Figure 2.

Intuitively, a media instance $\mathbf{mi} = (\mathbf{ST}, \mathbf{fe}, \Re, \mathcal{F}, \mathbf{Var}_1, \mathbf{Var}_2)$ depicting the above two photographs contains:

1. a state $s \in \mathbf{ST}$ captures a certain structure used to store information. For example, in Figure 2, the set $\mathbf{ST}$ is the set of all possible bitmaps of the appropriate dimensions. The two photographs shown in Figure 2 represent two specific states (i.e. bitmaps) in $\mathbf{ST}$. By just looking at a state, it is impossible to say anything about the objects of interest in that state.

2. A *feature* is a piece of information that is thought to be an item of significance/interest about a given state. For instance, the features of interest in our bitmapped domain may include `clinton, gore, bush, nixon, reno, reagan, kissinger`. (The fact that only some of these features appear in the two pictures shown in Figure 2 is irrelevant; the missing features may occur in other pictures not depicted above).

3. λ is a map that tells us which features are possessed by a given state. Thus, for instance, suppose s_1 and s_2 denote the two states depicted in Figure 2. Then

$$\begin{aligned}\lambda(s_1) &= \{\texttt{bush}, \texttt{clinton}, \texttt{nixon}\}.\\ \lambda(s_2) &= \{\texttt{clinton}, \texttt{reno}\}.\end{aligned}$$

The first equation above indicates that the features possessed by state s_1 are `clinton, nixon,` and `bush`.

4. Relations in $\Re$ represent connections between states. For instance, the relation $\mathtt{delete\_nixon(S,S')}$ could hold of any pair of states $(\mathtt{S},\mathtt{S'})$ where S contains `nixon` as feature, and $\mathtt{S'}$ has the same features as $\mathtt{S}$, with the feature `nixon` deleted. As implementation of inter-state relations is fundamentally dependent upon the particular medium in question, we will develop our theory to be independent of any particular implementation (though we will be assuming one exists).

5. Relations in $\mathcal{F}$ represent relationships between features in a given state. Thus, for instance, in the photograph of Clinton and Reno shown in Figure 2(b), there may be a relation $\mathtt{left(clinton,Reno,s_2)}$ specifying that Clinton is standing to the left of Reno in the state $\mathtt{s_2}$.

$\diamondsuit\diamondsuit$

Definition 2 A *state-term* of a media-instance $\mathbf{mi} = (\mathbf{ST}, \mathbf{fe}, \Re, \mathcal{F}, \mathbf{Var}_1, \mathbf{Var}_2)$ is any element of $(\mathbf{ST} \cup \mathbf{Var}_1)$. A *feature-term* of media-instance $\mathbf{mi} = (\mathbf{ST}, \mathbf{fe}, \Re, \mathcal{F}, \mathbf{Var}_1, \mathbf{Var}_2)$ is any element of $(\mathbf{fe} \cup \mathbf{Var}_2)$.

Definition 3 If $R \in \Re$ is an n-ary relation in media-instance $\mathbf{mi} = (\mathbf{ST}, \mathbf{fe}, \lambda, \Re, \mathcal{F}, \mathbf{Var}_1, \mathbf{Var}_2)$ and $t_1, \ldots, t_n$ are terms, then $R^*(t_1, \ldots, t_n)$ is a *state-constraint* in media instance $\mathbf{mi}$. This constraint is solvable iff there exists a way of replacing all variables occurring in $t_1, \ldots, t_n$ by states in $\mathbf{ST}$ so that the resulting n-tuple is in relation R.

Here, R^* is a symbol (syntactic entity) denoting the relation R (semantic entity).

Definition 4 If $\phi \in \mathcal{F}$ is an n-ary relation in media-instance $\mathbf{mi} = (\mathbf{ST}, \mathbf{fe}, \lambda, \Re, \mathcal{F}, \mathbf{Var})$ and $c_1, \ldots, c_{n-1}$ are features terms and s is a state-term, then $\phi^*(c_1, \ldots, c_{n-1}, s)$ is a *feature-constraint*. This constraint is solvable iff there exists a way of replacing all variables in $c_1, \ldots, c_{n-1}$ by features in $\mathbf{fe}$ and replacing s (if it is a state variable) by a state in $\mathbf{ST}$ so that the resulting n-tuple is in relation ϕ.

Here, $\phi*$ is a symbol (syntactic entity) denoting the relation ϕ (semantic entity).

The concept of a media-instance as defined above is extremely general and covers a wide range of possibilities. Below, we give a number of examples of media-instances, specifying different areas of applicability of this framework.

Example 2 Let us return to the Clinton-scenario depicted by the two pictures shown in Figure 2. It may turn out that some relevant audio-information is also available about that particular cast of characters, i.e. `clinton, gore, bush, nixon, reno, reagan, kissinger`, as well as some other entities e.g. `who, unesco, world_bank`. This, then, may be the set of features of a (restricted) audio media-instance. For instance, we may have a set of audio-tapes a_1, a_2, a_3 where a_1 depicts Clinton speaking about the WHO (World Health Organization), a_2 may be an audio-tape with Clinton and Gore having a discussion about unesco, while a_3 may be an audio-tape in which Bush and Clinton are engaged in a debate (about topics too numerous to mention). The feature assignment function, then is defined to be:

$$\lambda(a_1) = \{\mathtt{clinton}, \mathtt{who}\}.$$
$$\lambda(a_2) = \{\mathtt{clinton}, \mathtt{gore}, \mathtt{unesco}\}.$$
$$\lambda(a_3) = \{\mathtt{clinton}, \mathtt{bush}\}.$$

There may be an inter-state relation called `after` defined to be the transitive closure of $\{(a_1, a_2), (a_2, a_3)\}$ saying that a_2 occurs after a_1 and a_3 occurs after a_2. Feature-state relations specify connections between features and states. For instance, the relation `topic` may contain the tuples $(\mathtt{who}, \mathtt{a}_1)$, $(\mathtt{unesco}, \mathtt{a}_2)$ specifying the topics of a_1 and a_2, respectively. Likewise, the relation `speaker(i,person,frame)` may specify that the `i`'th speaker in a particular frame is `person` so and so. Thus, with respect to the audio-frame a_2, we may have the tuples:

$\mathtt{speaker}(1, \mathtt{clinton}, \mathtt{a}_2)$
$\mathtt{speaker}(2, \mathtt{gore}, \mathtt{a}_2)$
$\mathtt{speaker}(3, \mathtt{clinton}, \mathtt{a}_2)$
$\mathtt{speaker}(4, \mathtt{gore}, \mathtt{a}_2)$

specifying that Clinton speaks first in a_2, followed by Gore, followed again by Clinton, and finally concluded by Gore. $\diamondsuit\diamondsuit$

A more detailed scenario of how audio-information can be viewed as a media-instance is described later in Example 9. The following example revisits the Clinton-scenario with respect to document information.

Example 3 Suppose we have three documents, d_1, d_2 and d_3 reflecting information about policies adopted by various organizations. Let us suppose the set of features is identical to the set given in the previous example. Suppose document d_1 is a position statement of the World Health Organization about Clinton; document d_2 is a statement made by Clinton about the WHO and document d_3 is a statement about UNESCO made by Clinton. The feature association map, λ is defined as follows:

$$\lambda(d_1) = \{\mathtt{who}, \mathtt{clinton}\}.$$
$$\lambda(d_2) = \{\mathtt{who}, \mathtt{clinton}\}.$$
$$\lambda(d_3) = \{\mathtt{unesco}, \mathtt{clinton}\}.$$

Note that even though d_1 and d_2 have the same features, this doesn't mean that they convey the same information – after all, a WHO statement about Clinton is very different from a statement made by Clinton about the WHO. Hence, let us suppose we have a feature-state relation in $\mathcal{F}$ called $\mathtt{contents}(\langle\mathtt{author}, \mathtt{topic}, \mathtt{state}\rangle)$, and that this relation contains the following triples:

$\mathtt{contents}(\mathtt{who}, \mathtt{clinton}, \mathtt{d}_1)$
$\mathtt{contents}(\mathtt{clinton}, \mathtt{who}, \mathtt{d}_2)$
$\mathtt{contents}(\mathtt{clinton}, \mathtt{unesco}, \mathtt{d}_3)$.

A

red	green
red	green

B

blue	green
red	red

C

green	red
green	red

Figure 3: Example for the Matrix Media-Instance

The set $\Re$ of inter-state relations is left empty for now. $\diamondsuit\diamondsuit$

A more detailed scenario of how documents can be viewed as a media-instance is described later in Example 10. Above, we have described a scenario containing information pertaining to certain objects (e.g. `clinton, gore,`, etc.) and shown how this information can be represented using `video, audio` and `document` media-instances. We will refer to these three particular scenarios as the "Clinton-example" in the rest of this paper.

3.2 Examples of Media-Instances

The following examples show how the notion of a media-instance is very general and can be used to describe a wide variety of media types (and data representations on that medium) that are likely to be encountered in practice.

