QUERYING MULTIMEDIA DATA SOURCES AND DATABASES*

S-K Chang[1], G. Costagliola[2] and E. Jungert[3]

[1]*Department of Computer Science*
University of Pittsburgh

[2]*Dipartimento di Matematica ed Informatica*
Università di Salerno

[3]*Swedish Defense Research Institute (FOA)*

ABSTRACT

To support the retrieval and fusion of multimedia information from multiple sources and databases, a spatial/temporal query language called ΣQL is proposed. ΣQL is based upon the σ−operator sequence and in practice expressible in SQL-like syntax. ΣQL allows a user to specify powerful spatial/temporal queries for both multimedia data sources and multimedia databases, eliminating the need to write different queries for each. A ΣQL query can be processed in the most effective manner by first selecting the suitable transformations of multimedia data to derive the multimedia static schema, and then processing the query with respect to this multimedia static schema. In this paper we illustrate this approach by data fusion examples, investigate multimedia data transformations and provide query processing algorithms.

1. INTRODUCTION

With the rapid expansion of the wired and wireless networks, a large number of soft real-time, hard real-time and non-real-time sources of information need to be processed, checked for consistency, structured and distributed to the various agencies and people involved in an application [16]. In addition to spatial/temporal multimedia databases, it is also anticipated that numerous web sites on the World Wide Web will become rich sources of spatial/temporal multimedia information. The retrieval and fusion of spatial/temporal multimedia information from diversified sources calls for the design of spatial/temporal query languages capable of dealing with both multiple data sources and databases in a heterogeneous information system environment.

Powerful query languages for multiple data sources and databases are needed in applications such as emergency management (fire, flood, earthquake, etc.), tele-medicine, digital library, community network (crime prevention, child care, senior citizens care, etc.), military reconnaissance and scientific exploration (field computing). These applications share the common characteristics that information from multiple sources and databases must be integrated. A typical scenario for information fusion in emergency management may involve live report from a human observer, data collected by a heat sensor, video signal from a camera mounted on a helicopter, etc. Current systems often have preprogrammed, fixed scenarios. In order to enable the end user to effectively retrieve spatial/temporal multimedia information and to discover relevant associations among media objects, a flexible spatial/temporal multimedia query language for multiple data sources and databases should be provided.

* This research has been co-funded by the National Science Foundation, USA, the Swedish National Defence Institute and the Italian National Council of Research (CNR)

Data sources such as camera, sensors or signal generators usually provide continuous streams of data. Such data need to be transformed into abstracted information, i.e., into various forms of spatial/temporal/logical data structures, so that the processing, consistency analysis and fusion of data become possible. The abstracted information does not necessarily represent different levels of knowledge and can be various representations of common knowledge and therefore needs to be integrated and transformed into fused knowledge that is the common knowledge derivable from, and consistent with, the various abstractions.

As an example, information items such as time and number of people can be extracted from multiple data sources such as image, scenario document and sound [21]. Cooperation among various media is carried out by exchanging these information items. In an experimental study involving a TV drama scene, this approach could successfully realize good synchronization between image, scenario and sound, and moreover could also perform personal character identification [21]. As a second example, a recent study first identified known person's names from the text, and then tried to detect corresponding faces from the video stream [17]. As a third example, a video camera is a data source that generates video data. Such video data can be transformed into various forms of abstracted representations including: text, keyword, assertions, time sequences of frames, qualitative spatial description of shapes, frame strings, and projection strings [5]. To describe a frame containing two objects a and b, the text is *a is to the northwest of b*, the keywords are {a, b}, and the assertion is (a northwest b). The x-directional projection string is (u: a < b). The time sequence of three frames C_{t1} , C_{t2} , C_{t3} , is (t: $C_{t1} < C_{t2} < C_{t3}$). Some of these transformations will be explained later.

To support the retrieval and fusion of multimedia information from multiple sources and databases, a spatial/temporal query language called ΣQL is proposed. ΣQL is based upon the σ−operator sequence and in practice expressible in an SQL-like syntax. The natural extension of SQL to ΣQL allows a user to specify powerful spatial/temporal queries for both multimedia data sources and multimedia databases, eliminating the need to write different queries for each. A ΣQL query can be processed in the most effective manner by first selecting the suitable transformations of multimedia data to derive the multimedia static schema, and then processing the query with respect to this multimedia static schema.

Query language for heterogeneous multimedia databases is a new research area and therefore the body of related work only just begins to grow. There has been substantial research on query languages for images and spatial objects, and a survey can be found in [8, 9]. Of these query languages, many are based upon extension of SQL, such as PSQL [19] and Spatial SQL [11]. Next come video query languages where the focus is shifted to temporal constraints [1] and content based retrieval [4]. Recent efforts begin to address query languages involving images, video, audio and text. Vazirgiannis describes a multimedia database system for multimedia objects that may originate from sources such as text, image, video, [22]. The query language QL/G developed by Chan and Zhu supports the querying of geometric data bases and is applicable to both geometric and text data [3], but does not handle temporal constraints. In [18], Oomoto and Tanaka describes an SQL-like query language for video databases, where the emphasis is more on presentation, rather than on retrieval. A multimedia object query language MOQL that extends the object query language OQL is reported in [17]. An interoperable multi-database platform in a client/server environment using a common object model is described in [24], which can provide inter-operations between popular database systems. A related approach is to provide a database integrator (DBI) for customers who have data stored in multiple data sources, typically heterogeneous and/or non-relational, and want to view those data sources as a single logical database from the data and/or metadata perspective [12].

While the above described approaches each address some important issues, there is a lack of unified treatment of queries that can deal with both spatial and temporal constraints from both live data sources and stored databases. Since the underlying databases are complex, the user also needs to write complicated queries to integrate multimedia information. The proposed approach differs from the above in the introduction of a general powerful operator called the σ–operator, so that the corresponding query language can be based upon σ–operator sequences. The paper is organized as follows. The basic concepts of the σ–query is explained in Section 2. Section 3 introduces elements of Symbolic Projection Theory and the general σ–operator, and Secction 4 describes the SQL query language. An illustration of data fusion using the σ–query is presented in Section 5. Section 6 formalizes the representation for multimedia sources and then gives query processing examples. The techniques for query processing are explained in Section 7. Transformational analysis is described in Section 8. In Section 9 we discuss further research topics.

2. BASIC CONCEPTS OF THE σ–QUERY

As mentioned in Section 1, the σ–query language is a spatial/temporal query language for information retrieval from multiple sources and databases. Its strength is its simplicity: the query language is based upon a single operator - the σ–operator. Yet the concept is natural and can easily be mapped into an SQL-like query language. The σ–query language is useful in theoretical investigation, while the SQL-like query language is easy to implement and is a step towards a user-friendly visual query language. An example is illustrated in Figure 1. The source R, also called a universe, consists of time slices of 2D frames. To extract three pre-determined time slices from the source R, the query in mathematical notation is: $\sigma_t (t_1 , t_2 , t_3) R$.

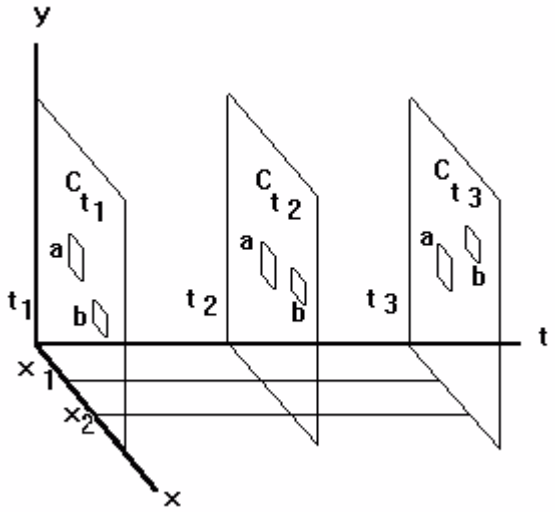


Figure 1. Example of extracting three time slices (frames) from a video source.

