Querying Distributed Multimedia Databases and Data Sources for Sensor Data Fusion

Shi-Kuo Chang, *Fellow, IEEE*, Gennaro Costagliola, *Member, IEEE*, Erland Jungert, and Francesco Orciuoli

***Abstract*—Sensor data fusion imposes a number of novel requirements on query languages and query processing techniques. A spatial/temporal query language called ΣQL has been proposed to support the retrieval and fusion of multimedia information from multiple sources and databases. In this paper we investigate fusion techniques, multimedia data transformations and ΣQL query processing techniques for sensor data fusion. Fusion techniques including fusion by the merge operation, the detection of moving objects, and the incorporation of belief values, have been developed. An experimental prototype has been implemented and tested to demonstrate the feasibility of these techniques.**

***Index Terms*—Data fusion, distributed database, multimedia database, query language, query processing, sensor data fusion.**

I. Introduction

SENSOR data fusion is an area of increasing importance that requires novel query languages and query processing techniques for the handling of spatial/temporal information. Sensors behave quite differently from traditional database sources. Most sensors are designed to generate information in a temporal sequence. Sensors such as video camera and laser radar also generate large quantities of spatial information. Therefore the query language and query processing techniques must be able to handle sources that can produce large quantities of streaming data within short periods of time.

Another aspect to consider is that user's queries may be modified to include data from more than one sensor and therefore require the fusion of multiple sensor information. In our empirical study we collected information from different type of sensors, including laser radar, infrared video (similar to video but generated at 60 frames/s) and CCD digital camera. In a preliminary analysis of the above described sensor data, it is found that data from a single sensor yields poor results in object recognition. For instance, the target object may be partially hidden by an occluding object such as a tree, rendering certain type of sensor ineffective. Object recognition can be significantly improved if the query is modified to obtain information from another type of sensor, while allowing the target being partially hidden. In other words, one (or more) sensor may serve as a guide to the other sensors by providing status information such as position, time and accuracy, which can be incorporated in multiple views and formulated as constraints in the refined query.

Existing query processing techniques are not designed to handle sensors that produce large quantities of streaming data within short periods of time. With existing query languages such as SQL, it is also difficult to systematically refine the query to deal with information fusion from multiple sensors and distributed databases. To support the retrieval and fusion of multimedia information from multiple sources and distributed databases, a spatial/temporal query language called ΣQL has been proposed [6]. ΣQL is based upon the σ-operator sequence and in practice expressible in a syntax similar to SQL. ΣQL allows a user to specify powerful spatial/temporal queries for both multimedia data sources and multimedia databases, eliminating the need to write separate 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 the selected multimedia static schema.

The main contribution of this paper is to provide a systematic approach consisting of fusion techniques, multimedia data transformations and query processing techniques for sensor data fusion. The paper is organized as follows. Section II presents background and related research. The basic concept of the dual representation of the σ-query is explained in Section III. The usage of the various types of operators is discussed in Section IV. The techniques of sensor data fusion are explained in Section V. Section VI describes the architecture of the system for sensor data fusion. Section VII presents the data model, and Section VIII the detailed syntax of ΣQL and some examples. In Sections IX–XI, we discuss sensor data fusion using the merge operation, the detection of moving objects in a video, and the incorporation of belief values, respectively. Section XII concludes the paper.

II. Background and Related Research

Sensor data fusion posed some special problems. First of all, there is no general solution to the problem of sensor data fusion for an unlimited number of different types of sensors. The problem is usually restricted to a limited number of object types observed from a specific perspective by a limited number of sensors [22]. One such example is to select sensors that are looking only at ground objects, primarily vehicles, from a top view perspective where the sensors are carried by a flying platform such as a helicoptor. By studying this restricted problem in detail, we may be able to understand better how to deal with complex

Manuscript received February 12, 2002; revised December 10, 2002. The associate editor coordinating the review of this manuscript and approving it for publication was Dr. Sankar Basu.

S.-K. Chang is with the Department of Computer Science, University of Pittsburgh, Pittsburgh, PA, 15260 USA (e-mail: chang@cs.pitt.edu).

G. Costagliola is with the Dipartimento di Matematica ed Informatica, Università di Salerno, Salerno, Italy (e-mail: gencos@unisa.it).

E. Jungert is with the Swedish Defense Research Institute (FOA), SE-172 90 Linköping, Sweden (e-mail: jungert@foi.se).

F. Orciuoli is with the Dipartimento di Ingegneria dell'Informazione e Matematica Applicata, Università di Salerno, Salerno, Italy (e-mail: orciuoli@crmpa.unisa.it).

Digital Object Identifier 10.1109/TMM.2004.834862

queries for sensor data fusion. For a more general view on sensor data fusion, see e.g., Waltz and Llinas [21].

As explained in the preceding section, sensor data fusion requires a query language that supports sensor sources and the systematic modification of queries. In early research on query modification, queries are modified to deal with integrity constraints [19]. In query augmentation, queries are augmented by adding constraints to speed up query processing [8]. In query refinement [20], multiple term queries are refined by dynamically combining precomputed suggestions for single term queries. Recently query refinement technique was applied to content-based retrieval from multimedia databases [3]. In our approach, the refined queries are manually created to deal with the lack of information from a certain source or sources, and therefore not only the constraints can be changed, but also the source(s). This approach has not been considered previously in database query processing because usually the sources are assumed to provide the complete information needed by the queries.

In addition to the related approaches in query modification, there is also recent research work in agent-based techniques that are relevant to our approach. Many mobile agent systems have been developed [1], [2], [16], and recently mobile agent technology is beginning to be applied to information retrieval from multimedia databases [15]. It is conceivable that sensors can be handled by different agents that exchange information and cooperate with each other to achieve fusion. However mobile agents are highly domain-specific and depend on ad-hoc, 'hardwired' programs to implement them. In contrast our approach offers a framework for data transformation and query processing and is applicable to different type of sensors, thus achieving a certain degree of sensor data independence.

III. The Dual Representation of the ΣQL Query Language

As noted in Section I, ΣQL is a spatial/temporal query language for information retrieval from multiple heterogeneous sources and databases. Unlike SQL, which does not explicitly deal with spatial/temporal queries, ΣQL is designed with that purpose in mind. Unlike SQL, which deals only with databases, ΣQL is designed to deal with both static sources (databases) and dynamic sources (sensors), and furthermore these sources are distributed. 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 is useful in theoretical investigation, while the SQL-like query language is easy to implement and is a step toward a user-friendly visual query language. An example is illustrated in Fig. 1. The *source* R, also called a *universe*, consists of time slices where each time slice consists of objects with the same time value. To extract three predetermined time slices from the source R, the σ-query in mathematical notation is $\sigma_{\mathrm{t}}\ (\mathrm{t}_1, \mathrm{t}_2, \mathrm{t}_3)\ \mathrm{R}$.

The meaning of the σ-operator in the above query is "select," i.e., we want to select the time dimension and three slices along this dimension. The subscript t in σ_{t} indicates the selection of the time dimension. In the SQL-like language the ΣQL query is expressed as

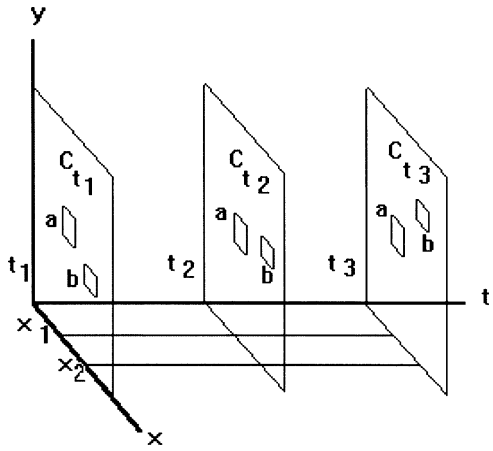


Fig. 1. Example of extracting three time slices from a source.

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

A new keyword "CLUSTER" is introduced so that the parameters such as t_1, t_2, t_3 for the σ-operator can be listed. The word "CLUSTER" indicates that objects belonging to the same subset of the universe (i.e.,, a cluster) must share some common characteristics (such as having the same time value, being similar to one another, etc.) Clustering is a technique used in pattern classification to form subsets of similar objects. A cluster may have a substructure specified in another (recursive) query. Clustering is a natural concept when dealing with spatial/temporal objects that are specifiable only through similarity to other objects. The mechanism for clustering will be discussed further in Section VIII. 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 IV.

The dual representation of ΣQL means that a query can be formulated as an SQL-like query [6] or as a sequence of *generic* operators (the σ-operators introduced above) and *specialized* operators (the ϕ-operators to be discussed in the following section). Translation from one representation to the other is quite straightforward.

The operators may handle both qualitative and quantitative information. Primarily, the operators allow operations on a sensor-data-independent level, i.e., sensor data should be transformed into information structures at high abstraction levels that are sensor independent. To accomplish this, the queries are expressed in terms of operator sequences where the transformations of the sensor data are carried out stepwise by the operators. The operators reduce the dimensions of the multudimensional search space to which each successive operator is applied. Intuitively, the reduced search space is also another *cluster*. Thus, as successive operators are applied, the clusters become more and more refined.

In contrast to this refinement of the search space, the *fusion* and related operators take multiple clusters as input and fuse the information to determine a belief value that may support a certain hypothesis such as whether different observations in the

clusters correspond to the same object. Furthermore the fusion and related operators can handle uncertain information through the use of belief values.