Example 4 **(2×2 Matrices)** Consider the set of 2×2 matrices whose values can be in the set { red, blue, green }. This forms the set, **ST**, of states of a media-instance **LM**. We can define several inter-state relations on this media-instance. For instance, we may define:

1. M_1 similar M_2 iff matrices M_1 and M_2 have the same color in at least 2 pixel entries. In figure 1, matrices A and B are similar, but A and C are not.
2. M_1 have the same colors M_2 iff the set of colors in M_1 and the set of colors in M_2 are the same. In figure 1, A and C have the same colors, but A and B do not, and B and C do not either.

Note that A, B, C shown in Figure 3 are state-terms in the matrix media-instance. In this example, we assume that the feature set is empty, and hence, the function λ is empty and $\mathcal{F}$ is empty. $\diamondsuit\diamondsuit$

Example 5 **(Quad-Tree Media-Instance)** Consider any elementary record structure called `INFO`, and suppose we consider the space of all quad-trees [17] that can be constructed using this record structure as the information field(s) in a node. In addition, there are four fields, NW, SW, NE, SE denoting the four quadrants of a rectangular region. Then we can define a media-instance called $\mathsf{QT} = (\mathbf{ST}, \mathbf{fe}, \lambda, \Re, \mathcal{F}, \mathbf{Var}_1, \mathbf{Var}_2)$ where $\mathbf{ST}$ is the set of all such quadtrees (this set may be infinite). The variables in $\mathbf{Var}_1$ can be instantiated to specific quadtrees. $\Re$ may contain a bunch of relations of (possibly) different arities. Some examples of such relations are:

- $\mathsf{nw\_empty}$ is a unary relation such that $\mathsf{nw\_empty}(V)$ is true of quad-tree V iff the NW-link of each node in quadtree V is empty.
- $V_1\,\mathsf{same\_num}\,V_2$ iff quad-trees V_1 and V_2 have the same number of nodes (even though both quadtrees may be very different).
- $V_1\,\mathsf{same}\,V_2$ iff V_1 and V_2 are identical.
- $\mathsf{between}(V_1, V_2, V_3)$ iff V_1 is a subtree of V_2 and V_2 is a subtree of V_3.

Suppose the quadtrees in question describe the geographical layout of Italy. Then some of the features of interest may be: `Rome, Venice, Genoa, Milan`. There may an inter-feature relationship called `larger_than` such that:

```
larger_than(milan,genoa,S)
larger_than(rome,venice,S)
        ...etc.
```

Above, `S` is a state-variable and the above constraints reflect the fact that Milan is larger than Genoa in all states. However, there may state-specific feature constraints: for instance, in a specific quad-tree instance showing a detailed map of Rome, we may have a constraint saying:

$$\mathtt{in}(\mathtt{rome}, \mathtt{colosseum}, \mathtt{s}_1).$$

However, in a full map of Italy, the constraint

$$\mathtt{in}(\mathtt{rome}, \mathtt{colosseum}, \mathtt{fullmap})$$

may not be present because the Colosseum may be a feature too small/unimportant to be represented in a full map of Italy. The feature assignment function would specify precisely in which states which features are relevant. $\diamondsuit\diamondsuit$

Example 6 (Relational Database Media-Instance) Consider any relational database having relational schemas

$$R_1(A_1^1, \ldots, A_{n_1}^1), \ldots, R_k(A_1^k, \ldots, A_{n_k}^k).$$

The media-instance, RDB of relational databases can be expressed as a 7-tuple $(\mathbf{ST}, \mathbf{fe}, \lambda, \Re, \mathcal{F}, \mathbf{Var}_1, \mathbf{Var}_2)$ as follows. Let $\mathbf{ST}$ be the set $\bigcup_{i=1}^{k} \bigcup_{j=1}^{n_k} dom(A_j^i)$. Let $\Re = \{R_1^\flat, \ldots, R_k^\flat\}$ where $R_i^\flat$ is the set of tuples in relation R_i. The variables range over the elements in $\mathbf{ST}$. All other parts of the 7-tuple are empty. $\diamondsuit\diamondsuit$

Example 7 (Deductive Database Media-Instance) Suppose we consider definite Horn clause logic programs [14] over a given alphabet. Then we can define a media-instance DDL as follows: **ST**, the set of states, is the set of ground terms generated by this alphabet. **fe** = $\emptyset$ and so is λ. $\mathbf{Var}_1$ is simply the set of variable symbols provided in the alphabet (and, as usual, these variables range over the ground terms in the language). For each n-ary predicate symbol p in the alphabet, there is an n-ary relation, R^p in $\Re$; $\Re$ contains no other relations. (A logician might recognize that DDL is, intuitively, just an Herbrand model [14]). All other components of the media instance are empty. $\diamondsuit\diamondsuit$

Example 8 (Object-Oriented Media-Instances) Suppose we consider an inheritance hierarchy containing individuals $i_1, \ldots, i_n$, classes $c_1, \ldots, c_r$, methods $m_1, \ldots, m_s$, and properties $p_1, \ldots, p_k$. Let H be the hierarchy relationship, i.e. $H(x, y)$ means that individual/class x is a member/subclass of class y. Then we can define a media-instance, OOL as follows: the set of states, **ST**, is $\{i_1, \ldots, i_n, c_1, \ldots, c_r, m_1, \ldots, m_s\}$. Variables range over individuals, classes and methods. Each property p_j is a unary relation in $\Re$.

Some additional examples of relations that could be in $\Re$ are:

- subclass(V_1, V_2) iff V_1 is a subclass (resp. individual) of (resp. in) class V_2.
- same_num(V_1, V_2) iff V_1 and V_2 are both classes containing the same number of individuals.
- An important relation is the applicability of methods to classes. This could be encoded as a special relation, applicable(m_j, c_w) saying that method m_j is applicable to class c_w. All other components of the 7-tuple are empty.

$\diamondsuit\diamondsuit$

Example 9 (Audio Media-Instances) Suppose we consider audio input. It is well-known that voice/audio signals are sets of sine/cosine functions[1]. Let VL be the language defined as follows. The set, **ST**, is the set of all sine/cosine waves. Features may include the properties of the signals such as frequency and amplitude which in turn determine who/what is the originator of the signals (e.g. Bill Clinton giving a speech, Socks the cat meowing, etc.). State variables range over sets of audio signals. Examples of relations in $\Re$ are:

- same_amplitude(V_1, V_2) iff V_1 and V_2 have the same amplitude.
- Similarly, binary relations like higher_frequency and more_resonant may be defined.

Relations in $\mathcal{F}$ may include feature-based relations such as `owns(clinton,socks,S)` specifying that Socks is owned by Clinton in all states in our system. $\diamondsuit\diamondsuit$

[1] Technically, it would be more correct to say that it is possible to approximate any audio signal with sine and cosine waveforms (using Fourier series) as long as the signal is periodic. The reason is that you need the fundamental frequency (or time period) to decompose the signal into a series.

Example 10 (Document Media-Instances) Suppose we consider an electronic document storage and retrieval scheme. Typically, documents are described in some language such as SGML. Let DOCL be the media-instance defined as follows. **ST** is the set of all document descriptions expressible in syntactically valid form (e.g. in syntactically correct SGML and/or in Latex or in some other form of hypertext). State variables range over these descriptions of documents. Examples of relations in $\Re$ are:

- university_tech_rep(V) is true iff the document represented by V is a technical report of some university.
- cut_paste(V_1, V_2, V_3, V_4) iff V_4 represents the document obtained by cutting V_1 from document V_3 and replacing it by V_2.
- comb_health_benefits_chapter$(V_1, \ldots, V_{50}, V)$ iff V represents the document obtained by concatenating together, the chapter on health benefits from documents represented by $V_1, \ldots, V_{50}$. For example, $V_1, \ldots V_{50}$ may be handbooks specifying the legal benefits that employees of companies are entitled in the 50 states of the U.S.A. V, in this case, would be a document describing the health benefits laws in the different states.

Features of a document may include entities such as:

`dental`, `hospitalization`, `emergency_care`.

Feature constraints (i.e. members of $\mathcal{F}$) may include statements about maximal amounts of coverage, e.g. statements such as:

```
max_cov(dental,5000,d_1),
max_cov(hospitalization,1000000,d_1),
max_cov(emergency,100000,d_1).
```

Here, `d_1` is a specific document describing, say, the benefits offered by one health care company. Conversely, `d_2` may be a document reflecting similar coverage offered by another company, except that the maximal coverage amounts may vary from those provided by the first company. $\diamondsuit\diamondsuit$

Definition 5 A *multimedia system* MMS is a finite set of media instances.

4 Indexing Structures and a Query Language for Multimedia Systems

Consider a multimedia system $\mathcal{M} = \{\mathcal{M}_1, \ldots, \mathcal{M}_n\}$ that a user wishes to retrieve information from. In this section, we will develop a query language and indexing structures for accessing such multimedia systems.

4.1 Frame-Based Query Language

In this section, we develop a query language to express queries addressed to a multimedia system $\mathbf{MMS} = \{\mathcal{M}_1, \ldots, \mathcal{M}_n\}$ where

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

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.
   (b) Each $s \in \mathbf{ST}^i$ for $1 \leq i \leq n$ is a constant symbol in the query language.
   (c) Each integer $1 \leq i \leq n$ is a constant symbol.
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 $\psi \in \Re_2^i$ 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 11 Let us return to the video-domain in the Clinton-example (Figure 2). Let us suppose that we have the following feature-state relations.