The meaning of the σ–operator in the above query is select, i.e. we want to select the time axis and three slices along this axis. The subscript t in σ_t indicates the selection of the time axis. In the SQL-like language a ΣQL query is expressed as:

SELECT t
CLUSTER t_1, t_2, t_3
FROM R

A new keyword "CLUSTER" is introduced, so that the parameters for the σ–operator can be listed, such as t_1, t_2, t_3. The word "CLUSTER" indicates that objects belonging to the same cluster must share some common characteristics (such as having the same time parameter value). A cluster may have a sub-structure specified in another (recursive) query. Clustering is a natural concept when dealing with spatial/temporal objects. The mechanism for clustering will be discussed further in Section 3. The result of a ΣQL query is a string that describes the relationships among the clusters. This string is called a *cluster-string*, which will also be discussed further in Section 3.

A cluster is a collection of objects sharing some common characteristics. The SELECT- CLUSTER pair of keywords in ΣQL is a natural extension of the SELECT keyword in SQL. In fact, in SQL implicitly each attribute is considered as a different axis. The selection of the attributes' axes defines the default clusters as those sharing common attribute values. As an example, the following ΣQL query is equivalent to an SQL query to select attributes' axes "sname" and "status" from the suppliers in Paris.

```
SELECT sname, status
CLUSTER *
FROM supplier
WHERE city = "Paris"
```

In the above ΣQL query, the * indicates any possible values for the dimensions sname and status. Since no clustering mechanism is indicated after the CLUSTER keyword the default clustering is assumed. Thus by adding the "CLUSTER *" clause, every SQL query can be expressed as a ΣQL query.

Each cluster can be open (with objects inside visible) or closed (with objects inside not visible). The notation is t_2^o for an open cluster and t_2^c or simply no superscript for a closed cluster. In the ΣQL language the keyword "OPEN" is used:

```
SELECT t
CLUSTER t1 , OPEN t2 , t3
FROM R
```

With the notation described above, it is quite easy to express a complex, recursive query. For example, to find the spatial relationship of two objects 'a' and 'b' from the three time slices of a source R, as illustrated in Figure 1, the ΣQL query in mathematical notation is:

$$\sigma_x (x_1 , x_2)(\sigma_t (t_1^o, t_2^o, t_3^o) R)$$

In the ΣQL language the query can be expressed as:

```
SELECT x
CLUSTER x1, x2
FROM
        SELECT t
        CLUSTER OPEN t1, OPEN t2 , OPEN t3
        FROM R
```

The query result is a cluster-string describing the spatial/temporal relationship between the objects 'a' and 'b'. How to express this spatial/temporal relationship depends upon the (spatial) data structure

used. In the next section we explain Symbolic Projection as a means to express spatial/temporal relationships.

3. A GENERAL σ–OPERATOR FOR σ–QUERIES

As mentioned above, the ΣQL query language is based upon a single operator - the σ–operator - which utilizes Symbolic Projection to express the spatial/temporal relationships in query processing. In the following, Symbolic Projection, the cutting mechanism and the general σ–operator are explained, which together constitute the theoretical underpinnings of ΣQL.

Symbolic Projection [10, 15] is a formalism where space is represented as a set of strings. Each string is a formal description of space or time, including all existing objects and their relative positions viewed along the corresponding coordinate axis of the string. This representation is qualitative because it mainly describes sequences of projected objects and their relative positions. We can use Symbolic Projection as a means for expressing the spatial/temporal relationships extracted by a spatial/temporal query.

Continuing the example illustrated by Figure 1, for time slice C_{t1} its x-projection using the Fundamental Symbolic Projection is:

$$\sigma_x (x_1 , x_2) C_{t1} = (u: C_{x1,t1} < C_{x2,t1})$$

and its y-projection is:

$$\sigma_y(y_1 , y_2) C_{t1} = (v: C_{y1,t1} < C_{y2,t1})$$

In the above example, a time slice is represented by a cluster C_{t1} containing objects with the same time attribute value t1. A cluster-string is a string composed from cluster identifiers and relational operators. The single cluster C_{t1} is considered a degenerated cluster-string. After the σ_y operator is applied, the resulting cluster $C_{y1,t1}$ contains objects with the same time and space attribute values. In the above example, the cluster-string (v: $C_{y1,t1} < C_{y2,t1}$) has the optional parentheses and projection variable Òv:Ó to emphasize the direction of projection.

The query $\sigma_t(t_1 , t_2 , t_3) R$ yields the following cluster-string α:

$$\alpha = (t: C_{t1} < C_{t2} < C_{t3})$$

When another operator is applied, it is applied to the clusters in a cluster-string. Thus the query $\sigma_x (x_1, x_2) \sigma_t(t_1^o, t_2^o, t_3^o)R$ yields the following cluster-string β:

$$\beta = (t: (u: C_{x1,t1} < C_{x2,t1}) < (u: C_{x1,t2} < C_{x2,t2}) < (u: C_{x1,t3} < C_{x2,t3}))$$

The above cluster-string β needs to be transformed so that the relationships among the objects become directly visible. This calls for the use of a *materialization function* MAT to map clusters to objects. Since $C_{x1,t1} = C_{x1,t2} = C_{x1,t3} = \{a\}$ and $C_{x2,t1} = C_{x2,t2} = C_{x2,t3} = \{b\}$, the materialization $MAT(\beta)$ of the above cluster-string yields:

$$MAT(\beta) = (t: (u: a < b) < (u: a < b) < (u: a < b))$$

Returning now to the ΣQL query that is equivalent to an SQL query to select attributes (i.e., axes) "sname" and "status" from the suppliers in Paris.

SELECT sname, status
CLUSTER *
FROM supplier
WHERE city = "Paris"

The result of the above query is a cluster-string α that describes the relationships among the clusters. Since each cluster corresponds to a unique (sname, status) pair, the query result α is:

$$\alpha = C_{\text{sname1,status1}} > C_{\text{sname2,status2}} > \ldots > C_{\text{sname-n,status-n}}$$

where > denotes an ordering relation. When this cluster string α is materialized into objects using a materialization function MAT_R, the result $MAT_R(\alpha)$ is an ordered list of (sname, status) pairs from suppliers in Paris.

The query result in general depends upon the clustering that in turn depends upon the cutting mechanism. The cutting is an important part of Symbolic Projection because a cutting determines both how to project and also the relationships among the objects or partial objects in either side of the cutting line. In most of the examples presented in this paper, the cuttings are ordered lists that are made in accordance with the Fundamental Symbolic Projection. The cutting type, κ-type, determines which particular cutting mechanism should be applied in processing a particular σ–query.

The general σ–operator is defined by the following expression where, in order to make different cutting mechanisms available, the cutting mechanism κ–type is explicitly included:

$$\sigma_{axes,\ k\text{-}type}^{\sigma\text{-type}}\ (\text{clusters})_{\boldsymbol{j}} \ \langle\text{cluster-string}\rangle = stype : \langle\text{cluster-string}\rangle$$

The general σ–operator is of the type ***s-type*** and selects an *axis* or multiple *axes*, followed by a cutting mechanism of the type κ–type on $(\text{clusters})_{\boldsymbol{j}}$ where ***j*** is a predicate that objects in the clusters must satisfy. The σ–operator operates on a cluster-string that either describes a data source (e.g. data from a specified sensor) or is the result of another σ–operator. The result of the σ–operator is another cluster-string of type *stype*. Since the result of the σ–operator is always a cluster-string, a materialization operator *MAT* is needed to transform the cluster-string into real-world objects and their relationships for presentation to the user.