IV. Operator Classes

The operators in ΣQL can be categorized with respect to their functionality. The two main classes are the *transformational operators* (the σ-operators) and the *fusion operators* (the ϕ-operators). In this section the two main operator classes are discussed according to their input, output and functionality.

A. σ-Operators

A σ-operator is defined as an operator to be applied to any multudimensional source of objects in a specified set of intervals along a dimension. The operator projects the source along that dimension to extract clusters [6]. Each cluster contains a set of objects or components whose projected positions fall in one of the given intervals along that dimension. As an example, let us write a σ-expression for extracting the video frame sequences in the time intervals $[t_1 - t_2]$ and $[t_3 - t_4]$ from a video source VideoR. The expression is $\sigma_{\text{time}}([t_1 - t_2], [t_3 - t_4])$ VideoR where VideoR is projected along the time dimension to extract clusters (frames in this case) whose projected positions along the time dimenson are in the specified intervals.

In case of uncertainty, the components of the clusters may be associated with various probabilities or belief values. Input and output data may be of either qualitative or quantitative type, although generally the later type is of main interest. Thus input data will be accessed from either a raw-data source such as a sensor, or from a structured data source such as a database, or from some internal source such as *qualitative strings* that are strings consisting of object descriptions projected along certain dimension(s) [6]. The output data correspond to clusters in relational representations that in practice may be available as qualitative strings of various types [6]. The general formalism can thus be expressed in the following way:

$$\sigma_{\text{dimension}}\left(\langle\text{intervals}\rangle\right)\{\langle\text{source}\rangle|\langle\text{cluster}\rangle|\langle\text{cluster\_strings}\rangle|\ldots\} \Rightarrow \{\langle\text{cluster}\rangle|\langle\text{cluster\_strings}\rangle|\ldots\}.$$

A variety of σ-operators can be defined [11]. Many of these operators are common in most spatial applications. Examples are the determination of various attributes and spatial relations, such as "northwest-of," "to the left of," etc. For simple inputs, these operators can be described as

$$\sigma_{\text{attribute}}\left(\langle\text{attribute} - \text{values}\rangle\right)\langle\text{cluster\_strings}\rangle \Rightarrow \langle\text{relational\_strings\_of\_}(\langle\text{attribute}\rangle\langle\text{object}\rangle\langle\text{attribute value}\rangle)\rangle$$

$$\sigma_{\text{relation}}\left(\langle\text{relational\_values}\rangle\right)\langle\text{cluster\_strings}\rangle \Rightarrow \{(\langle\text{relational}\rangle(\langle\text{relational\_value}\rangle\langle\text{object}\rangle - \text{i}\langle\text{object}\rangle - \text{j}))\}.$$

As an example to find a pair of objects such that the blue object precedes ($<$) the red object along the spatial dimension V, the σ-operator instance is

$$\sigma_{\text{color}}(\text{blue}, \text{red})(\text{V} : \text{A} < \text{B}) = (\text{V} : (\text{color A blue}) < (\text{color B red}))$$

where $(\text{V} : \text{A} < \text{B})$ is a cluster string.

In case of uncertainty the input and output to the σ-operators may include an attribute corresponding to a specific belief value. The σ_{type}- and the σ_{motion}-operators may include this attribute. The σ_{type}-operator is concerned with matching between objects found in a sensor image and objects stored in a library database and where both objects are described in the same terms that may be either qualitative or quantitative. Traditionally matching was regarded as a part of information fusion. Generally, however, the σ_{type}-operator and its result, i.e., a set of objects and their corresponding normalized belief values, can be expressed as follows when the input to the operator is a single cluster:

$$\sigma_{\text{type}}(\text{type\_value})\langle\text{cluster\_strings}\rangle \Rightarrow \langle\text{cluster\_strings\_of\_}(\langle\text{type\_value}\rangle\langle\text{object}\rangle - \text{i}\langle\text{nbv}\rangle - \text{i})\rangle$$

where $\langle nbv\rangle$ corresponds to the normalized belief value that in practice becomes an attribute to the actual object. An instance of this is

$$\sigma_{\text{type}}(\text{car})(\text{U} : \text{A} < \text{B} < \text{C}) = (\text{U} : (\text{car A } 0.7) < (\text{car B } 0.1) < \text{C}).$$

The σ_{motion}-operator can be expressed as follows:

$$\sigma_{\text{motion}}\left(\langle\text{motion\_value}\rangle\right)\{\langle\text{cluster} - \text{sequence}\rangle|\langle\text{cluster}\rangle\} \Rightarrow \{(\langle\text{motion\_value}\rangle\langle\text{object}\rangle - \text{i})\}.$$

For example, to find three moving objects, each preceding the other, along the spatial dimension U, the operator instance is:

$$\sigma_{\text{motion}}(\text{moving})(\text{U} : \text{A} < \text{B} < \text{C}) = (\text{U}(\text{moving A}) < (\text{moving B}) < (\text{moving C})).$$

B. ϕ-Operators

The ϕ-operators are more complex because they are concerned with sensor data fusion. Consequently these operators require more complex expressions as well as input data in different time periods from multiple sensors.

The ϕ_{fusion}-operator performs sensor data fusion from heterogeneous data sources to generate fused objects. Fusion of data from a single sensor in different time periods is also allowed. The output of the ϕ_{fusion}-operator is some kind of high level, qualitative representation of the fused object, and may include object type, attribute values and status values. The output may also include a normalized belief value for each fused object.

The fusion operators rely upon solutions to the association problem [11], which is generally concerned with how to determine whether an object of a certain class observed at one time is the same object observed at a later time. The observations may be made by the same sensor or by different sensors of either the

same or different types. This is a complex problem [11] that normally requires probability-based approaches such as Baysian networks [10] or Dempster–Shafer theory of evidence [23].

The output from the fusion operator may serve as the answer to a query. This result may consist of a list of objects each having a belief value. The object with the highest belief value is the most likely answer to the query and thus should come first in the list. The general description of the fusion operator is therefore

$$\phi_{\text{fusion}}(\text{cluster}|\text{cluster}-\text{string}) \Rightarrow \{(\text{type},\text{obj},\text{nbv})|(\text{type},\text{obj},\text{nbv})-\text{list}\}.$$

Last, but not least, a $\phi_{\text{similarity}}$-operator that takes two images from either the same sensor or two different sensors, transforms them into unified cluster data and then establishes the similarity between the two with respect to their contents, can be described as

$$\phi_{\text{similarity}}(\text{image}-\text{cluster}_1,\text{image}-\text{cluster}_2) \Rightarrow \{(\langle\text{similar}\rangle\text{image}-\text{cluster}_1,\text{image}-\text{cluster}_2,nbv)\}.$$

The similarity operator relies upon qualitative techniques such as the technique described in [7]. Similarity retrieval has for a long time been part of image information retrieval and includes techniques for iconic indexing [4]. However in similarity retrieval the complete content of the images, instead of just single objects, is of concern. Similarity retrieval is also less concerned with the identity of the objects but with sets of objects, and their relationships and positions.

V. Sensor Data Fusion

In sensor data fusion [4], queried objects from the different sensors need be associated to each other in order to determine whether they are the same object registered at different times and at different locations. Tracking is another problem that is closely related to the sensor data fusion problem. In tracking, the problem is to verify whether different object observations represent the same or different objects. Another problem of concern is the uncertainties of the sensor data. Consequently, there is a demand for a well-structured and efficient general framework to deal smoothly with a large number of different query types with heterogeneous input data.

In sensor data fusion, the main issue is how to develop a general framework that can be applied to carry out the fusion process in query processing. The fusion framework that will be applied in this work is based on a method described by Horney *et al.* [25], which makes use of a technique that is quite general and can be applied not only to fusion of sensor data but also to other problems. However this fusion method should be *replaceable* by other methods, which will be tested later. The fusion method is chosen because it is efficient, simple to implement, demonstrates a high degree of generality and fits well into the environment of concern in this research work. Other related approaches to fusion are given by Parker [18] and Klein [14].

For certain sensors the objects can only be determined with respect to their type but rarely with respect to their identity. Therefore classification of the objects is necessary. This is a process that can be carried out in a matching algorithm that should display a result that includes not only the type of the object but a normalized belief value, nbv, associated to the observed object type. A number of attributes and state variables can be extracted from the sensor data where the actual types of attributes depend on the actual sensor types. Among the most important state variables are orientation, type and position, direction of motion, speed and acceleration. Most attributes and state variables, such as position and orientation, may be determined either in quantitative or in qualitative terms. In ΣQL, reasoning is generally performed on qualitative information, basically represented in terms of Symbolic Projection. For this reason, it is an advantage to use qualitative input to the fusion process as well.

TABLE I
Example of Basic Spatial State Values and Their Belief Values and With Their Qualitative Value Sets

attribute	Belief value	value set
Position	$B(A=B \mid pos_B)$	{vl, l, m, h}
Direction	$B(A=B \mid dir_B, pos_B)$	{l, m, h}
Type	B(A)	{l, m, h}

The following probabilities are determined during the fusion process for indication of the belief values in different state variables.

The probability $\mathrm{P(A = B)}$ is the probability that object observation A is the same as object observation B.

$\mathrm{P(A = B|pos_B)}$ is the probability that the observations A and B can be associated to each other given the position of B relative the position of observation A.

$\mathrm{P(A = B|dir_B, pos_B)}$ is the probability associating the observations A and B to each other given the position of B and its direction relative the position of A.