1. `running_mate(X,Y,S)`: `X`'s running mate is `Y`.
2. `appointed(X,Y,P,S)`: `X` appointed `Y` to position `P` in state `S`.
3. `with(X,Y,S)`: `X` is with `Y` in state `S`.

Observe that in the first two relations listed above, the state (i.e. the video-frame) does not actually matter – Clinton's running mate is Gore, independent of which picture is being looked at. Clinton appointed Reno as Attorney General, and this is independent of the picture being looked at. The third relation above is picture-specific, though. In picture frame 1 Clinton is with Bush and with Gore – this contributes the facts:

```
with(clinton, bush, 1).
with(clinton, nixon, 1).
```

while the fact

```
with(clinton, reno, 2).
```

is contributed by the second picture. In addition, we will allow background inference rules to be present; these allow us to make statements of the form:

$$\mathtt{with(Y, X, S)} \quad \leftarrow \quad \mathtt{with(X, Y, S)}$$

specifying that if $\mathtt{X}$ is $\mathtt{with}$ $\mathtt{Y}$ in state $\mathtt{S}$, then $\mathtt{Y}$ is $\mathtt{with}$ $\mathtt{X}$ in that state.

A user of the multimedia system consisting of the picture frames may now ask queries such as:

1. $(\exists \mathtt{X}, \mathtt{P}, \mathtt{S})\mathtt{appointed(clinton, X, P, S)} \,\&\, \mathtt{with(clinton, X, S)} \,\&\, \mathtt{frametype(video)}$): This query asks whether there is anyone who is a Clinton-appointee who appears in a picture/video frame with Clinton. The answer is "yes" with $\mathtt{X} = \mathtt{reno}$, $\mathtt{P} = \mathtt{Attorney\ General}$ and $\mathtt{S} = 2$. (We are assuming here that atoms defining the predicate $\mathtt{appointed}$ are stored appropriately.)

2. $(\exists \mathtt{X}, \mathtt{Y}, \mathtt{S}, \mathtt{S}_1, \mathtt{S}_2)\mathtt{president}(\mathtt{X}, \mathtt{S}_1), \,\&\, \mathtt{president}(\mathtt{Y}, \mathtt{S}_2) \,\&\, \mathtt{X} \neq \mathtt{clinton} \,\&\, \mathtt{Y} \neq \mathtt{clinton} \,\&\, \mathtt{X} \neq \mathtt{Y} \,\&\, \mathtt{with(clinton, X, S)} \,\&\, \mathtt{with(clinton, Y, S)} \,\&\, \mathtt{frametype(S, video)}$): This query asks if there is any picture in which which three Presidents of the USA (one of whom is Clinton) appear together.

3. $(\exists \mathtt{S})(\mathtt{clinton} \in \mathtt{flist(S)} \,\&\, \mathtt{horse} \in \mathtt{flist(S)} \,\&\, \mathtt{on(clinton, horse)} \,\&\, \mathtt{frametype(S, video)})$: This question asks if there is a picture of Clinton on a horse.

4. $(\exists \mathtt{S})(\mathtt{clinton} \in \mathtt{flist(S)} \,\&\, \mathtt{socks} \in \mathtt{flist(S)} \,\&\, \mathtt{meowing\_at(socks, clinton)} \,\&\, \mathtt{frametype(S, audio)})$: Is there an audio-frame in which both Clinton and Socks are "featured" and Socks, the cat, is meowing at Clinton ?

5. $(\exists \mathtt{S}_1, \mathtt{S}_2)\mathtt{nixon} \in \mathtt{flist}(\mathtt{S}_1) \,\&\, \mathtt{frametype}(\mathtt{S}_1, \mathtt{video}) \,\&\, \mathtt{X} \in \mathtt{flist}(\mathtt{S}_1) \,\&\, \mathtt{X} \neq \mathtt{nixon} \,\&\, \mathtt{person(X)} \,\&\, \mathtt{X} \in \mathtt{flist}(\mathtt{S}_2) \,\&\, \mathtt{frametype}(\mathtt{S}_2, \mathtt{audio})$): This query looks to find a person pictured in a video-frame with Nixon, who is speaking in an audio-frame elsewhere.

◊◊

In general, if we are given a media-instance

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

then we will store information about the feature-state relations as a logic program. There are two kinds of facts that are stored in such a logic program.

State-Independent Facts: These are facts that reflect relationships between features that hold in all states of media-instance $\mathcal{M}_i$. Thus, for example, in the Clinton example, the fact that Gore is Clinton's vice-president is true in all states of the medium $\mathcal{M}_i$. This is represented as:

$$\texttt{vice\_pres}(\texttt{clinton}, \texttt{gore}, \texttt{S}) \leftarrow$$

where `S` is a state-variable.

State-Dependent Facts: These are facts that are true in some states, but false in others. In particular, if $\phi \in \mathbf{fe}$ is a j-ary relation ($j \geq 1$), and tuple $\vec{\mathtt{t}}, \mathtt{s} \in \phi$, then the unit clause (or fact)

$$\phi^*(\vec{\mathtt{t}}, \mathtt{s}) \leftarrow$$

is present in the logic program. Thus, for instance, in a particular picture (e.g. figure 2), Clinton is to the left of Reno, and hence, this can be expressed as the state-dependent fact

$$\texttt{left}(\texttt{clinton}, \texttt{reno}, \texttt{s}_2)$$

where $\mathtt{s}_2$ is the name of the state in Figure 2(b).

Derivation Rules: Last, but not least, the designer of the multimedia system may add extra rules that allow new facts to be derived from facts in the logic program. For instance, if we consider the predicate `left(person1,person2,S)` denoting that `person1` is to the left of `person2` in state `S`, then a designer of the media-instance in question (video) may want to add a derived predicate `right` and insert the rule:

$$\texttt{right}(\texttt{P1}, \texttt{P2}, \texttt{S}) \leftarrow \texttt{left}(\texttt{P2}, \texttt{P1}, \texttt{S}).$$

A word of caution is in order here. The more complex the logic programs grow, the more inefficient are the associated query processing procedures. Hence, we advocate using such derivation rules with extreme caution when building multimedia systems within our framework; however, we leave it to the system designer (based on available hardware, etc.) to make a decision on this point according to the desired system performance.

4.2 The Frame Data Structure

In this section, we will set up a data structure called a *frame* that can be used to access multimedia information efficiently. We will discuss how frames can be used to implement all the queries described in the preceding section.

Suppose we have n media instances, $\mathcal{M}_1, \ldots, \mathcal{M}_n$ where

$$\mathcal{M}_i = (\mathbf{ST}^i, \mathbf{fe}^i, \lambda^i, \Re_1^i, \Re_2^i, \mathbf{Var}_1^i, \mathbf{Var}_2^i)$$

for $1 \leq i \leq n$. We will have *two* kinds of structures used, in conjunction with each other, to access the information in these n media instances.

1. The first of these, called an `OBJECT-TABLE`, is used to store information about which states (possibly from different media instances) contain a given feature. Thus, for each feature $f \in \bigcup_{i=1}^{n} \mathbf{fe}^i$, a record in the `OBJECT-TABLE` has as its key, the name f, together with a pointer to a list of nodes, each of which contains a pointer to a state (represented by a data structure called a `frame` described below) in which f occurs as a feature. As the `OBJECT-TABLE` is searched using alphanumeric strings as the keys, it is easy to see that the `OBJECT-TABLE` can be organized as a standard hash-table, where relatively fast access methods have been implemented over the years.

2. The second of these structures is a `frame`. It should be noted that the `OBJECT-TABLE` data structure and the `frame` data structure are closely coupled together. With each state $s \in \bigcup_{i=1}^{n} \mathbf{ST}^i$, we associate a pointer which points to a list of nodes, each of which, in turn, points to a feature in the `OBJECT-TABLE` (or rather, points to the first element in the list of nodes associated with that feature).

We now give formal definitions of these structures, and later, we will give examples showing how these structures represent bodies of heterogeneous, multimedia data.

Definition 6 Suppose $\mathcal{M}_i = (\mathbf{ST}^i, \mathbf{fe}^i, \lambda^i, \Re_1^i, \Re_2^i, \mathbf{Var}_1^i, \mathbf{Var}_2^i)$ and `framerep` is a data structure that represents the set, $\mathbf{ST}_i$, of states. Then, for each state $s \in \mathbf{ST}^i$, a *frame* in medium $\mathcal{M}_i$ is a record structure consisting of the fields shown in Figure 3 such that:

1. for each feature $f \in \lambda^i(s)$, there is a node in `flist` having f as the `info` field of that node, and

2. if f occurs in the `info` field of a node in `flist`, then $f \in \lambda^i(s)$, and

3. if $f \in \mathbf{fe}^i$ is a feature, then there is an `object` whose `objname` is f and such that the list pointed to by the `link2` field of this `object` is the list of all states in which f is a feature, i.e. is the list of all states $s \in \mathbf{ST}$ such that $f \in \lambda^i(s)$.