4. THE ΣQL QUERY LANGUAGE

ΣQL is an extension of SQL to the case of multimedia sources. In fact, it is able to query seamlessly traditional relational tables and multimedia sources and their combination. The ΣQL query language operates on the extended *multimedia static structure* MSS which will be described in Section 6. The syntax of ΣQL can be presented in BNF notation:

<query>::= <select_type> <dimension_list>
CLUSTER <cluster_type> <cluster_values>
FROM <source>
WHERE <condition>

PRESENT <presentation_description>
<select_type> ::= SELECT | MERGE_AND | MERGE_OR
<dimension_list> ::= <dimension>, <dimension_list> | <dimension>
<dimension>::= x | y | z | t | image_object | audio_object | video_object | type | attribute | object | ..
<cluster_type ::= ε | interval_projection | ..
<cluster_values> ::= * | <cluster_list>
<cluster_list> ::= <cluster_val>, <cluster_list> | <cluster_val>
<cluster_val>::= <val> | OPEN <val>
| (<val> ALIAS <identifier>) | OPEN (<val> ALIAS <identifier>)
<val> ::= <variable_identifier> | <string_constant> | <numeric_constant>
<source>::= <query> | <source_name>
<condition>::= <string>
<presentation_description> ::= <string>
<source_name> ::= <source_identifier>

A template of an ΣQL query is given below:

SELECT *dimension_list*
CLUSTER [*cluster_type*] [OPEN] *cluster_val*$_1$, .., [OPEN] *cluster_val*$_n$
FROM *source*
WHERE *conditions*
PRESENT *presentation_description*

which can be translated as follows: "Given a source (FROM *source*) and a list of dimensions (SELECT *dimensions*), select clusters (CLUSTER) corresponding to a list of projection values or variables ([OPEN] *cluster_val*$_1$, ..) on the dimension axes using the default or a particular clustering mechanism ([*cluster_type*]). The clusters must satisfy a set of conditions (WHERE *conditions*) on the existing projection variables and/or on cluster contents if these are open ([OPEN]). The final result is presented according to a set of presentation specifications (PRESENT *presentation_description*)."

Each σ–query can be expressed as an ΣQL query. For example, the σ–query $\sigma_{s,\kappa}(s_1, s_2^o, s_3, .., s_n)_\phi$ R can be translated as follows:

SELECT s
CLUSTER κ s_1, OPEN s_2, s_3, .., s_n
FROM R
WHERE ϕ

A σ–query can be processed according to the following procedure.

Procedure σ–query_Processor($\sigma_{s,\kappa}$ (s_1, s_2, s_3, .., s_n) R)
Input: (1) A cluster-string representing R, and (2) a σ–query.
Output: The retrieval results.
Step 1: Apply cutting mechanism κ to R to find all of its sub-clusters according to the clustering (s_1, s_2, s_3, .., s_n).
Step 2: Apply $\sigma_{s,\kappa}$ to all the clusters C_{s1}, C_{s2}, ... , C_{sn} and return a cluster-string (a relational expression) on them.
Step 3: For each sub-cluster C_{si} , if s_i is closed it is treated as a single object and $\sigma(C_{si}) = C_{si}$. If s_i is open it is treated as a set of objects and σ can be applied to the constituent objects that may be sub-sub-clusters.

The above algorithm is recursive, i.e., each R may itself be the form $\sigma_{w,\kappa}$ (w_1, w_2, w_3, ... , w_n) R' and can be evaluated recursively.

5. AN ILLUSTRATION TO MULTISENSOR DATA FUSION

In this section, ΣQL will be illustrated with a query that uses data from two different sensors, i.e. a laser radar and a video. The data from these two sensors are heterogeneous with respect to each other. An example of a laser radar image is given in Figure 2. This image shows a parking with a fairly large number of cars, which look like rectangles when viewed from the top. The only moving car can be seen in the lower right part of the image with a north-south orientation while all other cars in the image have an east-west orientation. The moving car and five of its parked neighbours can also be seen in Figure 4. This image is a somewhat enlarged version of a part of the image in Figure 2 and viewed in an elevation angle that shows the three dimensions of the image. The holes at the vehicles in this figure are due to the fact that no information from the sides of the vehicles has been registered.

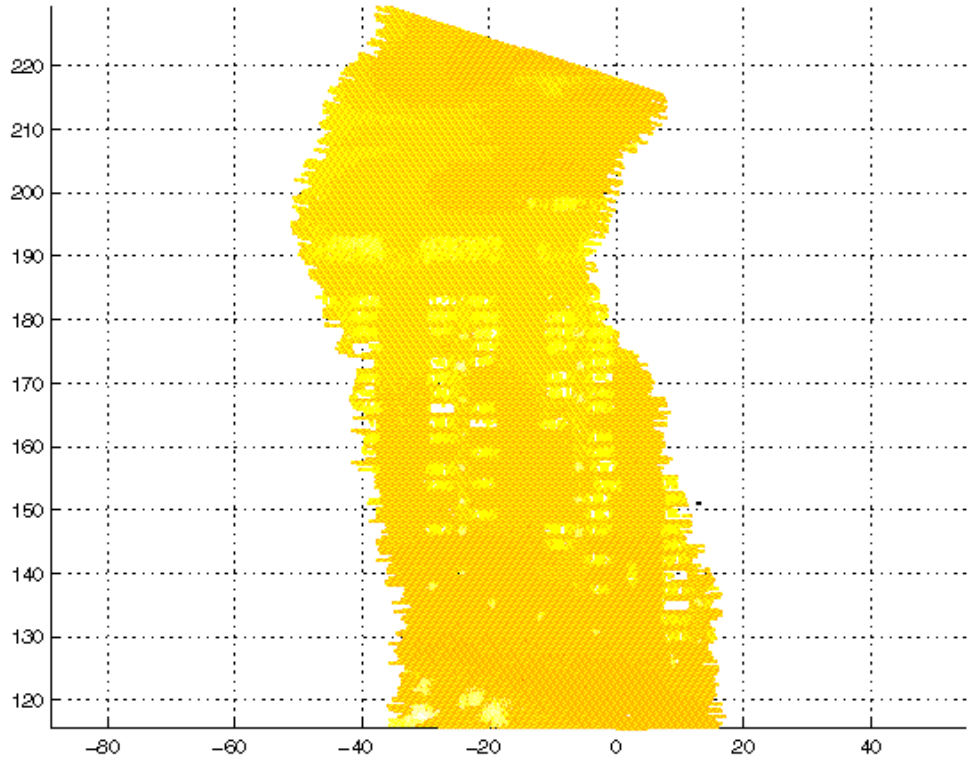


Figure 2. An example of a laser radar image taken across a parking with a moving car in the lower right part of the image in north-south orientation.

Laser radar images are characterized by being three dimensional and in having geometric properties, that is, each image point is represented with x-, y- and z-coordinate values. The particular laser radar used here, is manufactured by SAAB Dynamics in Sweden, is helicopter born and generates image elements from a laser beam that is split into short pulses by a rotating mirror. The laser pulses are transmitted to the ground, in a scanning movement, and when reflected back to the platform a receiver collects the returning pulses which are stored and analyzed. The result of the analyze are points represented with their three coordinates. There is also a time indication for each point. The resolution of a laser radar image is about 0.3 m.

Figure 3. Two video frames showing a white car at the entrance in the middle of the image (a) and the white car between some of the parked cars (b).

The video, see Figure 3 (a) and (b), is of ordinary type and carried by the same platform. The two sensors are observing the same area at the same time. This means, that most cars in the parking can be seen in the images from both the sensors. The moving car in Figure 3 (a) is outside the parking and immediately to the left of the entrance and is white. In Figure 3 (b) it has entered the parking and reached the first of the parked cars. It is quite simple to generate the various projection strings from both types of sensor images. Figure 5 shows two symbolic images corresponding to the two video images in Figure 3. Almost identical projection strings can be generated from the laser radar image.

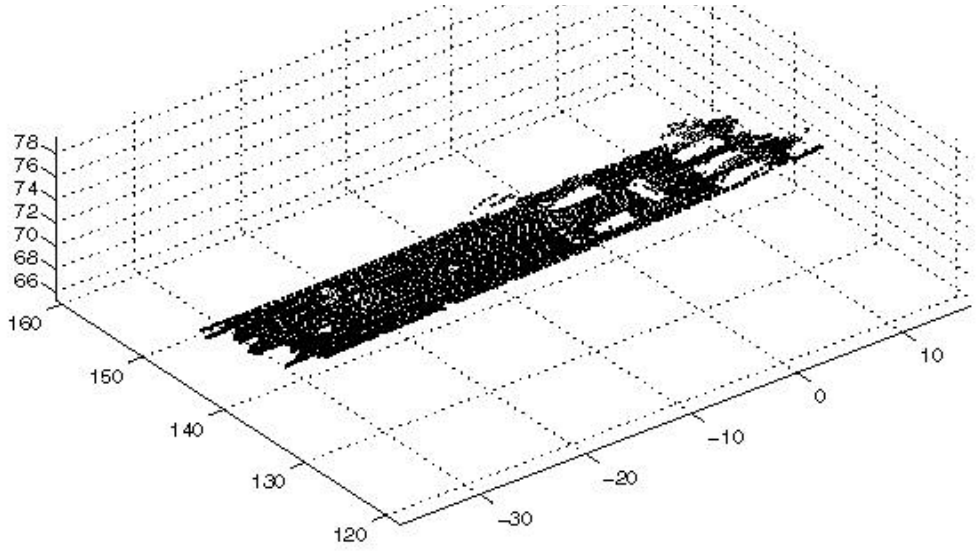


Figure 4. A narrow slice of the laser radar image in Figure 2 shows the moving car and some of its parked neighbours.