These probabilities can be replaced by a set of *belief values* as can be seen in Table I. Here the belief values are of qualitative type but they may be quantitative as well.

Given some spatial attributes and their probabilities or normalized belief values, as in Table I, the basic means to carry out the fusion process are available. The attributes/state values and their belief values correspond to information obtained from the sensor data as a consequence of the queries. However, the result of this process must clearly be represented in a structure that efficiently supports the fusion process. The structure proposed for completion of the fusion process is to a high degree related to a scheme proposed by Chong *et al.* [7]. The structure here is called a *fusion scheme*. A difference between the two schemes is that the one used by Chong *et al.* is merely concerned with determination of tracks of objects, i.e., both the tracks registered by individual sensors and tracks from the fusion process, which they call a system track. In this work, the scheme is concerned with all information that relates to the objects, i.e., including state variables and attributes.

Generally, the information acquired from the sensors is stored as instance descriptions and eventually forwarded into the fusion composition table (FCT), which serves as input to the fusion routine. FCT is organized as a set of *pages*, one for each observation. These pages are called object instance pages (OIP).

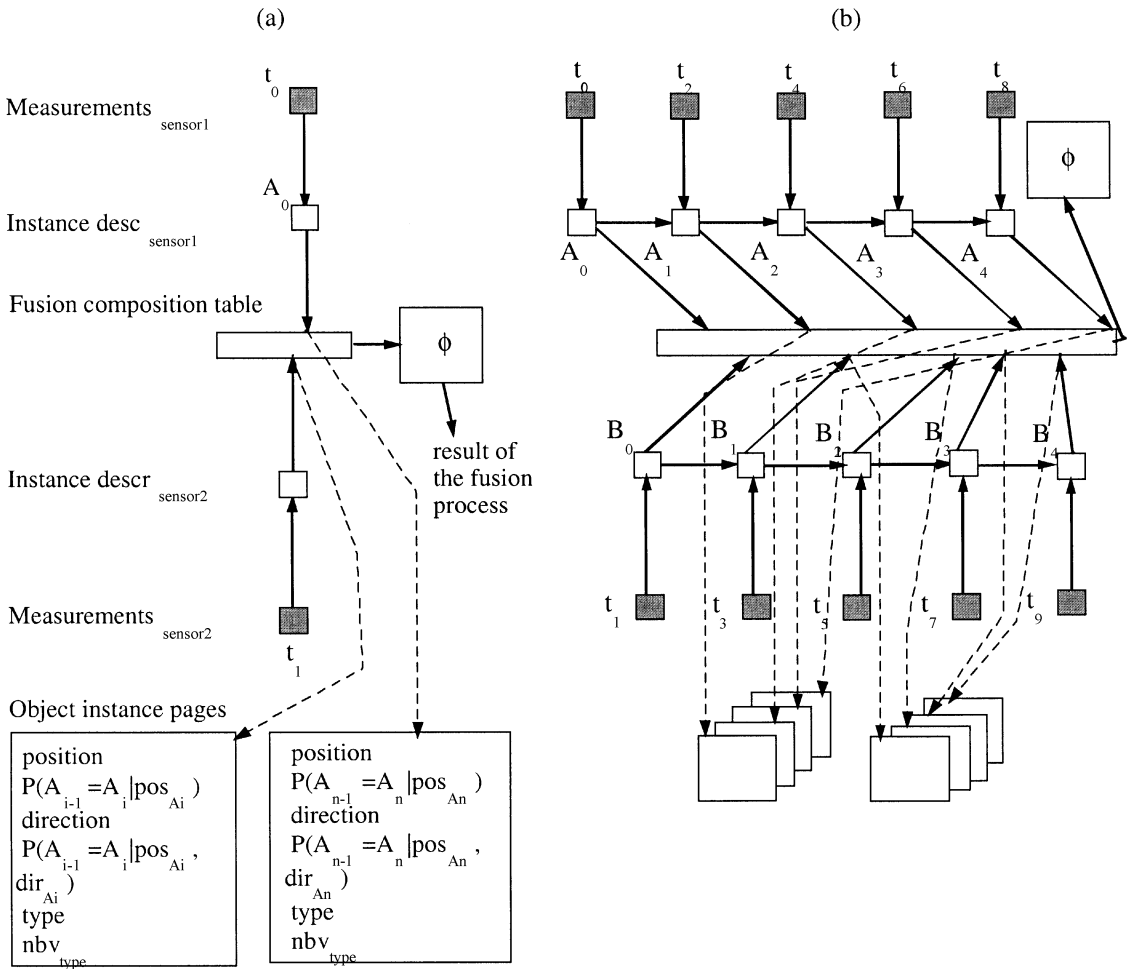


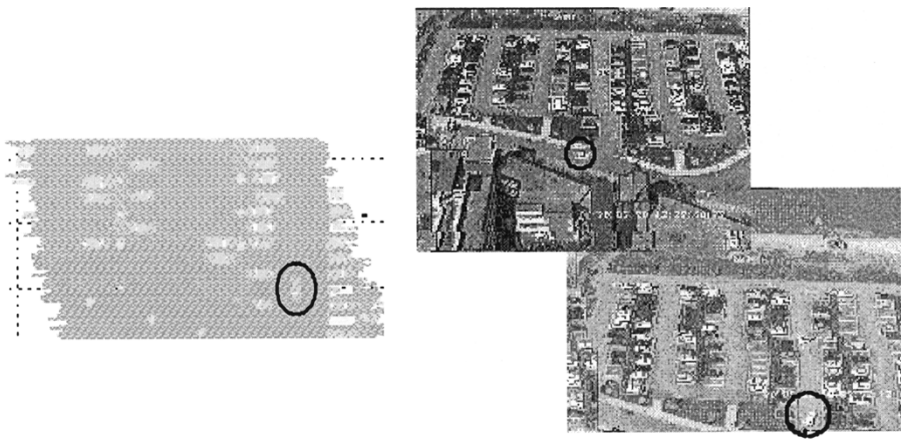
Fig. 2. (a) Flow of the sensor data fusion process for two single instance sensors. (b) Flow of the sensor data fusion process for two sensors with multiple instances.

In some applications it may be convenient to represent these pages in terms of HTML code since that way it may be possible to carry out the fusion process in a distributed environment. The fusion process can now be seen as a process where information acquired from a set of sensors is fed into the query system and its fusion routine. Query execution corresponds, in principle, to a process where a multudimensional search space through projection is reduced to clusters in lower dimensionality. In the extreme, a cluster may correspond to a single object instance, which together with its belief value is fed into the FCT. Eventually the fusion is carried out by the ϕ-operator. This scheme is automatically created by the system. The structure of the scheme may vary depending on the type of sensor information that is under consideration, e.g., image sequences from a video generate sequences of pages while for single images just a single page is generated [see Fig. 2(a) and (b)].

The fusion process proceeds all through the execution of the query feeding the FCT and produces a response to a proposition, i.e., which of the registered objects can be associated to each other. The registered objects are determined by the query interpreter and inserted into the FCT for a fusion step. Since the object instance pages in the FCT contains the actual attribute values and their corresponding qualitative belief values it is feasible for the ϕ-operator to answer propositions like: "*are all the observations in the FCT corresponding to the same object*." The result of the fusion process will tell which object observations that can be associated to each other, although the result, to some degree, exhibits a qualitative uncertainty, which must be evaluated by the user in his final decision process.

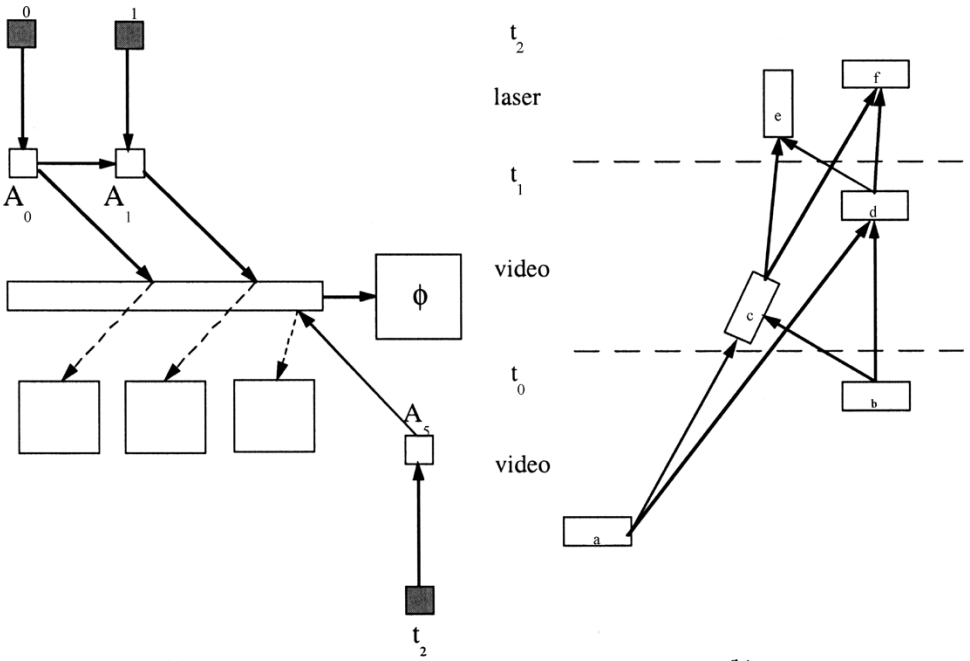
The fusion method is applied to the FCT under consideration of the current proposition, which depends on the query, the sensor types and the data. For all the object combinations and with respect to the qualitative belief values for the various attributes and state variables, the process is carried out with respect to the given proposition. The value set of the qualitative belief values is {h(igh belief), m(edium belief), l(ow belief)}. In practice, this means that a score need to be given to the actual voting alternative of FCT, which may be 3 for a high belief value, and so on. The scores given for a certain attribute alternative may be added up and a total score is available for each alternative. The alternative with the highest score is considered the most probable and the remaining alternatives are ordered with respect to their rank.