4. We assume that all `feature-state` relations are stored as a logic program as specified in Section 4.1.

The above definition specifies frames independently of the medium (e.g. audio, video, latex file, quadtree, bit maps, etc.) used to store the specific data involved. The internal representation of the structures are specified using the data type `framerep` listed (and intentionally not defined) above. When several different data structures are being simultaneously used, we will use $\mathtt{framerep_1}, \ldots, \mathtt{framerep_k}$ to denote the different instantiated structures. Some examples of data representable as frames are the following:

```
frame = record of
   name:  string;  /* name of frame */
   frametype: string; /* type of frame: audio, video, etc. */
   rep:  ^framerep ; /* disk address of internal frame representation */
   flist: ^node1  ; /* feature list */
   end record

node1 = record of
   info:  string ;  /* name of object */
   link:  ^node1 ;  /* next node in list */
   objid: ^object ; /* pointer to object structure named in "info" field */
   end record;

object = record of
   objname:  string ; /* name of object */
   link2  :  ^node2 ; /* list of frames */
   end record

node2 = record of
   frameptr : ^frame ;
   next : ^node2
   end record
```

Figure 4: Data Structure for Frame-Based Feature Storage

- a "still" photograph;
- a video/audio clip;
- a Latex document
- a bitmap of a geographic region, etc.

In addition to the above, for any $\mathcal{M}_i$, we assume that there is an associated string, called the `frametype` of $\mathcal{M}_i$. Intuitively, this string may be "video," "audio," etc. Let us now consider a couple of very, very simple examples below to see how a collection of objects can be represented as a frame.

Example 12 (Indexing for a Single Medium) Let us return to the Clinton-example and reconsider the two video-clips v_1 and v_2 in Figure 2. The first video clip shows three humans who are identified as George Bush, Bill Clinton, and Richard Nixon. The second clip shows two humans, identified as Bill Clinton and Janet Reno.

This set of two records contain *four* significant objects – Bush, Clinton, Nixon and Reno. Information about these four objects, and the two photographs may be stored in the following way.

Suppose $\mathtt{v}_1$ and $\mathtt{v}_2$ are variables of type `frame`. Set:

$$\begin{aligned} \mathtt{v}_1.\mathtt{rep} &= 100 \\ \mathtt{v}_2.\mathtt{rep} &= 590 \end{aligned}$$

specifying that the disk address at which the video-clips are stored are 100 and 590, respectively. Let us consider $\mathtt{v}_1$ and $\mathtt{v}_2$ separately.

- the field $\mathtt{v}_1$`.flist` contains a pointer to a list of three nodes of type `node1`. There are *three* nodes in the feature list because there are three objects of interest in video-frame $\mathtt{v}_1$. Each of these three nodes represents information about the objects of interest in video-frame $\mathtt{v}_1$.
  - the first node in this list has, in its `info` field, the name `BUSH`. It also contains a pointer, `P1` pointing to a structure of type `object`. This structure is an object-oriented representation of the object `BUSH` and contains information about *other* video-frames describing George Bush (i.e. a list of video-frames `v` such that for some node `N` in `v`'s `flist`, `N.info` = `BUSH`.) The list of video-frames in which `BUSH` appears as a "feature" in the manner just described is pointed to by the pointer `P1.link2` = (($\mathtt{v}_1$`.flist`)`.objid`)`.link2`. In this example that uses only two video-frames, the list pointed to by (($\mathtt{v}_1$`.flist`)`.objid`)`.link2` contains only one node, viz. a pointer back to $\mathtt{v}_1$ itself, i.e. (($\mathtt{v}_1$`.flist`)`.objid`)`.link2` points to $\mathtt{v}_1$.
  - the second node in this list has, in its `info` field, the name `CLINTON`. It also contains a pointer, `P2` pointing to alist of video-frames in which `CLINTON` appears as a "feature." In this case, `P2.link2` points to a list of two elements; the first contains $\mathtt{v}_1$, while the second points to $\mathtt{v}_2$.

- the third node in this list has, in its `info` field, the name `NIXON`. The rest is analogous to the situation with `BUSH`.

- the field $\mathtt{v}_2$`.flist` contains a pointer to a list of two nodes of type `node1`. There are *two* nodes because there are two objects of interest in video-frame $\mathtt{v}_2$.
  - the first node in this list has, in its `info` field, the name `CLINTON`. The `objid` field in this node contains the pointer $P2$ (the same pointer as in item 12 above. The values of the fields in the node pointed to by `P2` have already been described in item 12 above.
  - the second node in this list has, in its `info` field, the name `RENO`. The `objid` field in this node contains a pointer, `P4`. The node pointed to by `P4` has the following attributes: `P4.objname` = `RENO`, while `P4.link2` points to the start address where $\mathtt{v}_2$ is stored.

Figure 4 shows a diagrammatic representation of the storage scheme used to access the two video-frames described above. In this example, the `OBJECT-TABLE` is

bush	P1
clinton	P2
nixon	P3
reno	P4

$\diamond\diamond$

The main advantages of the indexing scheme articulated above are that:

1. queries based both on "object" as well as on "video frame" can be easily handled (cf. examples below). In particular, the `OBJECT-TABLE` specifies where the information pertaining to these four objects is kept. Thus, retrieving information where accesses are based on the objects in the table can be easily accomplished (algorithms for this are given in the next section).

2. the data structures described above are *independent* of the data structures used to physically store an image/picture. For instance, some existing pictures may be stored as bit-maps, while others may be stored as quad-trees. The precise mechanism for storing a picture/image does not affect our conceptual design. In this paper, we will not discuss precise ways of storing the `OBJECT-TABLE` – any standard hashing technique should address this problem adequately.

3. Finally, as we shall see in Example 14 below, the *nature* of the medium is irrelevant to our data structure (even though Example 12 uses a single medium, it can be easily expanded to multiple media as illustrated in Example 14 below).

Example 13 Let us return to the Clinton-example, and the two video-frames shown in in Figure 2. Let $(\exists \mathtt{X}, \mathtt{Y})\mathtt{with}(\mathtt{X}, \mathtt{Y})$ denote the query: "Given a value of `X`, find all people `Y` who appear in a common video-frame with person `X` ?" Thus, for instance, when `X` = `CLINTON`, `Y` consists of `RENO,NIXON` and `BUSH`. When `X=RENO`, then `Y` can only be `CLINTON`.