Basically, the query that is of concern here can be formulated as follows. Assume that we are interested in determining moving objects along a flight. This can theoretically be done by analyzing the video alone, but that requires hundreds and probably even more sequential video frames to be analyzed. This will take both a very long time and really large computational resources, which may not always be available. Furthermore, this problem cannot, at this time, be solved in real time. By using images from a laser radar, on the other hand, it is possible to recognize any type of vehicles in real time with respect to their time and position. This has been shown by Jungert et al. in [12, 13].

However, it cannot be determined from this sensor whether the vehicles are moving. The solution to this problem is to analyze the laser radar image to first find occurring vehicles and determine their position in time and from this information in a second step identify a very *limited* set of video frames that includes the vehicles found in the laser radar image. From the now limited set of video frames it is possible to determine which of the vehicles that are in motion. Finally, in a fusion process, it can be determined which of the vehicles that are moving. This will be illustrated by the query below where the query first is split into two subqueries that correspond to queries concerned with data from just one of the sensors. In the final step, it will also be demonstrated how the information from the sensors is fused in order to answer the query.

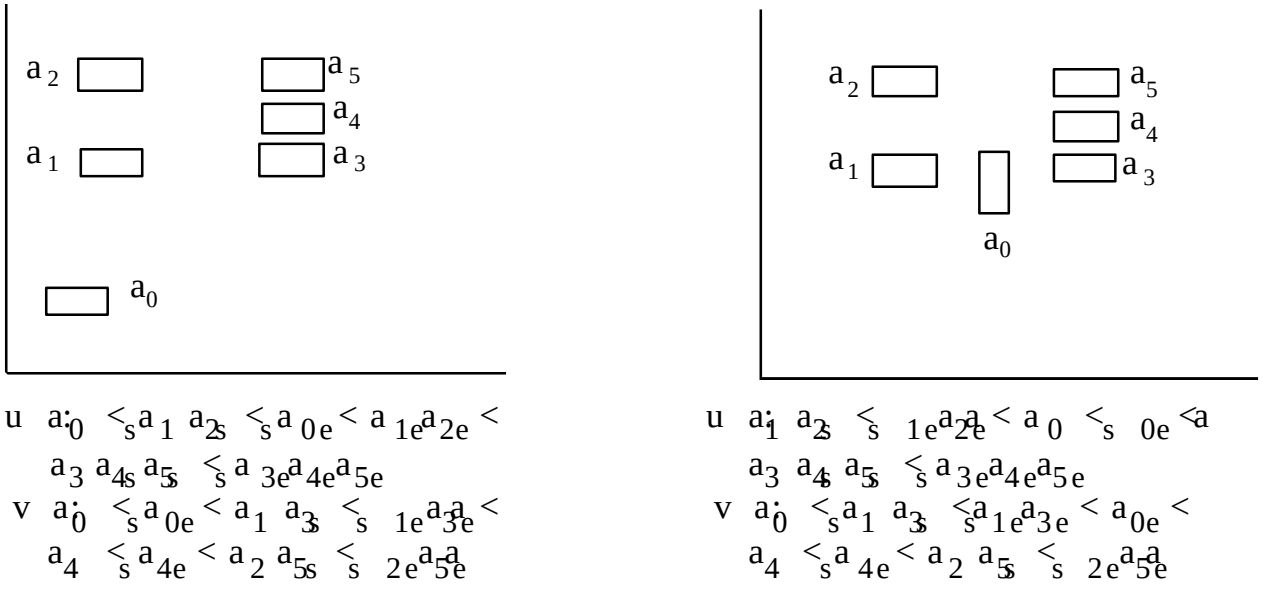


Figure 5. Two symbolic images showing the situation of the two video frames in Figure 3 with the moving car and its close neighbours and the corresponding interval projection strings [8].

An important problem that is not addressed in this work, but will be subject to future research, is the handling of uncertain sensor information. Clearly, this is a very important problem that cannot be excluded when designing a query language for sensor data fusion, where in particular all input data come from heterogeneous data sources. However, we have found it necessary to address the basic query techniques, the syntax of the query language and the basic abstract spatial/temporal structures for reasoning first. In this perspective, the query is first represented as σ-sequences and then translated into ΣQL-syntax.

subquery1: Are there any moving objects in the video sequence in the time interval t_1 through t_2 ?

$$Q_1 = \sigma_{motion}(moving)\sigma_{type}(vehicle)\ \sigma_{xy,interval\_cutting}(*)$$
$$\sigma_t(T^o)_{T \bmod 10 = 0 \text{ and } T>t1 \text{ and } T<t2}$$
$$\sigma_{media\_sources}\ (video^o)media\_sources$$

subquery2: Are there any vehicles in the laser radar image in the time interval t_1 through t_2 ?

$$Q_2 = \sigma_{type}\ (vehicle)\ \sigma_{xyz,interval\_cutting(*)}\sigma_t(T^o)_{\ T>t1 \text{ and } T<t2}$$
$$\sigma_{media\_sources}(laser\_radar^o)\ media\_sources$$

The subquery Q_1 first selects the video source and then the video frames which all are opened. However, the selection of video frames also includes some conditions with respect to which frames

to accept and in which time interval. In this case we have chosen to select each tenth video frame within the interval [t1,t2]. In the next selection the σ_{xy}-operator is applied to the video frames using the interval cutting mechanism [8, 9]. This operator generates the (u,v)-strings from which the object types are determined by the σ_{type}-operator. That is, in this particular case, the vehicles, in the selected frames. Eventually the vehicles in motion are determined by the application of the motion operator. The motion string (m) is generated from the time projection string (t) where the single video frames are opened with respect to the x- and y-dimensions, i.e.:

$$
\begin{aligned}
&t: (u: a_{0s} < a_{1s}\ a_{2s} < a_{0e} < a_{1e}\ a_{2e} < a_{3s}\ a_{4s}\ a_{5s} < a_{3e}\ a_{4e}\ a_{5e},\\
&\quad v: a_{0s} < a_{0e} < a_{1s}\ a_{3s} < a_{1e}\ a_{3e} < a_{4s} < a_{4e} < a_{2s}\ a_{5s} < a_{2e}\ a_{5e}) <\\
&\quad (u: ..., v: ...) <\\
&\quad (u: a_{1s}\ a_{2s} < a_{1e}\ a_{2e} < a_{0s} < a_{0e} < a_{3s}\ a_{4s}\ a_{5s} < a_{3e}\ a_{4e}\ a_{5e},\\
&\quad\ v: a_{0s} < a_{1s}\ a_{3s} < a_{1e}\ a_{3e} < a_{0e} < a_{4s} < a_{4e} < a_{2s}\ a_{5s} < a_{2e}\ a_{5e}) <\\
&\quad (u: ..., v: ...) <\\
&\quad ...
\end{aligned}
$$

From this string the motion string is generated by applying the σ-operator which generates a string similar to an implicit merge_or-operation, i.e.:

$$m: t: (u: a_{0s} < a_{0e} < a'_{0s} < a'_{0e,}\ v: a_{0s} < a_{0e} < a'_{0s} < a'_{0e})$$

The subquery Q_2 returns first the (u,v)-strings for the time interval [t1,t2]. This is sufficient, since that gives the relative position of the vehicles (the z-information is normally unnecessary for this purpose and will only be used for object type recognition). An intermediate result of the subquery will thus look like:

$$
\begin{aligned}
&u: a_{1s}\ a_{2s} < a_{1e}\ a_{2e} < a_{0s} < a_{0e} < a_{3s}\ a_{4s}\ a_{5s} < a_{3e}\ a_{4e}\ a_{5e}\\
&v: a_{1s}\ a_{3s} < a_{0s} < a_{1e}\ a_{3e} < a_{4s} < a_{4e} < a_{2s}\ a_{5s} < a_{0e} < a_{2e}\ a_{5e}
\end{aligned}
$$

The laser scanner determines each point in a sequential order so that an object that is in the lower left corner of the image is first registered by the sensor. Therefore it is possible to implicitly determine the t-string, if needed.

$$t: (u: a_{1)} <_{...} < (u: a_{i})_{...}$$

This, however, requires a further application of σ_t but that has not been applied here. However, it can, nevertheless, be motivated because it may support the outcome of the data fusion process.