A query demonstrating the usage of ΣQL with the sensor sources video and laser-radar can now be provided. Given the input information from the laser-radar [Fig. 3(a)] and the video camera [Fig. 3(b)], the query is as follows: *is there a moving vehicle present in the given area and in the given time interval*? In this query, the laser-radar data can be used as an index to the video. In this way, most of the computational efforts can be



(a) (b)

Fig. 3. (a) Laser radar image of a parking lot with a moving car (encircled). (b) Two video frames showing a moving white vehicle (encircled) while entering a parking lot.



(a) (b)

Fig. 4. (a) Flow of the sensor data fusion process for the video/laser-radar example. (b) Set of objects observed at different times and where the arrows indicate the different situations in a simplification of Fig. 3(a) and Fig. 3(b).

avoided since the vehicles can be identified in almost real time in a laser-radar image. However, in the laser-radar used here, it cannot be determined whether an identified vehicle is moving or not. Consequently, once a vehicle has been identified in a laser–radar image, we need to determine whether it is moving by analyzing a small number of video frames taken in a short time interval. This is possible to accomplish because the location of the vehicle at a certain time is known from the laser-radar information, which is illustrated in Fig. 3(a) and (b). The three images illustrate a situation where a car is first about to enter a parking lot (the two video frames) and at a later time the car has entered the parking lot (the laser-radar image).

The query is logically split into two parts, one looking for the vehicles in the laser-radar image and another looking for the vehicles in the video frames during the same time interval as the laser–radar image. The result of these two subqueries are fused by applying the fusion operator $\phi_{\text{type,position,direction}}$, which includes the fusion procedure with the voting scheme. The fusion is applied with respect to type, position and direction including also their belief values.

The query demonstrating the problem can thus be expressed as

$$\begin{aligned}
&\phi_{\text{pe,position,direction}}\\
&(\sigma_{\text{motion}}(\text{moving})\sigma_{\text{type}}(\text{vehicle})\\
&\sigma_{\text{xy}}(*)\\
&\sigma(\text{T})_{\text{T mod 10=0 and T>t1 and T<t2}}\\
&\sigma_{\text{media_sources}}(\text{video})\text{media_sources}\\
&\sigma_{\text{ype}}(\text{vehicle})\sigma_{\text{xyz}}(*)\\
&\sigma(\text{T})_{\text{T>t1 and T<t2}}\\
&\sigma_{\text{media_sources}}\ (\text{laser_radar})\text{media_sources})\,.
\end{aligned}$$

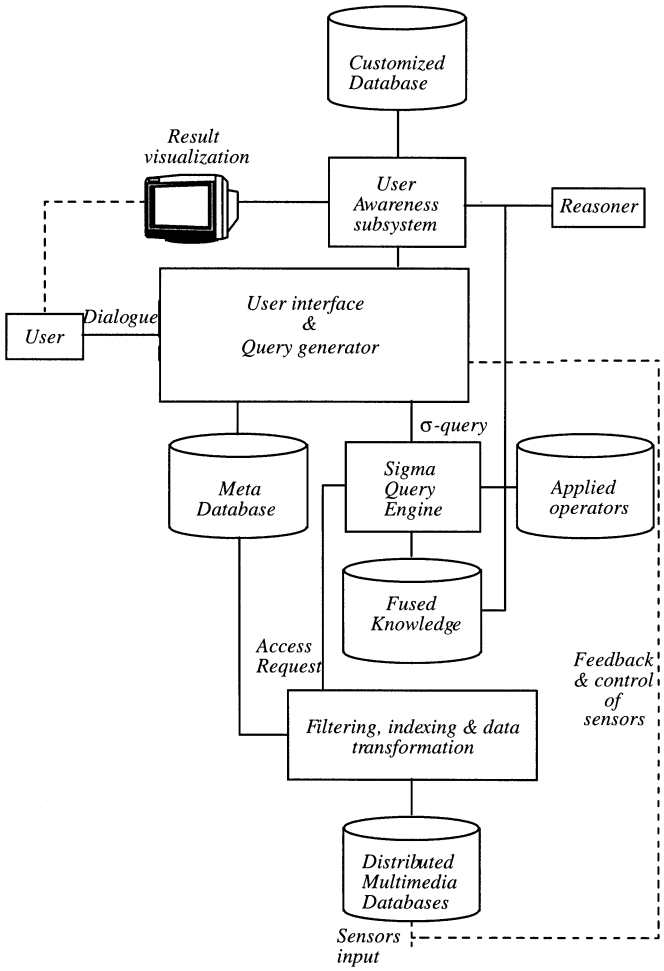


Fig. 5. System architecture for sensor data fusion.

The ϕ-operator performs the fusion process during the execution of the query. Each vehicle identified in any of the images during the query is subject to the determination of these state variables and attributes and their belief values. Fusion is then completed with respect to all the possible combinations of the identified vehicles. The proposition that takes place in the fusion process for this query can thus be formulated as: *can observation A be associated to observation B*?

A simplified description of the situation in Fig. 3(a) and Fig. 3(b) is shown in Fig. 4(b), where just one of the parked cars is left, i.e., the one to the right. The arrows or rather all their combinations taken in sequence correspond to all the possible combinations that will be subject to fusion. Consequently there are eight alternatives to discriminate between.

VI. System Architecture for Sensor Data Fusion

Fig. 5 illustrates a generic system architecture for sensor data fusion. A *user* interacts with a user interface to produce a *σ-query*. This can be done directly or through a domain specific virtual environment customized to the user's level of expertise and described in the *user awareness subsystem*. Once a *σ-query* has been formulated, it can be compiled and its correctness and feasibility checked. For a *σ-query* to be executable all the required operators must have been implemented in the system.

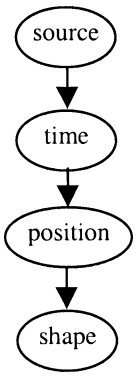


Fig. 6. Hierarchy of dimensions.

The knowledge of what type of queries the system can execute is given in a knowledge base formed by the *Meta Database* and the *Applied operators*. The Meta Database contains a set of tables describing which operators are implemented in the system and how they can be used. The Applied operators are the set of algorithms that implement the operators. Once a query has been successfully compiled the *Sigma Query Engine* executes it against the *Distributed Multimedia Database* or directly against the *sensors input*. During the execution it applies *input filtering, indexing and data transformation* required by the application of the operators in the query. The execution of a query produces *(Fused) Knowledge* that can then be used to modify the virtual environment in which the user operates, providing useful feedback through appropriate *result visualizations*.

One of the important characteristics of this architecture is that the same query language can be used to access any data source. This is possible due to the fact that the Meta Database and the Applied Operators hide the information to be processed, providing the ΣQL processor with a general data model, on which programmers base their queries.

VII. Data Model

The ΣQL query language makes use of the Dimension Hierarchy Data Model (DHDM) to support a common and uniform treatment of heterogeneous data sources. The dimension hierarchy is a generic structure, based upon which different instances of representation schema can be constructed to best suit the sources and the queries. In this data model, a data source type is represented by a set of table schemas called *Representation Schema* (RS for short). Each table schema describes a *dimension* along which to cluster the sources of that type and is characterized by a name and a list of fields: $\mathrm{dim} = [\mathrm{attr}, \mathrm{dim}_1, \mathrm{dim}_2, \ldots, \mathrm{dim}_\mathrm{k}]$ (with $\mathrm{k} >= 0$) where

- dim is the name of the table schema;
- attr is a proper attribute of dim;
- $\mathrm{dim}_1, \ldots, \mathrm{dim}_\mathrm{k}$ are the names of other table schemas.

Given an RS, a *Representation Schema Instance* (RSI) of a source is a set of instances of the table schemas in RS. In the following, an instance of a table schema dim will be referred to as a table of type dim and is composed by a possibly ordered set of rows $(val_{attr}, dval_1, dval_2, \ldots, dval_k)$ where val_{attr} is a value for the attribute $attr$ of dim and $dval_j$ is a reference to a table of type dim_j.