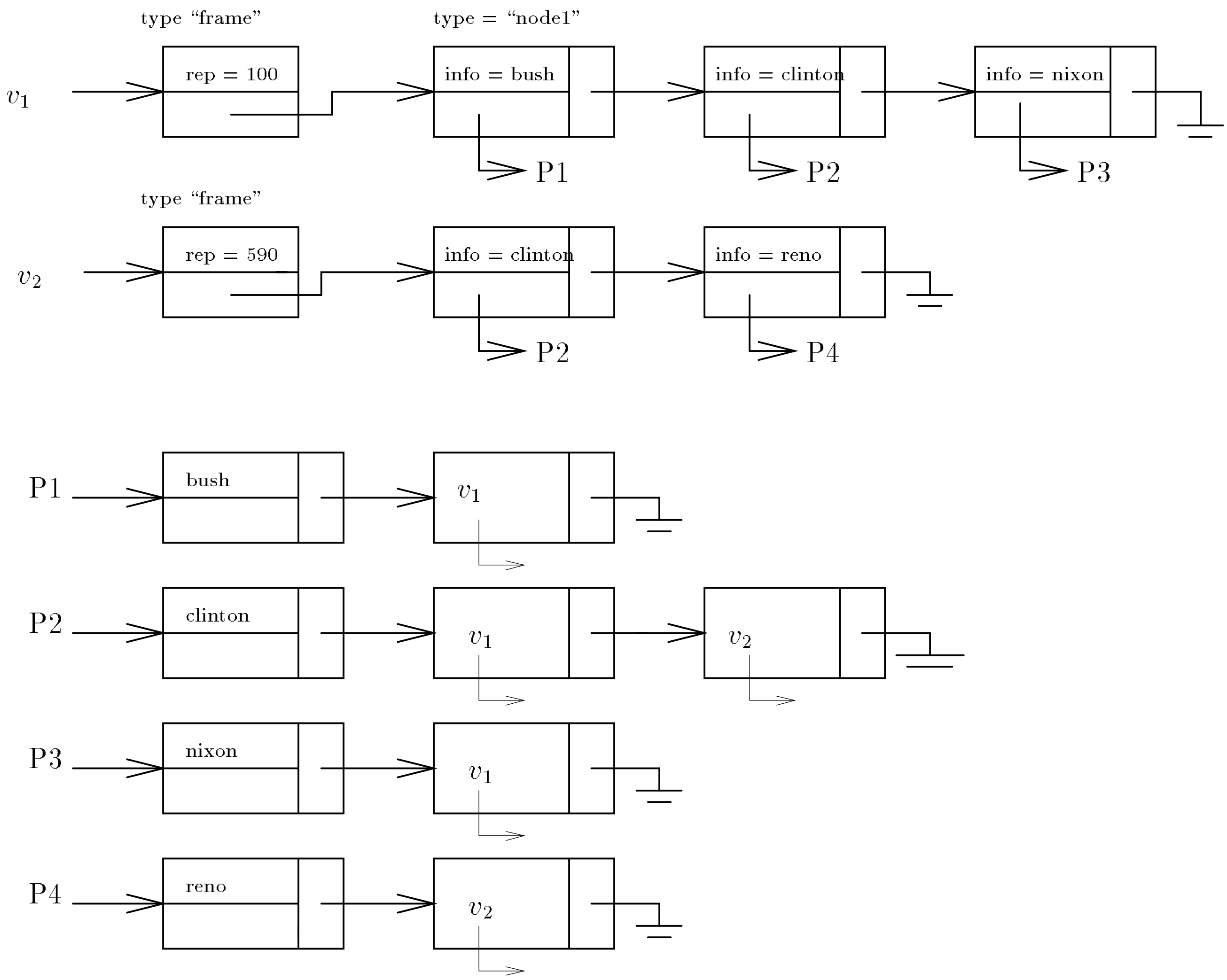


Figure 5: Data Structure for the 2 Video-Frame Example

Such a query can be easily handled within our indexing structure as follows: When `X` is instantiated to, say, `CLINTON`, look at the `object` with `objname` = `CLINTON`. Let `N` denote the node (of type `object`) with its `objname` field set to `CLINTON`. The value of `N` can be easily found using the `OBJECT-TABLE`. `N`.$link2$ is a list of nodes, `N'` such that `N'.frameptr` points to a frame with Clinton in it. For each node `N'` in the list pointed to by `N`.$link2$, do the following: traverse the list pointed to by (`N'.frameptr`).`flist`. Print out the value of ((`N'.frameptr`).`flist`).`objname` for every node in the list pointed to by (`N'.frameptr`).`flist`. Repeat this process for each node in the list pointed to by `N.link2`. $\diamond\diamond$

The following example shows how the same data structure described for storing frames can be used to store *not only video data, but also audio data, as well as data stored using other media.*

Example 14 (Using the Frame Data Structure for Multimedia Information) Suppose we return to example 12, and add two more frames – one is the audio-frame a_1 from the Clinton-example, while the other is the structured document d_1 from the Clinton example. Note that in Example 12, the structure used to store a picture/video-clip did not affect the design of a frame. Hence, it should be (correctly) suspected that the same data structure can be used to store audio data, document data, etc.

We know that our audio-frame a_1 is a text read by Bill Clinton, and that it is about the World Health Organization (WHO, for short). Then we can create a pointer, a_1 (similar to the pointers v_1 and v_2 in Example 12). The pointer a_1 points to a structure of type `frame`. Its feature list contains two elements, `CLINTON` and `WHO` referring to the fact that this audio-clip has two objects of interest. The list pointed to by `P2` is then updated to contain an extra node specifying that a_1 is an address where information about Clinton is kept. Furthermore, the pointer associated with the object `WHO` in the `OBJECT-TABLE` is `P5` which points to an object called `WHO`. The list of frames associated with `P5` consists of just one node, viz. a_1.

We also know that the document d_1 is a position statement by the `WHO` about `CLINTON`. Then we have a new pointer, d_1 (similar to the pointers v_1 and v_2 in Example 12). The pointer d_1 points to a structure of type `frame`. Its feature list contains two elements, `CLINTON` and `WHO` referring to the fact that this audio-clip has two objects of interest. The list pointed to by `P2` is then updated to contain an extra node specifying that d_1 is an address where information about Clinton is kept. Furthermore, the pointer list of frames associated with the entry in the `OBJECT-TABLE` corresponding to `WHO`, i.e., `P5`, is updated to consist of an extra node, viz. d_1.

Figure 6 contains the new structures added to Figure 45in order to handle these two media. $\diamond\diamond$

4.3 Query Processing Algorithms

In this section, we will develop algorithms to answer queries of the form described in Section 4.1. As queries are existentially closed conjunctions of atoms, and as atoms can only

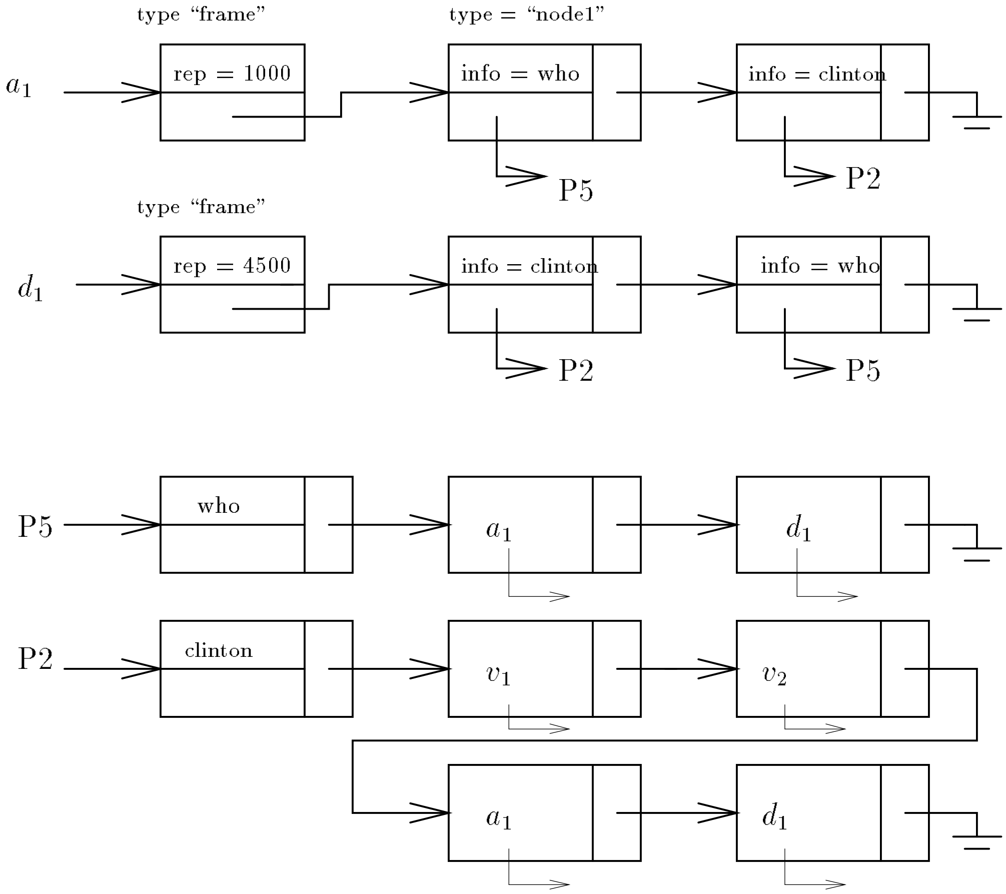


Figure 6: Data Structure for Multimedia-Frame Example

be constructed in certain ways, we will first discuss how atomic queries can be answered (depending on the kinds of atoms involved) and then show how conjunctive queries can be handled (just as a `JOIN`).

4.3.1 Membership Queries

Suppose we consider a ground atom of the form $\mathtt{t} \in \mathtt{flist(s)}$ where t is an object-name and s is a state. As the query is ground, the answer is either yes or no. The algorithm below shows how such a query may be answered.

```
proc ground_in(t:string; s:↑node1):boolean;
   found := false; ptr := s.flist;
      while (not(found) &  ptr ≠ NIL do
         if (ptr.info = t) then found := true
         else ptr := ptr.link ;
      return found.
end proc.
```

It is easy to see that the above algorithm is linear in the length of $\mathtt{flist(s)}$.

Suppose we now consider non-ground atoms of the form $\mathtt{t} \in \mathtt{flist(s)}$ 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. If it is, then there is no solution to the existential query "$(\exists t) t \in \mathtt{flist}(s)$." Otherwise, simply return the "info" field of $s.\mathtt{flist}$. Thus, this kind of query can be answered in constant time.

(Case 2: s non-ground, t ground) This case is more interesting. t is a feature, and hence, an object. Thus, t must occur in the `OBJECT-TABLE`. Once the location of t in the `OBJECT-TABLE` is found (let us say PTR points to this location), and if PTR.link2 is non-NIL, then return (((PTR.link2).frameptr).name). If PTR.link2 is NIL, then halt – no answer exists to the query "$(\exists s) t \in \mathtt{flist}(s)$." Thus, this kind of query can be answered in time O(k) where k is the length of the list PTR.link2.

(Case 3: s non-ground, t non-ground) In this case, find the first element of the `OBJECT-TABLE` which has a non-empty "link2" field. If no such entry is present in the table, then no answer exists to the query "$(\exists s, t) t \in \mathtt{flist}(s)$." Otherwise, let PTR be a pointer to the first such entry. Return the solution

$$\mathrm{t} = \mathrm{PTR}; \;\; \mathrm{s} = (((\mathrm{PTR.link2}).\mathrm{frameptr}).\mathrm{name}).$$

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

4.3.2 Other Queries

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

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 just joins of answers to their atomic parts. Join queries can be optimized by adapting standard methods to work with our data structures.