In the final step of this subquery existing vehicles are determined by applying the σ_{type}-operator.

The two subqueries can now be fused with respect to equality. For this purpose another operator that can perform this is needed. However, this fusion operator is different from the σ-operators that have been used, so far, since its input will be coming from multiple data sources, that must be of equal type. For this reason, a fusion operator, called ϕ is defined. This operator performs the data fusion operation, here called *merge_and*, with respect to the three dimensions x, y and t; in other words, it fuses the vehicle information with respect to equality of type and position in time. The object type in question is in this case already determined. All object types are consequently equal in both subqueries. The final query with the fusion operator thus becomes:

$$Q_3 = \phi_{xyt}^{\text{merge-and}}(*)(Q_1,Q_2)$$

This means that a fusion operation is applied such that only those objects selected from the two subqueries and which can be *associated* to each other will remain in the output string, which here is called mo (motion objects). This gives us the following result:

mo: a_0

The complete query now looks like:

$\phi_{xyt}^{\text{merge-and}}(*)$

$(\sigma_{\text{motion}}(\text{moving})\sigma_{\text{type}}(\text{vehicle})\ \sigma_{\text{xy,interval\_cutting}}(*)$
$\sigma_t(T^o)_{\text{T mod 10 = 0 and T>t1 and T <t2}}$
$\sigma_{\text{media\_sources}}\ (\text{video}^o)\text{media\_sources},$

$\sigma_{\text{type}}\ (\text{vehicle})\ \sigma_{\text{xyz,interval\_cutting}(*)}\sigma_t(T^o)_{\ \text{T>t1 and T<t2}}$
$\sigma_{\text{media\_sources}}(\text{laser\_radar}^{o)}\ \text{media\_sources})$

The important problem here is, as always in data fusion, the association problem. In other words, the query must determine whether a certain object found in one of the two subqueries is the same as any of the vehicles found in the other subquery. This problem is generally very difficult and is discussed more deeply in [23].

Translating the σ-query into ΣQL-syntax is now a fairly simple task and the result from this translation becomes:

```
MERGE-AND x,y,t
CLUSTER *,*,[t1,t2]
FROM (SELECT type
        CLUSTER vehicle
        FROM SELECT x,y,z
                CLUSTER interval, *
                FROM SELECT t
                        CLUSTER OPEN (* ALIAS T)
                        FROM SELECT media_sources
                                CLUSTER OPEN laser_radar
                                FROM media_sources
                        WHERE T > t1 AND T < t2,
        SELECT motion
        CLUSTER moving
        FROM SELECT type
                CLUSTER vehicle
                FROM SELECT x,y
                        CLUSTER interval *
                        FROM SELECT t
                                CLUSTER OPEN (* ALIAS T)
                                FROM SELECT media_sources
                                        CLUSTER OPEN video
                                        FROM media_sources
                                WHERE T mod 10 = 0 AND T>t1 AND T<t2)
```

6. REPRESENTING A MULTIMEDIA SOURCE

In the previous sections we have described a data source as a simple projection string. However, in general, in order to describe data sources we need a more complex data structure. In this section we describe an extension of the MSS model proposed in [6] for the description of multimedia data.
A multimedia source description is composed of a hierarchy of *entities*. Each entity has the following format:

<name, type, list of legal descriptions>

where:
1. the name is the entity identifier
2. the type is the entity type
3. each description is a triple $((d_1..d_m)$: $\{e_1, e_2, .., e_n\}$: rel_expr) with $m \geq 1$ and $n \geq 0$ containing
   - a list of dimensions d_i according to which the entity is being clustered
   - a set of component entity identifiers resulting from the clustering
   - a relational expression where relations (depending on the dimensions) are used to relate the component entities

   A description is legal if n = 0 or a clustering mechanism able to derive the description from the source is available. In the case n=0, the entity is an atom with respect to the description dimension and the relational expression reduces to a simple value.

Depending on the chosen description type (and consequently on the associated clustering mechanism) a source can be seen as a temporal sequence of entities, or a spatial disposition of entities, or as a set of attribute-value pairs, etc.
In general we make use of the 4 type of descriptions:

1. Temporal descriptions: the dimension is given by the *time* axis and the relational expression makes use of the temporal relation *before than* denoted as "<". This is a legal description for video sources: the clustering is given by the extraction of the single frames making the video. As an example, a video clip segment of three frames R, S, T may be described by the triple (*time*: {R, S, T}: (t: R < S < T)). On the other hand, the triple (*time*: {}: t) is a temporal description of a frame where the empty set indicates that the frame is atomic with respect to time and then cannot be decomposed, and t indicates its projection value on the time axis.

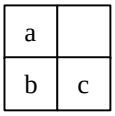


Figure 6. A simple symbolic picture

2. x-coord spatial description: the dimension is given by the *x* axis of a Cartesian plane and the relational expression is written as a 1-D string [10]. There are many clustering available to produce a 1-D string but the Fundamental Symbolic Projection is taken as the default clustering for this description.
   As an example, the description (*x*: {a, b, c}: (u: a = b < c)) represents the projection on the x axis of the symbolic picture in Figure 6.

3. y-coord spatial description: the dimension is given by the *y* axis of a Cartesian plane and the relational expression is written as a 1-D string [10]. Again the Fundamental Symbolic Projection si considered the default clustering mechanism.
   As an example, the triple ((*x*, *y*): {a, b, c}: ((u : a = b < c), (v: b=c<a))) is a spatial description along the two axis x and y of the symbolic picture in Figure 6. The use of both the x and y dimensions produces a 2-D string as relational expression.
4. object description: the dimension is given by a *set-of-attributes* (or *object*) axis and the relational expression reduces to a set of facts (AttributeName : Value) on the component entities. An entity is always considered atomic with respect to the dimension *object.*
   As an example, the following triple (*object*,: {}: {(clip_id : X), (length :Y)}) is the description of a video clip whose clip_id and length have values X and Y, respectively. Note that the descriptions (*time*: {}: 3) and (*object*: {}: {(time : 3)}) are equivalent. Moreover, since there is no clustering an object description is always legal.