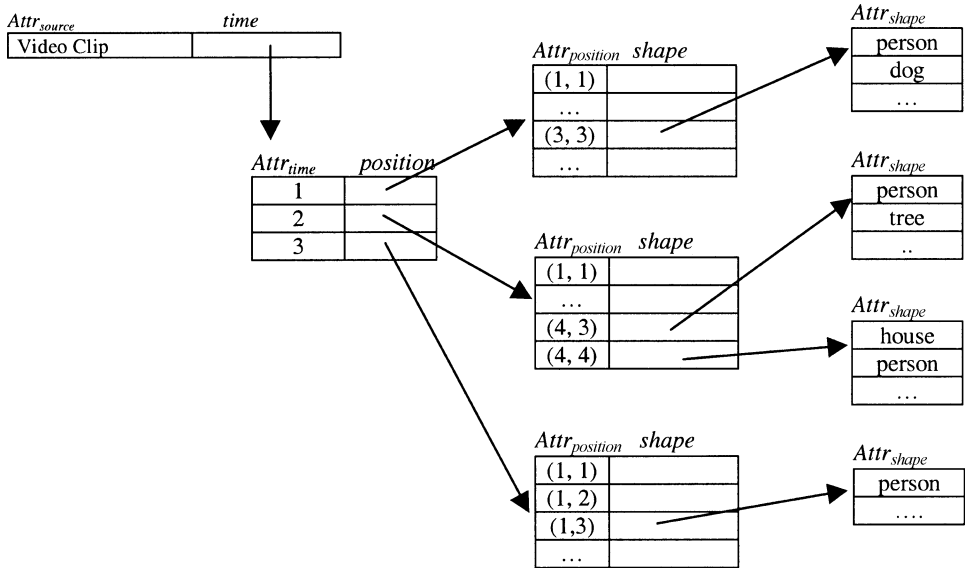


Fig. 7. RSI of a short video clip illustrating the spatial/temporal position of the people shown in the video clip.

Let us give an example showing how to represent a video clip according to DHDM. The first thing to do is to create an RS for video clips. The main idea is to successively project or decompose the video clip along some dimensions. On one hand, the type and number of dimensions strictly depends on the level of detail of the video clip representation we need to reach in order to get to the entities of interest occurring in the video clip. On the other hand, each dimension must have been implemented as an operator that must be present in the *Applied operators* repository. As an example, let us suppose we want to represent a video clip and are interested in representing the spatial/temporal position of the people shown in it. The already implemented dimensions we want to use are then the $time$, the $position$ and the $shape$, in this order. The resulting RS is given by the following table schemas:

$$\begin{aligned} source &= [\mathrm{Attr}_{\mathrm{source}}, \mathrm{time}] \\ time &= [\mathrm{Attr}_{\mathrm{time}}, \mathrm{position}] \\ position &= [\mathrm{Attr}_{\mathrm{position}}, \mathrm{shape}] \\ shape &= [\mathrm{Attr}_{\mathrm{shape}}]. \end{aligned}$$

The initial dimension $source$ defines the sources to be processed $(\mathrm{Attr}_{\mathrm{source}})$ and the dimension $(time)$ along which these are to be decomposed. Similarly the dimension $time$ defines the temporal instances to be considered $(\mathrm{Attr}_{\mathrm{time}})$ and the dimension $(position)$ along which to decompose the resulting entities. The dimension $shape$ does not refer to any other dimension since we are not interested in further details. It can be noted that the chosen RS actually defines a hierarchy of dimensions along which to decompose the source. The underlying hierarchy is shown in Fig. 6.

Let us suppose that the video clip has three frames, each partitioned into 4×4 slots, and shows, among others things, four people in the slot position (3, 3) of the first frame, the slot positions (4, 3) and (4, 4) of the second frame, and the slot position (1, 3) of the third frame, respectively. Fig. 7 shows the corresponding RSI.

It should be noted that the table of type $source$ has only one entity, i.e., the video clip under consideration. This entity refers to a table of type $time$ containing three entities that, in the above example, are the three frames of the video. Each frame refers to a table of type $position$, each containing a set of entities corresponding to the slots partitioning the frame. Finally, the tables of type $shape$ contain entities corresponding to the actually recognized shapes in each slot. For the sake of clarity, not all the tables are shown in Fig. 7.

An alternative way of depicting the RSI shown in Fig. 7 is the graph view given in Fig. 8 where each dimension (table) is depicted with an oval and each entity (row) is depicted by a box. Again for the sake of clarity, not all the graph nodes are shown.

From the graph view of Fig. 8 the hierarchy of dimensions underlying the representation schema becomes evident. As a matter of fact, the hierarchy shown in Fig. 6 is easily built from Fig. 8 by disregarding the entity nodes.

Given a data source, there are many ways of representing it. This means that a source can be represented with different RSIs, depending on which information in the data source needs to be represented and then queried. Therefore, flexibility is the main motivation behind this approach. As an example, an alternative dimension hierarchy for representing a video clip is shown in Fig. 9. Here, again, we look at a video clip as a sequence of frames, but now each frame is seen as a set of separated rows and columns. Moreover, each frame row or column is seen as a set of shapes of a certain type.

VIII. ΣQL Language

To write a ΣQL query on a given data source, a programmer must know which *representation schemas* are available on that source and, for each schema, the corresponding dimension hierarchy. This is similar to SQL, where a programmer must know the table names and the table attributes of the relational database for the query. To better explain the behavior of a ΣQL query, we first need to refine the concept of clustering first introduced in Section III.

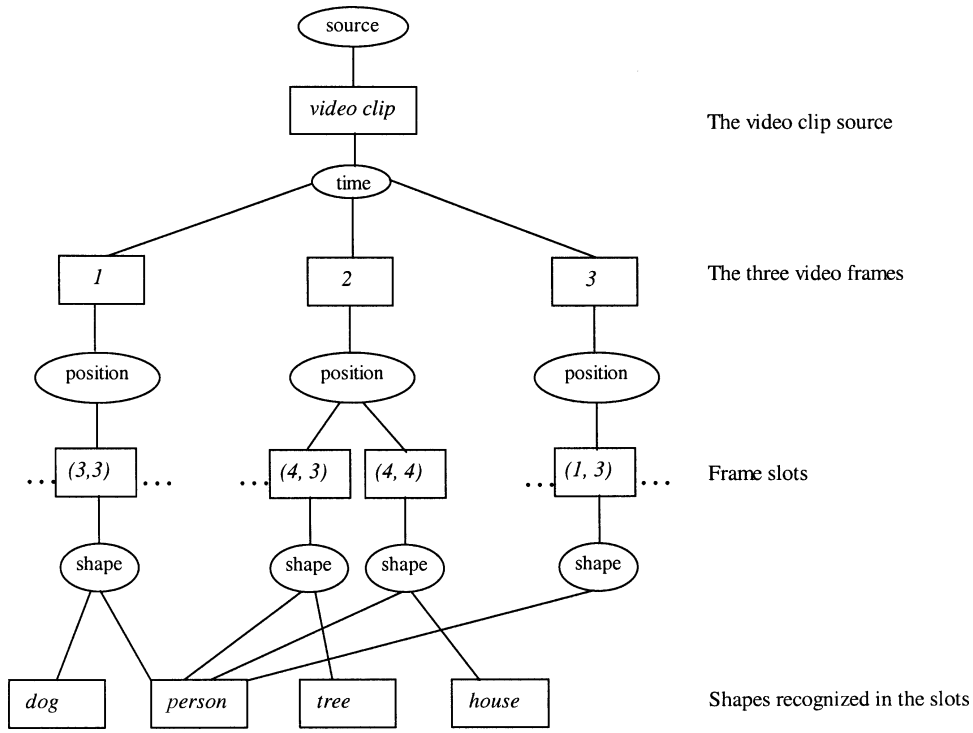


Fig. 8. Graph view of the RSI shown in Fig. 7.

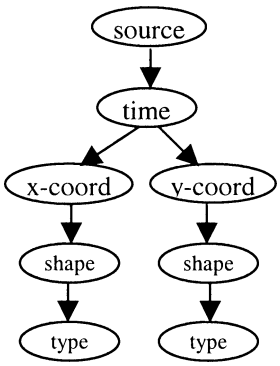


Fig. 9. Alternative hierarchy of dimensions to represent a video clip.

Given a *representation schema instance*, i.e., a set of instantiated tables representing a source, a *cluster* is represented by a set of table rows from one or more tables. Given a subset of tables *st* in a *representation schema instance*, with the term *clustering* we refer to the operation of creating a cluster from *st* by deleting or simply not selecting some of the table rows in *st*.

Examples of clustering on the RSI of Fig. 7, are

$\{time : *\}$ that creates a cluster, on the table of type "time," with all the video frames;

$\{time$: 1, 3$\}$ that creates a cluster, on the table of type "time," with only the video frames with time stamps 1 and 3;

$\{position : (3, *)\}$ that creates a cluster, on the tables of type "position," with all the video frame areas occurring in the third column of each frame.

A clustering operation on a subset *st* of tables can work in one of the following modes: *destructive mode*, where all the table rows or entities that are not selected are deleted from *st*; and *conservative mode*, where all the current table rows or entities that are not selected are kept in *st* but not included in the cluster, i.e., they are virtually deleted. They will be used to show the context of the selected cluster elements.

On the other hand, a cluster can be

open the cluster, created by a clustering operation, is open to successive destructive clustering operations, i.e., the elements in the cluster can be deleted by successive clustering and selection operations;

close the elements in the cluster can only be virtually deleted by successive clustering and selection operations.

As an example, if the clustering operation $\{time$: 1, 3$\}$ on the table of type "time" in Fig. 7 is considered destructive, then the second video frame, i.e., the second row in the table, will be deleted from the RSI.

Basically, a ΣQL query refines a source RSI by successively clustering it on some dimension under some constraints. In other words, by using clustering operations, the ΣQL query selects and/or deletes entities in a data source. The result of a query can be used by a nesting query to further refine the data source or can be output according to some presentation criteria.