4.4 Updates in Multimedia Databases

It is well-known that database systems need to be frequently updated to reflect new information that was not available before, or which reflect corrections to previously existing information. This situation will affect multimedia database systems in the same way current database systems are affected by it. However, *how* these updates are incorporated will change because of the nature of the indexing structures we use. Updates to an integrated multimedia system can be of two types:

1. **Feature Updates within States:** It may be the case that features in a given state were either not identified at all, or were incorrectly identified. For instance, a pattern recognition algorithm which extracts features from video may leave Jack Kemp unclassified simply because he was not on the list of features the system knew about. Am enhanced pattern recognition algorithm pressed into service later may wish to add a new feature, viz. `kemp`, to the list of features possessed by a certain video-frame. In the same vein, a Bill Clinton look alike may mistakenly be classified as Bill Clinton and later, it may become apparent that the feature `clinton` should be deleted from this video-clip (as Clinton is not in the video). We show, in Section 4.4.1 and 4.4.2 below, how features can be efficiently added and deleted from states.

2. **State Updates:** When new states arrive they need to be processed and inserted into the multimedia database system. For instance, new video-information showing Clinton speaking at various places may need to be added. In the same, deletions of existing states (that have been determined to be useless) may also need to be accomplished. Section 4.4.3 and 4.4.4 specify how these insertions and deletions may be accomplished.

4.4.1 Inserting Features into States

In this section, we develop a procedure called feature_add that takes a feature f and a pre-existing state s as input, and adds f to state s. This must be done in such a way that the underlying indexing structures are modified so that the query processing algorithms can access this new data.

proc feature_add(f:feature; s:state);
  Insert f into `OBJECT-TABLE` at record R.

Let N be the pointer to state S.
Set R to N.
Add R to the list of features pointed to by node N.
end proc.

It is easy to see that this algorithm can be executed in constant time (modulo the complexity of insertion into the `OBJECT-TABLE`).

4.4.2 Deleting Features From States

In this section, we develop a procedure called feature_del that takes a pre-existing feature f and a pre-existing state s as input, and deletes f from s's feature list.

proc feature_del(f:feature; s:state);
Find the node N in s's `flist` having N.`info` $= f$.
Set T to N.`objid`.
Delete N.
Examine the list of states in $T.link2$ and delete the node whose `frameptr` field is s.
end proc.

It is easy to see that this algorithm can be executed in linear time (w.r.t. the lengths of the lists associated with s and f, respectively).

4.4.3 Inserting New States

Adding a new state s is quite easy. All that needs to be done is to:

1. Create a pointer S to a structure of type `frame` to access state s.
2. Insert each feature possessed by state s into S's `flist`.
3. For each feature f in s's `flist`, add s into the list of frames pointed to by f's `frameptr` field.

It is easy to see that the complexity of inserting a new state is linear in the length of the feature list of this state.

4.4.4 Deleting States

The procedure to delete state s from the index structure is very simple. For each feature f in s's `flist`, delete s from the list pointed to by f.`frameptr`. Then return the entire list pointed to by S (where S is the pointer to the `frame` representing s) to available storage. It is easy to see that the complexity of this algorithm is

$$length(\texttt{flist}(s)) + \sum f \in \texttt{flist}(s)\ell(f.\texttt{frameptr})$$

where $length(\texttt{flist}(s))$ is the number of features s has, and $\ell(f.\texttt{frameptr})$ is the length of the list pointed to by $f.\texttt{frameptr}$, i.e. the number of states in which f appears as a feature.

In this section, we have made three contributions: we have defined a logical query language for multimedia databases, an indexing structure that can be used to integrate information across these different media-instances, query processing procedures to execute queries in the query language using the indexing structure, and database update procedures that use the indexing structure based on improved data.

5 Multimedia Presentations

The description of multimedia information systems developed in preceding sections is completely static. It provides a query language for a user to integrate information stored in these diverse media. However, in many real-life applications, different frames from different media sources must come together (i.e. be synchronized) so as to achieve the desired communication effect. Thus, for example, a video-frame showing Clinton giving a speech would be futile if the audio-track portrayed Socks the cat, meowing. In this section, we will develop a notion of a *media-event* – informally, a media event is a concatenation of the states of the different media at a given point in time. The aim of a media presentation is to achieve a desired sequence of media-events, where each individual event achieves a coherent synchronization of the different media states. We will show how this kind of synchronization can be viewed as a form of constraint-solving, and how the generation of appropriate media-events may be viewed as query processing. In other words, we suggest that:

$$\begin{aligned} \text{Generation of Media Events} &= \text{Query Processing.} \\ \text{Synchronization} &= \text{Constraint Solving.} \end{aligned}$$

5.1 Generation of Media Events = Query Processing

In the sequel, we will assume that we have an underlying multimedia system $\mathbf{MMS} = \{\mathcal{M}_1, \ldots, \mathcal{M}_n\}$ where $\mathcal{M}_i = (\mathbf{ST}^i, \mathbf{fe}^i, \lambda^i, \Re_1^i, \Re_2^i, \mathbf{Var}_1^i, \mathbf{Var}_2^i)$.

Definition 7 A *media-event* w.r.t. $\mathbf{MMS}$ 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 medium $\mathcal{M}_i$, 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 15 Suppose we return to the Clinton Example, and suppose we consider the video-frame shown in Figure 2(b). Let us suppose that this represents the state s_1 when Reno was sworn in as Attorney General, and let us suppose there is an audio-tape a_4 describing the events. Then the pair (s_1, a_4) is a media-event; intuitively, this means that state s_1 (video) and state a_4 (audio) must be "on" simultaneously. (We will go into details of synchronization in a later section. $\diamondsuit\diamondsuit$

We now formally define the notion of "satisfaction" of a formula in the query language by a media-event.

Definition 8 Suppose $\mathbf{me} = (s_1, \ldots, s_n)$ is a media event w.r.t. the multimedia system $\mathbf{MMS} = \{\mathcal{M}_1, \ldots, \mathcal{M}_n\}$ 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 $F = \texttt{frametype}(a, b)$ 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 $F = (c \in \texttt{flist}(b))$, 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 $(t_1, \ldots, t_n, s) \in \phi \in \Re^i_2$.
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[2]

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

Definition 9 A *multimedia specification* is a sequence of queries $Q_1, Q_2, \ldots$ to $\mathbf{MMS}$.

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 10 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.

[2]The notion of a "free" variable is the standard one, cf. Shoenfield [19]).

Example 16 (Multimedia Event Generation Example) Let us suppose that we have recourse to a very small multimedia library consisting of five video-frames, and five audio-frames. Thus, there are two media involved, $\mathcal{M}_1$ (audio) and $\mathcal{M}_2$ (video), and there are five states in each of these media. The tables below specify the audio states and video states, respectively:

Audio Frame Name	Features	Video Frame Name	Features
a_1	clinton	v_1	clinton, gore, bush
a_2	clinton, socks	v_2	clinton, gore
a_3	gore	v_3	clinton
a_4	bush	v_4	gore, reno
a_5	clinton, gore	v_5	clinton, gore, reno

Let us now suppose that the initial media-event is some pair $\mathbf{me}_0 = (a_0, v_0)$ consisting of a blank, i.e. the feature lists for both media are initially empty (i.e. there is no video, and no audio at time 0). Suppose we consider the evolution of this multimedia system over three units of time. Let us consider the multimedia specification Q_1, Q_2, Q_3 where:

$$\begin{aligned}
Q_1 &= (\exists \mathtt{S}_1, \mathtt{S}_2)(\mathtt{frametype}(\mathtt{S}_1, \mathtt{video})\,\&\,\mathtt{frametype}(\mathtt{S}_2, \mathtt{audio})\,\&\,\mathtt{clinton} \in \mathtt{flist}(\mathtt{S}_1)\,\&\\
&\quad \mathtt{gore} \in \mathtt{flist}(\mathtt{S}_1)\,\&\,\mathtt{clinton} \in \mathtt{flist}(\mathtt{S}_2)).\\
Q_2 &= (\exists \mathtt{S}_1, \mathtt{S}_2)(\mathtt{frametype}(\mathtt{S}_1, \mathtt{video})\,\&\,\mathtt{frametype}(\mathtt{S}_2, \mathtt{audio})\,\&\,\mathtt{clinton} \in \mathtt{flist}(\mathtt{S}_1)\,\&\\
&\quad \mathtt{gore} \in \mathtt{flist}(\mathtt{S}_1)\,\&\,\mathtt{gore} \in \mathtt{flist}(\mathtt{S}_2)).\\
Q_3 &= (\exists \mathtt{S}_1, \mathtt{S}_2)(\mathtt{frametype}(\mathtt{S}_1, \mathtt{video})\,\&\,\mathtt{frametype}(\mathtt{S}_2, \mathtt{audio})\,\&\,\mathtt{clinton} \in \mathtt{flist}(\mathtt{S}_1)\,\&\\
&\quad \mathtt{gore} \in \mathtt{flist}(\mathtt{S}_1)\,\&\,\mathtt{bush} \in \mathtt{flist}(\mathtt{S}_1)\,\&\,\mathtt{clinton} \in \mathtt{flist}(\mathtt{S}_2)\,\&\,\mathtt{gore} \in \mathtt{flist}(\mathtt{S}_2)).
\end{aligned}$$

Observe that query Q_1 can be satisfied by any substitution that sets S_1 to an element of $\{v_1, v_2, v_5\}$ and S_2 to an element of $\{a_1, a_2, a_5\}$ – thus there nine possible combinations of audio/video that could come up in response to this query at time 1. Had the user wanted to eliminate some of these nine possible s/he should have added further conditions to the query.