As an example, let us consider a video clip segment showing two trees and three walking persons. The video is being clustered according to the time dimension in three consecutive frames as shown in Figure 7. The video represents Cathy (c), Bill (b) and Dan (d) moving east, and two trees (a). It can be noted that Dan moves slowly and Cathy goes out of the scene in the third frame. The following entities are an MSS representation of the video:

[R, video, (τ: {R_1, R_2, R_3}: (t: $R_1 < R_2 < R_3$))]
[R_1, frame, (τ: {}: t_1),
((*x*, *y*): {a, b, c, d}: ((u: a < b < c=d < a), (v: d < b < a= a < c)))]
[R_2, frame, (τ: {}: t_2),
((*x*, *y*): {a, b, c, d}: ((u: a < < b=d < a=c), (v: d < b < a= a < c)))]
[R_3, frame, (τ: {}: t_3),
((*x*, *y*): {a, b, d}: ((u: a < < < a=b=d), (v: d < b < a= a <)))]
[a, plant, (*object*: {}: {(name: tree)})]
[b, person, (*object*: {}: {(name: Bill)})]
[c, person, (*object*: {}: {(name: Cathy)})]
[d, person, (*object*: {}: {(name: Dan)})]

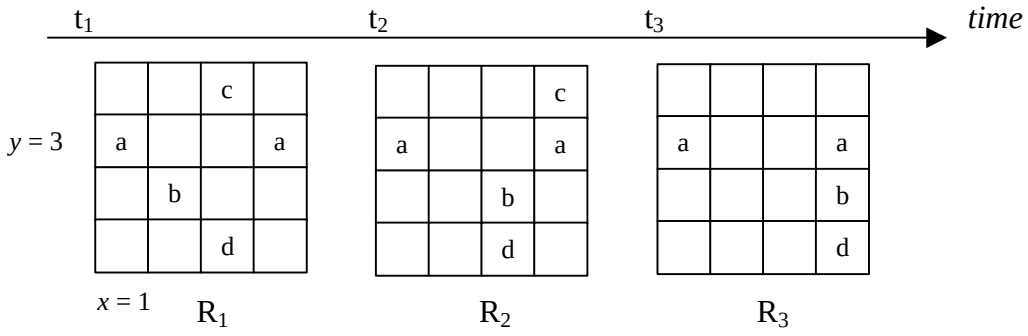


Figure 7. A video clip segment of three frames R_1, R_2, R_3

In the following we provide some examples of ΣQL queries acting on the above MSS.

Example 1. In the following query we want to retrieve all the frames showing Cathy. Moreover we want to know the temporal and spatial relation among them.

```
SELECT x
CLUSTER *
FROM
        SELECT  t
        CLUSTER  OPEN (* ALIAS ANYCLUSTER)
        FROM R
        WHERE ANYCLUSTER contains c
```

This query retrieves the two video frames occurring at times t_1 and t_2 and the temporal relations among the two frames. Since t is preceded by the keyword OPEN it has been possible to set a condition in the WHERE clause on the content of the frame at a generic time aliased as ANYCLUSTER.
The fact that no cutting mechanism is provided indicates that the default cutting mechanism is used to individuate the three clusters. The mathematical notation for the query is:

$$\sigma_x(*)\sigma_t(T^o)_{\text{cluster(T) contains c}}\ R.$$

The query returns the entity
[result, video,
(τ: [R_1, frame, (t: {}: t_1), ((*x*, *y*): {a, b, c, d}: ((u: a < b < c=d < a), (v: d < b < a= a < c)))] <
[R_2, frame, (t: {}: t_2), ((*x*, *y*): {a, b, c, d}: ((u: a << b=d <a=c), (v: d < b < a= a < c)))] <)]
in its formal representation.

Example 2:

Let us consider the following query to retrieve all the video clip parts showing the interaction in the time along the x axis of Cathy, Bill and any other video object interacting with them.

```
SELECT x
CLUSTER X
FROM
        SELECT t
        CLUSTER OPEN (* ALIAS ANY)
        FROM R
        WHERE ANY contains c  and ANY contains b
WHERE  X = c.x  or X = b.x
```

Given the video input sequence shown in Figure 7 the nested sub-query will return the entity
[result1, video, (τ:
[R_1, frame, (τ: {}: t_1),
((*x*, *y*): {a, b, c, d}: ((u: a < b < c=d < a), (v: d < b < a= a < c)))] <
[R_2, frame, (τ: {}: t_2),
((*x*, *y*): {a, b, c, d}: ((u: a < < b=d < a=c), (v: d < b < a= a < c)))])
]

since only R_1 and R_2 contain both Cathy and Bill.

The outer query will operate on the entity result1 to produce the output

[result2, video, (τ:
[R_1, frame, (τ: {}: t_1),

(*x*: {b, c, d}: ((u: < b < c=d <)))] <
[R_2, frame, (τ: {}: t_2),
(*x*: {a, b, c, d}: ((u: < < b=d < a=c)))])
]

where only the clusters built along the x axis containing b and c are left.

Example 3:

Suppose now we are interested in the video clip part involving only the interactions between Bill and Cathy and no other object. By querying the entity result2 from example 2 then we can build the following query:

SELECT object
CLUSTER c, b
FROM result2

The result will then be
[result3, video, (τ:
[R_1, frame, (τ: {}: t_1),
(*x*: {b, c}: ((u: < b < c <)))] <
[R_2, frame, (τ: {}: t_2),
(*x*: {b, c}: ((u: < < b < c)))])
]

The final query can then be read as "retrieve only the interactions between Cathy and Bill along with the x-dimension during the execution of the whole video R".
By looking at the disposition of the spatial-temporal relations < in the entity result3, it is easy to note how Bill and Cathy are moving at the same speed along the x-axis in the first two frames of the video clip.

7. ΣQL QUERY PROCESSING

Let us now see the phases needed to process a ΣQL query.
1. lexical and syntactic analysis
2. semantic correctness or *transformational analysis* where the following actions are taken:
   a. control the compatibility between the dimensions and the sources in each sub-query
   b. check if the overall query is consistent, i.e., there exist representations for the intermediate results that make the query execution feasible. If so, keep track of the representations. (Intermediate result type inference)
   c. For each intermediate result use a feasible representation (this can be done with the user help)
   d. if the main source is not structured then
      build the MSS schema
      else
      check if there exists a query-engine that allows to query the structured source (MMDB)
3. query optimization
4. query execution
   a. if the main source is not structured then

- populate the MSS structure according to the schema defined during the semantic analysis by extracting the appropriate information from the source. (The semantic validation implies that the algorithms to extract the information from the source are available)
- execute the query against the MSS
- present the results

else

- send the query to the query-engine and wait for results
- present the results

To improve the efficiency, the query execution can be pipelined.

7.1. Transformational analysis - The environment representation

In order to perform the transformational analysis of a ΣQL query, knowledge about the Multimedia Query Environment (MQE for short) is needed. An MQE is composed by a set of data structures and algorithms for the analysis of multimedia data, the extraction of features and their conversion in different representations.
In particular the processor must know

1. the σ dimensions that are supported
2. the sources that can be handled
3. the compatibility between clusterings on given dimensions and sources, i.e., if there exist algorithms to cluster a given source along a given dimension
4. the representations that the MQE is able to build from multimedia data
5. the compatibility between different representations, i.e., given two representations, if there exist operators defined on them.
6. which are the representations that can be build from a source when a given clustering along a dimension on it has been given.

Let

D be the set of dimensions supported by the environment
S be the set of sources that can be handled by the environment
C be the set of clusterings that can be applied along any given dimension
R be the set of representations that the environment is able to build from sources
O be the set of operators op: $R \times R \rightarrow R$ able to combine two representations

The environment can be coded through three 3D tables: the dimension-clustering-source or DCS table, the dimension-clustering-representation or DCR table, and the operator compatibility representation-representation-operator or RRO table.
The DCS table is indexed by the triple (d, c, s) where d is a dimension, c is a clustering and s is a source. Each table entry contains a (possibly empty) set of representations for s when clustered under clustering c along the dimension d.
For this table, the entry:

$$DCS(d, c, s) = set\_of\_rep$$

if and only if, for each $r' \in set\_of\_rep$ there exists an algorithm in the environment translating the data source s under clustering c along the dimension d into the representation r'.

The DCR table is indexed by the triple (d, c, r) where d is a dimension, c is a clustering and r is a representation. Each table entry contains a (possibly empty) set of representations derived from r when cut under clustering c along the dimension d.
For this table, the entry:

$$DCR(d, c, r) = set\_of\_rep$$

if and only if, for each $r' \in set\_of\_rep$ there exists an algorithm in the environment producing the representation r' by cutting the representation r under clustering c along the dimension d.

The compatibility representation table RRO is indexed by the triple (r, r', op) where r and r' are representations, and op is an operator on representations. Each table entry contains a (possibly empty) set of representations obtained by applying op on r and r'. For this table, the entry:

$$RRO(r, r', op) = set\_of\_rep$$

if and only if, for each $r'' \in set\_of\_rep$ there exists an implementation, in the environment, of the operator op producing the representation r'' when applied to r and r'.