The following is a simple ΣQL query template:

```
SELECT dimension_list
CLUSTER clustering_op_list
FROM source [query_re-
sult][{nested_query}]
[WHERE condition]
[PRESENT presentation_method]
```

The FROM clause requires the specification of the input to the query: this can be either the name of the data source, or the result of a previously executed query or a nested ΣQL query. The SELECT clause requires the list of the dimensions along which to cluster the input. The CLUSTER clause requires a list of clustering operations, one for each dimension declared in the SELECT clause. The form of a clustering operation is as those shown above:

$$\{dimension : [\text{filt\_mode}]\ \text{Val}^1_{dimension}, \ldots, \text{Val}^n_{dimension}\}$$

with the addition of the optional keyword "filt_mode" standing for one of the four filtering modes:

1) PUBLIC if the clustering operation is to be *destructive* and the resulting cluster is to be *open*.
2) PRIVATE if the clustering operation is to be *destructive* and the resulting cluster is to be *close*.
3) ALL&PUBLIC if the clustering operation is to be *conservative* and the resulting cluster is to be *open*.
4) ALL&PRIVATE if the clustering operation is to be *conservative* and the resulting cluster is to be *close*.

If no filtering mode is provided then the clustering operation is considered PUBLIC. Each $\text{Val}^i_{dimension}$ refers either to the wild character "$*$" indicating that all the dimension values must be considered, or to a constant value or to a variable whose constraints are to be set in the WHERE clause. The use of the filtering modes allow us to define the granularity of the context of a query result.

If the query is not nested in any other query it may not require any clustering and behave as a standard SQL query. In this case the keyword CLUSTER is followed by the wild character "$*$."

The PRESENTATION clause requires the name of a presentation module that should have been defined "ad hoc" for the presentation of the particular type of the query input. If omitted, the presentation of the query output will be done by using a default presentation module.

Given the dimension hierarchy in Fig. 6 representing a video clip, we want to write a query to retrieve the columns of the second frame in a video clip, containing some people.

```
// Subquery 4
SELECT type
CLUSTER *
FROM
  {
  // Subquery 3
  SELECT shape
  CLUSTER {shape: *}
  FROM
    {
    // Subquery 2
    SELECT x-coord
    CLUSTER {x-coord: *}
    FROM
      {
      // Subquery 1
      SELECT time
      CLUSTER {time: 2}
      FROM Video
      }
    }
  }
WHERE type = 'person'
```

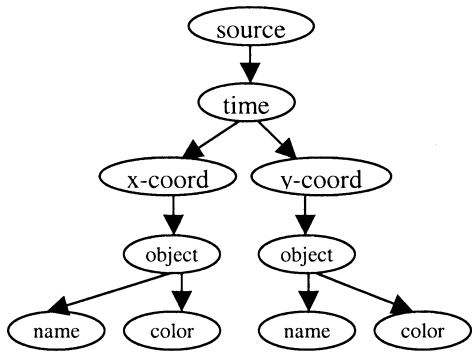


Fig. 10. Another possible hierarchy of dimensions for representing video clips.

The query must be read starting from the inner subquery and proceeding with the closest enclosing ones. Subquery 1 clusters the video source along the $time$ dimension and extracts the frame with time stamp 2. Subquery 2 extracts all the columns from the frame by clustering along the dimension *x-coord*. For each column, subquery 3 extracts all the shapes from all the columns by using the dimension $shape$.

Subquery 4 extracts all the people from all the shapes by asking for shapes of type "person." The final result will then be only the columns of frame 2 containing shapes representing people while all the other shapes will be lost. Note that subquery 4 does not use clustering but, more likely to an SQL query, uses the WHERE clause to further refine the data. This is possible because subquery 4 is not nested in any other subquery.

Given the hierarchy of dimension of Fig. 10, we will now write some queries to retrieve information from a video clip.

In the following, we will consider the simplified input video clip shown in Fig. 11.

Query 1: Extract all the video frame columns containing entities with the name John and entities with the name Bill.

```
SELECT name
CLUSTER PUBLIC *
FROM
  {
  SELECT x-coord
  CLUSTER { x-coord: PUBLIC *}
  FROM
    {
    SELECT time
    CLUSTER { time: PUBLIC *}
    FROM Video
    }
  }
WHERE name CONTAINS 'John' AND name CON-
  TAINS 'Bill'
```

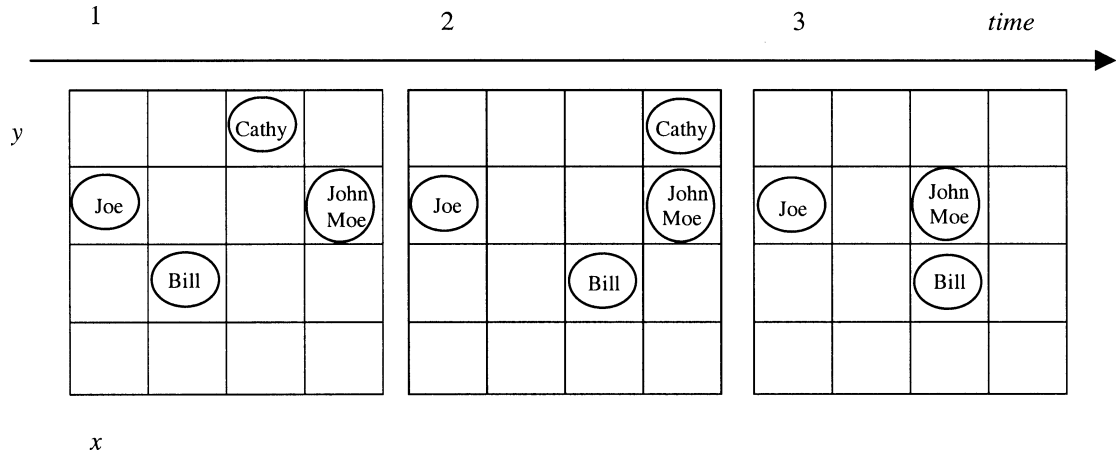


Fig. 11. Simplified video clip.

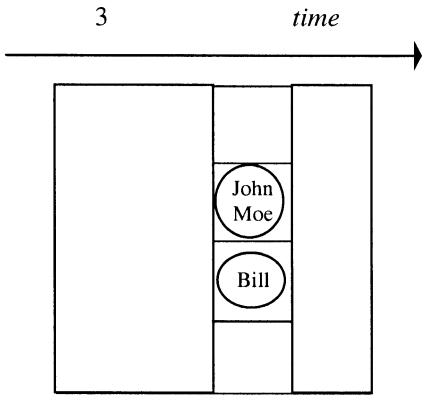


Fig. 12. Extracting all the video frame columns.

The result is given by the following frame column shown in Fig. 12.

Query 2: Extract all the video frame columns containing an entity with name Joe Moe. We want to know who else is in the same frame columns.

```
SELECT name
CLUSTER *
FROM
  {
  SELECT object
  CLUSTER {object: PRIVATE *}
  FROM
    {
    SELECT x-coord
    CLUSTER {x-coord: PUBLIC *}
    FROM
      {
      SELECT time
      CLUSTER {time: PUBLIC *}
      FROM Video
      }
    }
  }
WHERE name CONTAINS 'John' AND name CON-
  TAINS 'Moe'
```

The result is shown in Fig. 13. Note that because of the PUBLIC filtering mode along the dimensions *frame* and *column*, all the frames and columns that are not concerned with the final query have been eliminated. However, because of the filtering mode PRIVATE along the dimension *object*, all the objects have been kept both in the "SELECT object" and "SELECT name" subqueries. The final result shows the required objects in a highlighted form and also (nonhighlighted) others forming their column context.

Let us consider now the hierarchy of dimensions in Fig. 14 for querying the Web.

Let us now consider the following query:

```
SELECT pattern
CLUSTER *
FROM
  {/* subquery 1*/
  SELECT image, url
  CLUSTER {image, url: ALLPRIVATE *}
  FROM
    {/*subquery 2 */
    SELECT page
    CLUSTER { page: PUBLIC *}
    FROM WWW
    }
  }
WHERE pattern CONTAINS 'Bill Clinton'
AND pattern CONTAINS 'dog'
```

Subquery 2 allows the external queries to search one Web page at a time. Because of the PUBLIC filtering mode all the pages that will not match the whole query will be deleted. Subquery 1 will project each page against the dimensions *image* and *url* illustrating all the images and the address of each page. The outer query will then project all the resulting clusters of single images against the dimension *pattern* selecting the images with patterns corresponding to Bill Clinton and a dog. Because of the ALLPRIVATE filtering mode on *image* and *url*, the query will return all the complete web pages containing at least an image with Bill Clinton and a dog, together with their url address.

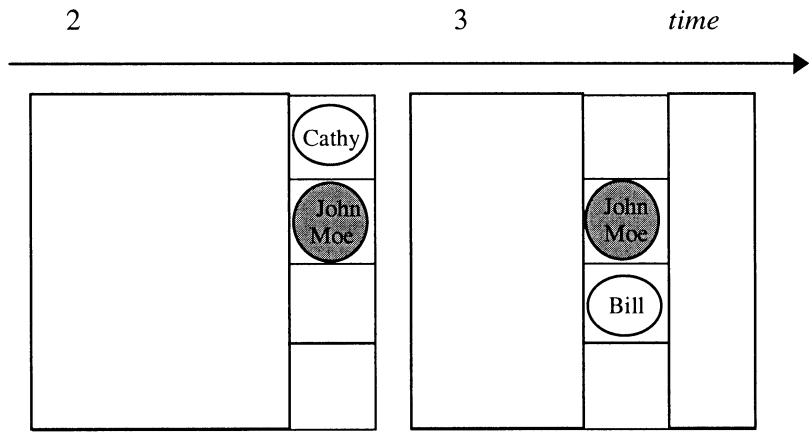


Fig. 13. Extracting all the video frame columns containing an entity with name Joe Moe.