When query Q_2 is processed, S_1 can be set to any of $\{v_1, v_2, v_5\}$ as before, but S_2 may be set only to one of $\{a_3, a_5\}$. Thus, any of these six possible audio-video combinations would form a legitimate media event at time 2.

Lastly, to satisfy Q_3, S_1 must be set to v_1 and S_2 must be set to a_5; no other media-event would satisfy Q_3.

As a final remark, we observe that not all queries are necessarily satisfiable (and hence, for some queries, it may be impossible to find an appropriate media-event). For instance, consider the query $(\exists S)(\mathtt{frametype}(S, \mathit{audio})\,\&\,\mathit{reno} \in \mathtt{flist}(S))$. It is easily seen that there is no audio-frame in our library which has Reno in its feature list, and hence, this query is not satisfiable. $\Diamond\Diamond$

5.2 Synchronization = Constraint Solving

In preceding sections, we have not considered the problem of synchronization. In particular, it is naive to assume, as done previously, that queries $Q_1, Q_2, Q_3, \ldots$ will be posed one after the other at times $1, 2, 3, \ldots$, respectively. Rather, experience with multimedia systems existing in the market suggests that a query may be "in force" for a certain period of time. In other words, the multimedia system (or the Multimedia Integrator shown in Figure 1) may be given the following inputs:

- a sequence $Q_1, Q_2, \ldots, Q_k$ of queries indicating that query Q_1 must be answered (i.e. a media-event that satisfies query Q_1 be "brought up"), followed by a media-event satisfying query Q_2, etc., and
- a deadline d by which the entire sequence of media-events must be completed, and
- for each query Q_i, $1 \leq i \leq n$, a *lower* bound $\texttt{LB}_i$ and an upper bound $\texttt{UB}_i$ reflecting how long the media-event corresponding to this query should be "on." $\texttt{LB}_i$ and $\texttt{UB}_i$ are integers – we assume discrete time in our framework.

The Multimedia Integrator's job is to:

- **(Task 1)** Answer the queries $Q_1, \ldots, Q_n$, i.e. find media events $\mathbf{me}_1, \ldots, \mathbf{me}_n$ that satisfy the above queries.
- **(Task 2)** Schedule the actual start time and end time of each media-event, and ensure that this time achieves the lower-bound and upper-bound alluded to earlier.

Task 1 has already been addressed in the preceding section; we now address task 2. We show that the scheduling problem is essentially a constraint satisfaction problem which may be formulated as follows.

Individual Media Constraints. Let s_i be a variable denoting the start time of media-event $\mathbf{me}_i$, and let e_i be a variable denoting the execution time of media-event $\mathbf{me}_i$ – it is important to note that the values of these variables may *not* be known initially. Then, as we know that media-event $\mathbf{me}_i$ must be "on" for between $\texttt{LB}_i$ and $\texttt{UB}_i$ time units, we know that:

$$\texttt{LB}_i \leq e_i \leq \texttt{UB}_i$$

is a constraint that must be satisfied within our framework. Furthermore, the constraints

$$s_i \geq 0\,;\; e_i \geq 0$$

must be satisfied as well.

Synchronization. The only remaining thing is to ensure that the media-event to query Q_{i+1} starts immediately after the media-event satisfying query Q_i. This may be achieved by the following constraint:

$$s_{i+1} = s_i + e_i$$

where $i < n$.

Deadline Constraint. Finally, we need to specify that the deadline has to be achieved, i.e. the completion-time of the last media-event must be achieved on, or before the deadline. This can be stated as:

$$s_i + e_i \leq d.$$

Together with the constraint that all variables (i.e. $s_1, \ldots, s_n, e_1, \ldots, e_n$) are non-negative, the solutions of the above system of equations specify the times at which the media-events corresponding to queries $Q_1, Q_2, \ldots, Q_n$ must be "brought up" or "activated".

5.3 Internal Synchronization

In the preceding section, we have assumed that though a media-event involves a multiplicity of participating media, all these different media-states are brought up simultaneously and synchronously. We call the problem of synchronizing the individual media-states participating in a particular media-event *internal synchronization* as this is related to the media-event generated by a specific query. An easy solution is to assume that while the media-event corresponding to query Q_i is "on," the system computes a media-event, $\mathbf{me}_{i+1}$, corresponding to query Q_{i+1} and stores the individual media-states in a buffer. Thus, there is a buffer, $\mathtt{BUF}_i$ corresponding to each media-instance, $\mathcal{M}_i$. In the next section, we discuss how these buffers can be organized and managed.

5.4 Media Buffers

Internal synchronization requires that at any given point in time, if the media-event $\mathbf{me}_i$ corresponding to query Q_i is "on," then the media-event $\mathbf{me}_{i+1}$ corresponding to query Q_{i+1} is ready and loaded in the buffers. Let

$$\begin{aligned} \mathbf{me}_i &= (s_1, \ldots, s_n) \\ \mathbf{me}_{i+1} &= (s'_1, \ldots, s'_n). \end{aligned}$$

Then, for each $1 \leq i \leq n$, it suffices to store the set of differences (this set is denoted δ_i) between state s_i and state s'_i. These two states reflect, respectively, the status of media-instance $\mathcal{M}_i$ when query Q_i is "on" and when query Q_{i+1} is "on." For instance, if media-event $\mathcal{M}_i$ is of frametype `video`, then s_i and s'_i may be pictures. Suppose, for instance, that we are discussing an audio-video presentation (say of some cowboys), and there are three differences between states s_i and s'_i, i.e. $\delta_i = \{d_1, d_2, d_3\}$ where:

1. d_1 represents a pistol which just appeared in a cowboy's hand,
2. d_2 represents a dog turning his head,
3. d_3 represents a leaf falling in the breeze.

Then it may be the case that d_1 is the "most important" of these changes, d_2 is the second most important, and d_3 is the least important of these differences. Hence, it may be critical, when bringing up state s'_i from the buffer, that d_1 be brought up first, then d_2 and only finally, d_3.

In general, we assume that associated with each medium $\mathcal{M}_i$, we have a *classification function*, $\texttt{cf}_i$, which assigns, to each difference, a non-negative integer called that difference's classification level. The buffer, $\texttt{BUF}_i$, associated with media-instance $\mathcal{M}_i$ is organized as a prioritized queue – all differences with priority 1 are at the front of the queue, all differences with priority 2 are next in the queue, and so on. Thus, when the queue is flushed (i.e. when the process of bringing state s_i' "up" is started), then the differences are brought up in the specified priority order. Note that if two differences are both labeled with the same classification level, then it is permissible to bring them up in any order relative to each other.

6 Related Work

There has been a good deal of work in recent years on multimedia. Zdonik [29] has specified various roles that databases can play n complex multimedia systems [29, p.409]. One of these is the logical integration of data stored on multiple media – this is the topic of this paper.

Kim et. al. [27, 28] show how object-oriented databases (with some enhancements) can be used to support multimedia applications. Their model is a natural extension of the object-oriented notions of instantiation and generalization. The general idea is that a multimedia database is considered to be a set of objects that are inter-related to each other in various ways. The work reported here is compatible to that [27, 28] in that the frames and features in a media-instance may be thought of as objects. There are significant differences, however, in how these objects are organized and manipulated. For instance, we support a logical query language (Kim et. al. would support on object-oriented query language), and we support updates (Kim et. al. can do so as well but using algorithms compatible with their object-oriented model). We have analyzed the complexity of our query processing and update algorithms. Furthermore, the link between query processing and generation of media events is a novel feature of our framework, not present in [27, 28]. Last, but not least, we have developed a formal theoretical framework within which multimedia systems can be formally analyzed, and we have shown how various kinds of data representations on different types of media may be viewed as special cases of our framework.

Oomoto and Tanaka [15] have defined a video-based object oriented data model, OVID. What the authors do primarily is to take pieces of video, identify meaningful features in them and link these features especially when consecutive clips of video share features. Our work deals with integrating multiple media and provide 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). The authors have developed feature identification schemes (which we have not) and this complements our work. In a similar vein, Arman et. al. [2] develop techniques to create large video databases by processing incoming video-data so as to identify features and set up access structures. Another piece of relevant related work is that of the QBIC (Query by Image Content) system of Barber et. al. [3] at IBM, They develop indexing techniques to query large video databases by images – in other words, one may ask queries of the form "Find me all pictures in which image I occurs." Retrievals are done on the basis of similarity rather than on a perfect match. In constrast to our theoretical framework, [3] shows how features may be identified (based on similarity) in video, and how queries can be formulated in the video domain.

Cardenas, et. al. [5] have developed a query language called PICQUERY+ for querying certain kinds of federated multimedia systems. The spirit of their work is similar to ours in that both works attempt to devise query languages that access heterogeneous, federated multimedia databases. The differences, though, are in the following: 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 [5]. However, there are many features in [5] that our framework does not currently possess – two of these are temporal data and uncertain information. Such features form a critical part of many domains (such as the medical domain described in [5]), and we look forward to extending our multimedia work in that direction, in keeping with a similar effort we have made previously [21] for integrating time, uncertainty, data structures, numeric constraints and databases.