7.2. Transformational analysis - The query representation

Once a Query Multimedia Environment Ψ has been defined it becomes possible to check queries for consistency under Ψ. In order to describe how the checks are realized we need to describe an alternative representation of a ∑QL query: we use a tree structure named *consistency check tree* with three kinds of nodes (SELECT-CLUSTER, SOURCE, and MERGE nodes) and two kinds of edges (FROM and ARGUMENT edges) as shown in Figure 8.

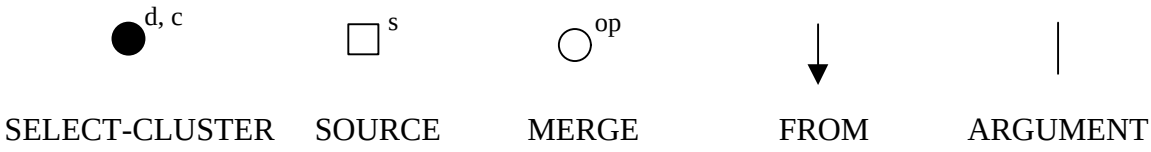


Figure 8. Visual tokens to represent a σ-query

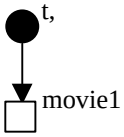
The node SELECT-CLUSTER is labeled with the dimension d and the cluster c as appearing in the SELECT and CLUSTER clauses. The node SOURCE and MERGE are labeled by the source s and the operator op, respectively, to which they refer. Since we will only make use of binary operators, the node MERGE can only have two children.
The tree structure can be easily understood by the following examples.

Example 4:

Given the query

```
SELECT t
CLUSTER  t1, t2
FROM movie1
```

its consistency check tree is given by



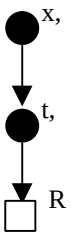
Example 5:

Given the query

```
SELECT x
CLUSTER X
```

```
FROM
        SELECT t
        CLUSTER OPEN (* ALIAS ANY)
        FROM R
        WHERE ANY contains c  and ANY contains b
                WHERE  X = c.x  or X = b.x
```

its consistency check tree is given by



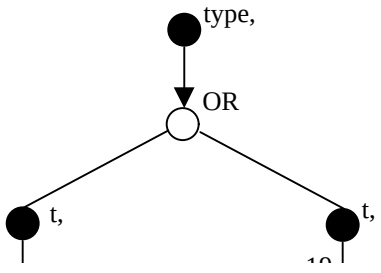
Example 6:

Given the query :

```
SELECT type
CLUSTER c1 , c2
FROM
        MERGE_OR t
        CLUSTER t1 , t2 , t3
        FROM
                (SELECT t
                CLUSTER t1 , t2 , t3
                FROM
                        SELECT audio-object
                        CLUSTER *
                        FROM audio_source,

                SELECT t
                CLUSTER t1 , t2 , t3
                FROM
                        SELECT video-object
                        CLUSTER *
                        FROM video_source)
```

its consistency check tree is given by :



Given a σ–query it is always possible to build a consistency check tree by using the traditional language technology. In the following section we will give an algorithm that, visiting a consistency check tree, will be able to decide if the corresponding query is consistent in the given environment and how the involved sources have to be clustered and represented in order to be searched efficiently.

Once a query has been declared consistent under a certain environment it can be executed in that environment. Before being executed, however, each intermediate result must be assigned one and only one representation. In fact the consistency check allows having more than one feasible representation for a temporary result. The final assignment can be done either randomly or letting the user select the type of representation at each step. How to make intuitive the user selection of a representation will be taken care of from the interactive visual query editor.

8. TRANSFORMATIONAL ANALYSIS FOR ΣQL QUERY PROCESSING

In the following we provide the algorithm *Consistency Check* that annotates each internal node of a consistency check tree with a set of *legal* representations if the original query is consistent or an error message otherwise. A representation is legal for a node if it can be derived either by clustering along a dimension a source or a representation, or by applying an operation on the representations of the child nodes.
The algorithm is made of two parts. The first part is based on a bottom-up visit of the tree and at each step it accesses one of the three environment tables DCS, DCR or RRO to check whether one or more representations for the current node can be synthesized. If the query is consistent, the algorithm will annotate all the internal nodes and the root with a non empty set of legal representations. It will emit an error message otherwise.
For each assigned representation the algorithm will also keep track of the representations from where it has been derived. This information will be used by the second part of the algorithm to assign a single legal representation to each internal node of the tree.

Algorithm *Consistency Check.*
Input: a consistency check tree for a σ–query Q and an environment Ψ with tables DCS, DCR and RRO.
Output: an annotated consistency check tree and a representation assignment for each internal tree node if the query is consistent or a message error otherwise.

Method:

1. Apply a bottom-up visit to the tree and for each node visited behave in the following way:

if the node is a SOURCE node **then** do nothing

if the node is a SELECT-CLUSTER node *SC* labeled (d, c) connected through a FROM edge to a SOURCE node labeled s
then
Annotate *SC* with the set of representations from the entry DCS(d, c, s)
if DCS(d, c, s) = ∅ **then**
emit "Source s can't be clustered through c under dimension d in the current environment";
exit.

if the node is a MERGE node *OP* labeled *op* with two child nodes labeled with the sets of representations set_1 and set_2, respectively
then
Annotate *OP* with the set of representations set_3 such that $r' \in set_3$ if and only if there exists at least a pair of representations $r_1 \in set_1$ and $r_2 \in set_2$ and $r' \in RRO(r_1, r_2, op)$.
Create a connection from each $r' \in set_3$ to each $r_1 \in set_1$ and $r_2 \in set_2$ such that $r' \in RRO(r_1, r_2, op)$
if $set_3 = \emptyset$ **then**
emit "The representations in set_1 and set_2 are not compatible under the operator *op* in the current environment";
exit.

if the node is a SELECT-CLUSTER node *SC* labeled (d, c) connected through a FROM edge to a node labeled with a set of representations set_1
then
Annotate *SC* with the set of representations set_2 such that $r' \in set_2$ if and only if there exists at least a representation $r \in set_1$ and $r' \in DCR(d, c, r)$.
Create a connection from each $r' \in set_2$ to each $r \in set_1$ such that $r' \in DCR(d, c, r)$.
if $set_2 = \emptyset$ **then**
emit "The representations in set_1 can't be opened through c under dimension d in the current environment";
exit.

2. Apply a top-down visit of the tree following the representation connections built in step 1 according to the following instructions:

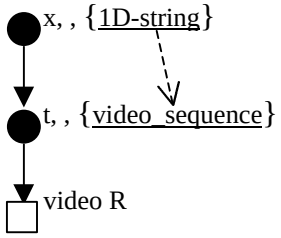
Select and **mark** a representation *head_r* from the set of representations annotating the tree root.
Add *head_r* to the empty set Marked_r
While Marked_r is not empty **do**
Extract a representation from Marked_r
Follow all the connections leaving that representation.
If it is connected to only one representation in the same set **then** select and mark that representation
If it is connected to more than one representation in the same set **then** select and mark only one of them.
Add all the marked representations to Marked_r
endwhile

end Algorithm

In step 2 the selections of particular representations can be done either randomly or under suggestions of a user. These suggestions can be obtained allowing the user to visually define his representation preferences.
The annotated consistency check tree resulting from the execution of the algorithm allows to build a query customized MSS for each source referred in the original query. This will allow an effective execution of the query against the coded MSSs.

Example 7:

Let us consider the query of Example 5 with its consistency check tree. The algorithm produces the following annotated tree assumed that DCS(t, default_cutting, video) = {video_sequence} and DCR(x, default_cutting, video_sequence)= {1D-string}. Step 2 of the algorithm in this case just needs to mark the single representations in each set.



The resulting MSS describing the video R will then need to store a video_sequence projected in the time dimension such that it can be opened in 1D_string representations along the x dimension.

9. DISCUSSION AND CONCLUSION

As explained in previous sections, ΣQL can express both spatial and temporal constraints individually using the SELECT/CLUSTER construct and nested subqueries. Its limitation seems to be that constraints simultaneously involving space and time cannot be easily expressed, unless embedded in the WHERE clause. Although such constraints may be rare in practical applications, further investigation is needed in order to deal with such complex constraints.