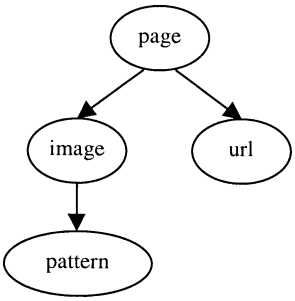


Fig. 14. Possible hierarchy of dimensions for the web.

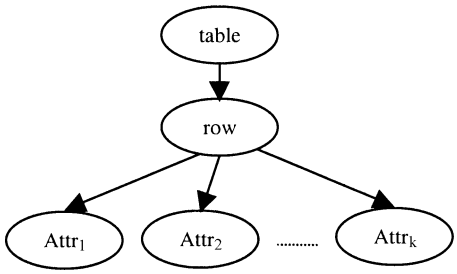


Fig. 15. Dimension hierarchy template for relational databases.

IX. Sensor Data Fusion Using the Merge Operation

In this and the next two sections, several important topics in sensor data fusion, including fusion by the merge operation, the detection of moving objects, and the incorporation of belief values, are discussed in detail. Although we cannot say they are the only topics of interest in sensor data fusion, we found these topics to be very important in order for sensor data fusion to work.

From the database point of view the merge operations are used to fuse information coming from two or more *RSI*s to produce a new *RSI*. There are many possible merge operations and they strictly depend on the particular environment the query has to work in. A simple example of merge operation is the CARTESIAN_MERGE operation defined in the environment of the relational databases. Each operator can be defined on one or more distinct pairs of RSI, but the same operator may not have different implementations for the same pair of RSI. It is important to note that the merge operators may behave differently depending on the clusterings defined in the queries used in the operation. The template of an application of a merge operation is

```
⟨MERGE OPERATOR⟩
⟨FROM⟩
  {select operation [merge operation]}
  ,
  {select operation [merge operation]}
```

Given the hierarchy of dimensions template for relational databases in Fig. 15, and a relational database containing, among other things, a table "Names" with attributes SSN and name containing data about some people. Suppose we want to look for all Web pages containing images about the people whose name is in the database. To do this, we first extract the table "Names" from the database by temporarily storing it in resultTable. This is done by using the following INSERT INTO clause:

```
INSERT INTO resultTable
{
  SELECT Table
  CLUSTER { Table: 'Names'}
  FROM DataBase
}
```

The final result is obtained through the following composed query:

```
MERGE_AND
FROM
  {
  SELECT name
  CLUSTER {name: *}
  FROM resultTable
  },
  {
  SELECT pattern
  CLUSTER {pattern: *}
  FROM
    {
```

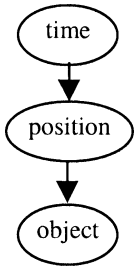


Fig. 16. Dimension hierarchy for VIDEO.

```
SELECT image
CLUSTER {image: *}
FROM
  {
   SELECT url
   CLUSTER {url: PUBLIC *}
   FROM WWW
  }
}
}
```

In this query, the operator MERGE_AND returns a RSI made of the Web documents containing images with the people whose names occur in the name field of the table "Names."

X. Detecting Moving Objects in a Video

Let us consider a data source VIDEO described by an RSI based on the dimension hierarchy in Fig. 16.

This hierarchy does not allows us to write ΣQL queries to detect which objects are moving in a given time interval, though it allows us to select the frames in an interval. To detect movement, we would need to find two frames in the interval containing the same object with high belief value. If the two occurrences are in the same positions then it is likely that the object is still. Otherwise the object is likely to be moving.

In order to identify the moving objects in a VIDEO by using the ΣQL language we need to introduce the concept of *derived attribute*. A derived attribute of an entity stores information produced by the processing of already known information. The derived attributes are calculated by using *transformation methods* that translate an RSI into another. This new RSI describes the same search space but it may be based on a different dimension hierarchy.

As an example, let us consider the following ΣQL query based on the dimension hierarchy of Fig. 16.

```
/* Query 1 */
INSERT INTO video_piece
{
  SELECT time
  CLUSTER {time : PUBLIC * ALIAS vTIME}
  FROM aVIDEO
  WHERE vTIME >= t_in AND vTIME <= t_fin
}
```

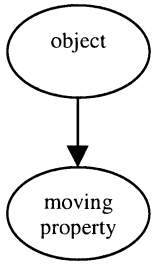


Fig. 17. Hierarchy dimension generated by the show_state algorithm.

Query 1 stores in *video_piece* all the frames of *aVIDEO* in the interval $[t_{in}, t_{fin}]$. In order to detect all the moving objects in such interval we write the following query:

```
/* Query 2 */
SELECT moving_property
CLUSTER *
FROM
  {
  SELECT object
  CLUSTER: show_state: { object:
PUBLIC *}
  FROM video_piece
  }
WHERE moving_property='moving'
```

The inner subquery in Query 2 not only selects all of the objects in the required interval but also builds on them the new dimension *moving_property* by using the transformation method *show_state*. The old RSI is then translated into the new RSI in Fig. 17 that is then used to represent all the objects in *video_piece*. The external subquery in Query 2 is then able to select all the objects with *moving_property*='moving'.

The transformation method *show_state* calculates the *moving_property* attribute value of each object by checking the object coordinates in each frame. In order to give the appropriate values, for each object, it has to calculate the object identity over the frames.

The change of *RSI* in ΣQL is an operation very similar to the creation of a *view* in SQL: indeed, starting with an RSI we build a new RSI by structuring already stored information in a different way. On the other hand the attribute *moving_property*, calculated by the algorithm translating the *RSI*, may be considered as a *derived attribute* in SQL, since it provides information derived by other existing data.

XI. Belief Values

Detecting the object identity over several frames is not an easy task and most of the times it is not possible to state with certainty whether two objects in two different frames are the same. It becomes then necessary to use belief values for derived attributes. In this case we can rewrite Query 2 as follows:

```
/* Query 3 */
SELECT moving_property
CLUSTER *
```

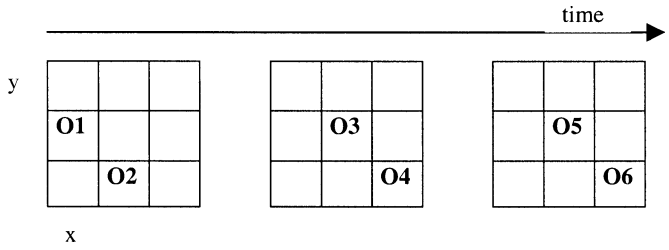


Fig. 18. Example of video_piece.

```
FROM
  {
  SELECT object
  CLUSTER: show_probable_state: { ob-
ject: PUBLIC *}
  FROM video_piece
  }
WHERE moving_property >= 0.9
```

This query selects not only the objects that certainly moved in the required interval in *video_piece* but also the objects that moved with a belief value. As an example, supposing to use the dimension hierarchy of Fig. 18, let us apply Query 3 to the input *video_piece* shown in Fig. 18.

The inner subquery uses the method *show_probable_state* to obtain an *RSI* based on the dimension hierarchy of Fig. 17. The transformation method will use the function *similar* defined as follows:

$$\boldsymbol{similar} : \boldsymbol{object} \times \boldsymbol{object} -> [0,1]$$
$$\text{such as } \boldsymbol{similar}(\boldsymbol{a},\boldsymbol{b}) = \boldsymbol{similar}(\boldsymbol{b},\boldsymbol{a}).$$

This function returns the belief value for the identity of its two object arguments:

$$\boldsymbol{similar}(\boldsymbol{O1},\boldsymbol{O3}) = 0.7$$
$$\boldsymbol{similar}(\boldsymbol{O3},\boldsymbol{O5}) = 1$$
$$\boldsymbol{similar}(\boldsymbol{O1},\boldsymbol{O5}) = 0.7$$
$$\boldsymbol{similar}(\boldsymbol{O2},\boldsymbol{O4}) = 1$$
$$\boldsymbol{similar}(\boldsymbol{O4},\boldsymbol{O6}) = 0.9$$
$$\boldsymbol{similar}(\boldsymbol{O2},\boldsymbol{O6}) = 0.9$$

(for all the other combinations the belief value is zero).

Due to the execution of *show_probable_state*, the objects in *video_piece* are represented as shown in Fig. 19.

The probability that O_1 moved is 0.7 since O_1 is the same as O_3 and O_5, and it has a different x-coordinate. O_2 and O_4 are occurrences of the same object with belief value equal to 1 and have different positions, then we can state that the corresponding object moved. O_3 and O_5 are occurrences of the same object and have the same position in the frames they appear, O_3 (resp. O_5) is the same as O_1 with belief value 0.7 and its x-coordinate is different from that of O_1, then it moved with belief value 0.7. O_6 and O_2 (resp. O_4) are the same object with belief value 0.9 and have different x-coordinates, then O_6 moved with belief value 0.9. The external part of Query 3 deletes all the objects with belief value less than 0.9.

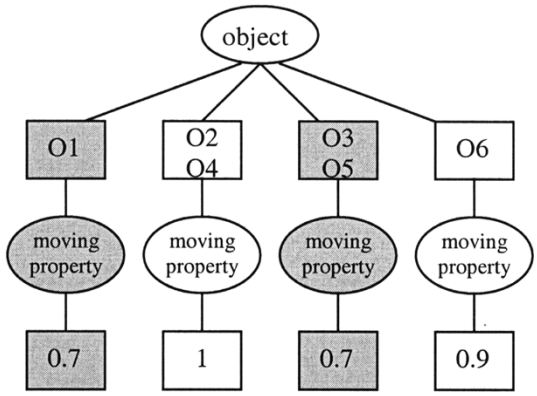


Fig. 19. Output of Query 3 (parts colored in gray are not to be considered).