Little and Ghafoor [13] have developed methods for satisfying temporal constraints in multimedia systems. This relates to our framework in the following way: suppose there are temporal constraints specifying how a media-buffer (as defined in this paper) must be flushed. Little and Ghafoor [13] show how this can be done. Hence, their methods can be used in conjunction with ours. In a similar vein, Prabhakaran and Raghavan [16] show how multimedia presentations may be synchronized.

Other related works are the following: Gaines and Shaw [10] develop an architecture to integrate multiple document representations. Eun et. al. [6] show how Milner's Calculus of Communicating Systems can be used to specify interactive multimedia but they do not address the problem of querying the integration of multiple media. Gemmel and Christodoulakis [7] study delay-sensitive data using an approach based on constrained block allocation. This work is quite different from ours.

Finally, we note that multimedia databases form a natural generalization of heterogeneous databases which have been studied extensively in [1, 8, 11, 12, 18, 20, 21, 22, 23, 24, 25, 26, 30]. How exactly the work on heterogeneous databases is applicable to multimedia databases remains to be seen, but clearly there is a fertile area to investigate here.

7 Conclusions

As is evident from the "Related Work" section, 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. In the Introduction, we raised four questions. Below, we review the progress made in this paper towards answering those four questions, and indicate directions for future work along these lines.

- **What are multimedia database systems and how can they be formally/ mathematically defined so that they are independent of any specific application domain ?**

  Accomplishments: In this paper, we have argued that in all likelihood, the designer of the Multimedia Integrator shown in Figure 1 will be presented with a collection of *pre-existing* databases on different types of media. The designer must build his/her algorithms "on top" of this pre-existing representation – delving into the innards of any of these representations is usually prohibitive, and often just plain impossible. Our framework provides a method to do so once features and feature-state relationships can be identified.

  Future Work: However, we have not addressed the problem of identifying features or identifying feature-relationships. For instance, in the Clinton Example (cf. Figure 2), Clinton is to the left of Nixon. However, from a bitmap, it is necessary to determine that Clinton and Nixon are actually in the picture, and that Clinton is to the left of Nixon. Such determinations depend inherently on the medium involved, and the data structure(s) used to represent the information (e.g. if the bitmap was replaced by a quadtree in the pictorial domain itself, the algorithms would become vastly different). Hence, feature identification in different domains is of great importance and needs to be addressed.

- **Can indexing structures for multimedia database systems be defined in a similar uniform, domain-independent manner ?**

  Accomplishments: We have developed a logic-based query language that can be used to execute various kinds of queries to multimedia databases. This query language is extremely simple (using nothing more than relatively standard logic), and hence it should form an easy vehicle for users to work with.

  Future Work: The query language developed in this paper does not handle uncertainty in the underlying media and/or temporal changes in the data. These need to be incorporated into the query language as they are relevant for various applications such as those listed by Cardenas et. al. [5].

- **Is it possible to uniformly define query languages and access methods based on these indexing structures ?**

  Accomplishments: We have developed indexing structures for organizing the features (and properties of the features) in a given media-instance, and we have developed algorithms that can be used to answer queries (expressed in the logical query language described in the paper). These algorithms have been shown to be computable in polynomial-time.

  Future Work: Supporting more complex queries involving aggregate operations, as well as uncertainty and time in the queries (see preceding bullet) will require further work.

- **Is it possible to uniformly define the notion of an update in multimedia database systems and to efficiently accomplish such updates using the above-mentioned indexing structures ?**

  Accomplishments: We have defined a notion of an update to multimedia database systems that permits new features and states to be inserted into the underlying indexing structure when appropriate. Similarly deletions of old features and states are also supported. We have shown that these algorithms can be executed efficiently.

  Future Work: Of the update algorithms developed in this paper, the algorithm for deleting states is less efficient than the other three. In applications that require large-scale state deletions, it may be appropriate to consider alternative algorithms (and possibly alternative indexing structures as well).

- **What constitutes a multimedia presentation and can this be formally/mathematically defined so that it is independent of any specific application domain ?**

  Accomplishments: We prove that there is a fundamental connection between query processing and the generation of media-events. What this means is that a media presentation can be generated by a sequence of queries. This is useful because it may be relatively easy to specify a query articulating the criteria of importance – the system may be able to respond by picking any one of several media-events that satisfies this query. In addition, we show that synchronization really boils down to solving constraints.

  Future Work: A great deal of work has been done on synchronizing multimedia streams in a network[13, 16]. It should be possible to take advantage of these works to enhance the synchronization of answers to a query.

Acknowledgements. We are extremely grateful to Sushil Jajodia for many enlightening conversations on the topic of multimedia databases. We have also benefited from conversations with Sandeep Mehta, Raymond Ng, S. V. Raghavan and Satish Tripathi. We are grateful to H. Garcia-Molina and C. Faloutsos for drawing our attention to [3].

References

[1] S. Adali and V.S. Subrahmanian. (1993) *Amalgamating Knowledge Bases, II: Algorithms, Data Structures and Query Processing*, Univ. of Maryland CS-TR-3124, Aug. 1993. Submitted for journal publication.

[2] 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.

[3] R. Barbet, W. Equitz, C. Faloutsos, M. Flickner, W. Niblack, D. Petkovic, and P. Yanker. (1993) *Query by Content for Large On-Line Image Collections*, IBM Research Report RJ 9408, June 1993.

[4] J. Benton and V.S. Subrahmanian. (1993) *Hybrid Knowledge Bases for Missile Siting Problems*, accepted for publication in 1994 Intl. Conf. on Artificial Intelligence Applications, IEEE Press.

[5] 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.

[6] S.B. Eun, E.S. No, H.C. Kim, H. Yoon, and S.R. Maeng. (1993) *Specification of Multimedia Composition and a Visual Programming Environment*, First ACM Intl. Conf. on Multimedia, pps 167–174.

[7] D.J. Gemmel and S. Christodoulakis. (1992) *Principles of Delay-Sensitive Multimedia Data Storqage and Retrieval*, ACM Trans. on Information systems, 10, 1, pps 51–90.

[8] J. Grant, W. Litwin, N. Roussopoulos and T. Sellis. (1991) *An Algebra and Calculus for Relational Multidatabase Systems*, Proc. First International Workshop on Interoperability in Multidatabase Systems, IEEE Computer Society Press (1991) 118-124.

[9] F. Hillier and G. Lieberman. (1986) *Introduction to Operations Research*, 4th edition, Holden-Day.

[10] B. R. Gaines and M. L. Shaw. (1993) *Open Architecture Multimedia Documents*, Proc. First ACM Intl. Conf. on Multimedia, pps 137–146.

[11] W. Kim and J. Seo. (1991) *Classifying Schematic and Data Heterogeneity in Multidatabase Systems*, IEEE Computer, Dec. 1991.

[12] A. Lefebvre, P. Bernus and R. Topor. (1992) *Querying Heterogeneous Databases: A Case Study*, draft manuscript.

[13] T.D.C. Little and A. Ghafoor. (1993) *Interval-Based Conceptual Models of Time-Dependent Multimedia Data*, IEEE Trans. on Knowledge and Data Engineering, 5, 4, pps 551–563.

[14] J. Lloyd. (1987) *Foundations of Logic Programming*, Springer Verlag.

[15] 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.

[16] B. Prabhakaran and S. V. Raghavan. (1993) *Synchronization Models for Multimedia Presentation with User Participation*, First ACM Intl. Conf. on Multimedia, pps 157–166.

[17] H. Samet. (1989) *The Design and Analysis of Spatial Data Structures*, Addison Wesley.

[18] A. Sheth and J. Larson. (1990) *Federated Database Systems for Managing Distributed, Heterogeneous and Autonomous Databases*, ACM Computing Surveys, 22, 3, pp 183–236.

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

[20] A. Silberschatz, M. Stonebraker and J. D. Ullman. (1991) *Database Systems: Achievements and Opportunities*, Comm. of the ACM, 34, 10, pps 110–120.

[21] 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.

[22] 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.

[23] G. Wiederhold. (1992) *Mediators in the Architecture of Future Information Systems*, IEEE Computer, March 1992, pps 38–49.

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

[25] G. Wiederhold, S. Jajodia, and W. Litwin. Dealing with granularity of time in temporal databases. In *Proc. 3rd Nordic Conf. on Advanced Information Systems Engineering*, Lecture Notes in Computer Science, Vol. 498, (R. Anderson et al. eds.), Springer-Verlag, 1991, pages 124–140.

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

[27] D. Woelk, W. Kim and W. Luther. (1986) *An Object-Oriented Approach to Multimedia Databases*, Proc. ACM SIGMOD 1986, pps 311-325.

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

[29] S. Zdonik. (1993) *Incremental Database Systems: Databases from the Ground Up*, Proc. 1993 ACM SIGMOD Conf. on Management of Data, pps 408–412.

[30] R. Zicari, S. Ceri, and L. Tanca. (1991) *Interoperability between a Rule-Based Database Language and an Object-Oriented Language*, Proc. First International Workshop on Interoperability in Multidatabase Systems, IEEE Computer Society Press (1991) 125-135.