A visual language version of this new ΣQL language may be suitable for the Hypermapped Virtual World (HVW) information model [6], which is a combination of hypermap with virtual reality, so that each hyperlink can lead to a virtual world. Hypermaps can be used advantageously as a metaphor for the representation of all the multimedia hyperbase elements. For example, in GeoAnchor [2] a map can be built dynamically as a view of the multimedia hyperbase. Each displayed geometry is an anchor to either a geographic node or to a related node. Hence, the map on the screen acts both as an index to the nodes and as a view to the multimedia hyperbase. As another example, in a Virtual Classroom a hypermap can also be used as a metaphor to link the most frequently accessed items such as reading rooms, book shelves, etc. to present different views to the end user. This combined metaphor of Hypermapped Virtual Classroom (which is a combination of the VR information space and the logical information hyperspace [7]) may lead to efficient access of multimedia information in a distance learning environment. The σ–query may serve as the basis of a visual query language for the Hypermapped Virtual World.

The visual ΣQL can be applied directly to a scene such as Figure 1, by visually selecting and clustering (decomposing) the 2D (or 3D, 4D, etc.) space. Applications to distance learning, remote sensing, etc. can be explored for the Visual ΣQL. Another important application of visual ΣQL, as

pointed out in Section 6, is to facilitate user-system interaction in selecting feasible representations in the transformational analysis of a σ–query, in other words, in computer-assisted visual reasoning.

In addition to building the ΣQL prototype, we need to further evaluate its efficiency by testing the prototype against multimedia static schemas (MSSs) of various degree of complexity. This can be accomplished by carefully selecting the application domains, taking sample multimedia data and constructing automatically many similar MSSs with different degree of complexity, and running the ΣQL prototype against different MSSs to compare the results.

We will also evaluate the effectiveness of the proposed spatial/temporal query language by studying several different applications. Formative evaluation will be carried out to improve the design, and summative evaluation will be carried out to conclude the experimental studies.

REFERENCES

[1] Ahanger, G., Benson, D. and Little, T.D., "Video Query Formulation", Proceedings of Storage and Retrieval for Images and Video Databases II, San Jose, February 1995, SPIE, pp. 280-291.

[2] Caporal, A J., Viemont, A Y. A Y. , "Maps as a Metaphor in a Geographical Hypermedia System", Journal of Visual Languages and Computing, vol. 8, No 1, February 1997, pp 3-25.

[3] Chan, E. P. F. and Zhu, R., "QL/G - A query language for geometric data bases", Proceedings of the 1st International Conf. on GIS in Urban Regional and Environment Planning, Samos, Greece, April 1996, pp 271-286.

[4] Chang, S. F., Chen, W., Meng, H. J., Sundaram, H., and Zhong, D., "VideoQ: An automated Content Based Video Search System using Visual Cues", Proceedings of the Fifth ACM International Multimedia Conference, November 1997.

[5] Chang, S.-K. and Jungert, E., "Human and system directed fusion of multimedia and multimodal information using the s-tree data model", Proceedings of the 2nd International conference on Visual information systems, San Diego, CA, December 15-17, 1997.

[6] Chang, S. K., "Content-Based Access to Multimedia Information", Proceedings of Aizu International Student Forum-Contest on Multimedia, (N. Mirenkov and A. Vazhenin. eds.), The University of Aizu, Aizu, Japan, Jul 20-24, 1998, pp. 2-41. (The paper is available at www.cs.pitt.edu/Óchang/365/cbam7.html)

[7] Chang, S. K. and Costabile, M. F., "Visual Interface to Multimedia Databases", in Handbook of Multimedia Information Systems, W. I. Grosky, R. Jain and R. Mehrotra, eds., Prentice Hall, 1997, pp 167-187.

[8] Chang, S.-K. and Jungert, E., Symbolic Projection for Image Information Retrieval and Spatial Reasoning, Academic Press, 1996.

[9] Chang, S.-K. and Jungert E. Pictorial data management based upon the theory of Symbolic Projection, Journal of Visual Languages and Computing, vol. 2, no 3, 1990, pp 195-215.

[10] Chang, S.-K., Shi, O. Y. and Yan, C. W., Iconic indexing by 2D strings, IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI), Vol. 9, No. 3, pp. 413-428, 1987.

[11] Egenhofer, M., "Spatial SQL: A Query and Presentation Language", IEEE Transactions on Knowledge and Data Engineering, Vol. 5, No. 2, 1991, pp. 161-174.

[12] Holden, R., "Digital's DB Integrator: a commercial multi-database management system", Proceedings of 3rd International Conference on Parallel and Distributed Information Systems,

Austin, TX, USA, Sept. 28-30 1994, IEEE Comput. Soc. Press, Los Alamitos, CA, USA, pp. 267-268.

[13] E. Jungert, C. Carlsson and C. Leuhusen, "A Qualitative Matching Technique for Handling Uncertainties in Laser Radar Images", Proceeding of the SPIE conference on Automatic Target Recognition VIII, Orlando, Florida, April 13-17, 1998, pp 62-71

[14] Jungert, E., "A qualitative approach to recognition of man-made objects in laser-radar images", Proceedings of the conf. on Spatial Data Handling, Delft, The Netherlands, August 13-16, Vol II, pp A15-A26.

[15] Lee, S.-Y. and Hsu, F.-S., Spatial reasoning and similarity retrieval of images using 2D Cstring knowledge representation, Pattern Recognition, vol. 25, 1992, pp 305-318.

[16] Lin, C. C., Chang, S. K. and Xiang, J. X., Transformation and Exchange of Multimedia Objects in Distributed Multimedia Systems, ACM Journal of Multimedia Systems, Springer-Verlag, Vol. 4, Issue 1, 1996, 12-29.

[17] Li, J. Z., Ozsu, M. T., Szafron, D. and Oria, V., "MOQL: A multimedia Object Query Language", Proceedings of Third International Workshop on Multimedia Information Systems, Como, Italy, September 1997, pp. 19-28.

[18] Oomoto E. and Tanaka, K., Video Database Systems - Recent Trends in Research and Development Activities, in Handbook of Multimedia Information Management, (Grosky, W. I., Jain, R. and Mehrotra, R., eds.), Prentice Hall, 1997, pp 405-448.

[19] Roussopoulos, N., Faloutsos, C. and Sellis, T., "An Efficient Pictorial Database System for PSQL", IEEE Transactions on Software Engineering, Vol. 14, No. 5, May 1988, pp. 639-650.

[20] Scotti, P., "Multimedia interface: an integration of video, audio and data in 3D environment", Proceedings of the IEEE International Symposium on Industrial Electronics, Guimaraes, Portugal, July 7-11, 1997, IEEE, New York, NY, USA, vol.1, pp. SS235-SS237.

[21] Srihari, R. K. and Zhang, Z. F., "Finding Pictures in Context", Proceedings of MINAR'98, Hong Kong, August 1998, Lecture notes in computer science No. 1464, (Ip and Smeulders, eds.) Springer Verlag, pp 109-123.

[22] Vazirgiannis, M., Multimedia Data Object and Application Modelling Issues and an Object Oriented Model, Multimedia Database Systems: Design and Implementation, (Nwosu, K. C., Thuraisingham, B.and Berra P. B., Eds), Kluwer Academic Publishers, 1996, pp 208-250.

[23] E. Waltz and J. Llinas, "Multisensor data fusion", Artect House, Boston, 1990.

[24] Xu, X. B.; Shi, B. L. and Gu, N., "FIMDP: an interoperable multi-database platform", Proceedings of 8th International Hong Kong Computer Society Database Workshop. Data Mining, Data Warehousing and Client/Server Databases, Hong Kong, July 29-31 1997, SpringerVerlag Singapore, Singapore, pp 166-176.

[25] Yaginuma, Y. and Sakauchi, M., "Moving TV image analysis based on multimedia fusion focusing on extracted common concepts", Proceedings of IECON '93 - 19th Annual Conference of IEEE Industrial Electronics, Maui, HI, USA, Nov. 15-19 1993, IEEE, New York, NY, USA, vol.3, pp. 1803-1807.