As an example, let us now provide a ΣQL query translation for the σ-query discussed in Section V. We want to retrieve all the moving vehicles detected simultaneously by a laser radar and a video camera. The query is made of three parts. The fist part inserts into *vehicles* all the objects identified as vehicles by first clustering the data from a laser radar along the time dimension and then clustering the result along the object_3d dimension in the time interval between t_in and t_out.

```
INSERT INTO vehicles
{
SELECT type
CLUSTER *
FROM
  {
  SELECT object_3d
  CLUSTER {object_3d: PUBLIC *}
  FROM
    {
    SELECT time
    CLUSTER {time: PUBLIC * ALIAS T}
    FROM laser_radar
    WHERE T > t_in AND T < t_fin
    }
  }
WHERE type = 'vehicle'
}
```

In the second part, the inner subquery selects a video frame every ten frames in the same interval t_in and t_out. From each of these frames all the objects are retrieved and, successively, with a change of representation, the *moving* status of each object is made explicit, i.e., the attribute *status* is added to each object. Again, an operator such as show_state must exists in the Meta-Database in order to make the recognition of a moving object possible. Finally, all the resulting moving objects are inserted in *moving_objects*.

```
INSERT INTO moving_objects
{
SELECT state
CLUSTER *
FROM
```

```
    {
    SELECT object
    CLUSTER: show_state: {object: PUBLIC
  *}
    FROM
    {
    SELECT time
    CLUSTER {time: PUBLIC * ALIAS T}
    FROM video
    WHERE T mod 10 = 0 AND T > t_in AND T <
  t_fin
    }
  }
WHERE state ='moving'
}
```

The last part of the query simply applies the MERGE_AND construct to the results of the previous parts. Such operation will finally retrieve only the moving vehicles detected by both the laser radar and the video camera in the same time interval.

MERGE_AND

FROM vehicles, moving_object

This application of MERGE_AND is legal since *vehicles* stores a set of objects with attributes type, time, x, y, and z, while x, y and characterize the position of the object in the space. On the other hand, *moving_objects* stores a set of objects with attributes state, time, x and y. The attributes involved that can be *joined* are then time, x and y. It should be noted that the dimensions and directions, occurring in the original σ-query, are not considered here for the sake of clarity.

XII. Discussion

In this paper, we described how to carry out sensor data fusion from multiple sensors. In our approach, the queries are manually created, and then modified, to deal with the lack of information from a certain source or sources, and therefore not only the constraints can be changed, but also the source(s).

An experimental ΣQL query processing system has been implemented by researchers at the University of Pittsburgh, the University of Salerno, and the Swedish Defence Research Agency, to demonstrate the feasibility of applying the proposed techniques to data from three types of sensors, including laser radar, infrared video (similar to video but generated at 60 frames/s) and CCD digital camera. The users have successfully tested a number of queries, ranging from simple queries to complex ones for fusion, and systematic usability study is currently being conducted. Having established the feasibility of the techniques, we now discuss a number of issues for further research.

The sensors in the above experiment are limited to the three prespecified types of image sensors. To handle a large number of different sensors, we propose the following extension [5], [24]: the characteristics, applicable ranges, and processing algorithms of these sensors are stored in a knowledge base, which enables the system to deal with new sensors. The incorporation of domain-specific information into the knowledge base makes this approach extendible to other multimedia applications.

The fusion method is based on a method that is replaceable by other methods. Examples of other fusion methods that can be used are Basian networks [10] and Dempster–Schafer [23]. The proposed information structure is an information flow structure that works in parallel with the queries and allows acquisition and determination of the information necessary to carry out the sensor data fusion process. It is not only necessary to determine the objects, their state variables, and attributes that are requested by the query but also the belief values associated to them. This will put a heavy burden on the user to judge the result of the queries with respect to the belief values returned by the query system based on the uncertainty of the sensor information, because there will always be uncertainties in data registered by any sensor. How to replace the manual query refinement process by a semi-automatic or fully automatic query refinement process is of great importance from a user's point of view and will be further investigated.

Regarding the issue of generality of the ΣQL language, it is at least as powerful as SQL because an SQL query can be regarded as an ΣQL query with the clause "CLUSTER$*$." Since ΣQL can express both spatial and temporal constraints individually using the SELECT/CLUSTER construct and nested subqueries, and sensor data sources are by nature spatial/temporal, there is a good fit. Its limitation is that constraints simultaneously involving space and time cannot be easily expressed, unless embedded in the WHERE clause. Although such constraints may be infrequent in practical applications, further investigation is needed in order to deal with such complex constraints.

Finally, the qualitative methods used by the σ-operators are developed to support indexing and efficient inference making by transforming the information acquired from the heterogeneous data sources into a unified spatial/temporal structure. Such a unified structure is desirable because generic reasoning techniques can be applied independently of the original sensor data structures. Thus, generic σ-operators based on qualitative methods can be designed and implemented to support qualitative structure such as Symbolic Projection, which is discussed further in [4] where a number of alternative qualitative approaches can also be found.

References

[1] J. Baumann *et al.*, "Mole—concepts of a mobile agent system," *World Wide Web*, vol. 1, no. 3, pp. 123–137, 1998.

[2] C. Baumer, "Grasshopper—a universal agent platform based on MASIF and FIPA standards," in *First Int. Workshop on Mobile Agents for Telecommunication Applications (MATA'99)*, Ottawa, ON, Canada, Oct. 1999, pp. 1–18.

[3] K. Chakrabarti, K. Porkaew, and S. Mehrotra, "Efficient query refinement in multimedia databases," in *16th Int. Conf. Data Engineering*, San Diego, CA, Feb. 28–Mar. 3, 2000.

[4] S. K. Chang and E. Jungert, *Symbolic Projection for Image Information Retrieval and Spatial Reasoning*. London, U.K.: Academic, 1996.

[5] S. K. Chang, G. Costagliola, and E. Jungert, "Multi-sensor information fusion by query refinement," in *Proc. 5th Int. Conf. Visual Information Systems (Visual'02)*, Hsinchu, Taiwan, R.O.C., Mar. 2002.

[6] S. K. Chang and E. Jungert, "A spatial/temporal query language for multiple data sources in a heterogeneous information system environment," *Int. J. Cooperative Inform. Syst. (IJCIS)*, vol. 7, no. 2 & 3, pp. 167–186, 1998.

[7] C.-Y. Chong, S. Mori, K.-C. Chang, and W. H. Baker, "Architectures and algorithms for track association and fusion," in *Proc. Fusion'99*, Sunnyvale, CA, July 6–8, 1999, pp. 239–246.

[8] G. Grafe, "Query evaluation techniques for large databases," *ACM Comput. Surv.*, vol. 25, no. 2, June 1993.

[9] F. V. Jensen, *An Introduction to Bayesian Networks*. New York: Springer Verlag, 1996.
[10] E. Jungert, "A data fusion concept for a query language for multiple data sources," in *Proc. 3rd Int. Conf. Information Fusion (FUSION 2000)*, Paris, France, July 10–13, 2000.
[11] ——, "A qualitative approach to reasoning about objects in motion based on symbolic projection," in *Proc. Conf. Multimedia Databases and Image Communication (MDIC'99)*, Salerno, Italy, Oct. 4–5, 1999.
[12] ——, "An information fusion system for object classification and decision support using multiple heterogeneous data sources," in *Proc. 2nd Int. Conf. Information Fusion (Fusion'99)*, Sunnyvale, CA, U.S.A, July 6–8, 1999.
[13] E. Jungert, U. Söderman, S. Ahlberg, P. Hörling, F. Lantz, and G. Neider, "Generation of high resolution terrain elevation models for synthetic environments using laser-radar data," *Proc. SPIE, Model., Simul. Visualiz. Real Virtual Environ.*, vol. 3694, , pp. 12–20, April 7–8, 1999.
[14] L. A. Klein, "A boolean algebra approach to multiple sensor voting fusion," *IEEE Trans. Aerosp. Electron. Syst.*, vol. 29, no. 2, pp. 317–327, Apr. 1993.
[15] H. Kosch, M. Doller, and L. Boszormenyi, "Content-based indexing and retrieval supported by mobile agent technology," in *Multimedia Databases and Image Communication*, M. Tucci, Ed. Berlin, Germany: Springer-Verlag, 2001, pp. 152–166.
[16] D. B. Lange and M. Oshima, *Programming and Deploying Java Mobile Agents with Aglets*. Reading, MA: Addison-Wesley, 1999.
[17] S. Y. Lee and F. J. Hsu, "Spatial reasoning and similarity retrieval of images using 2D C-string knowledge representation," *Pattern Recognit.*, vol. 25, no. 3, pp. 305–318, 1992.
[18] J. R. Parker, "Multiple sensors, voting methods and target value analysis," *Proc. SPIE Signal Processing, Sensor Fusion and Target Recognition VI,*, vol. 3720, pp. 330–335, April 1999.
[19] M. Stonebraker, "Implementation of integrity constraints and views by query modification," *SIGMOD*, 1975 PAGES???.
[20] B. Vélez, R. Weiss, M. A. Sheldon, and D. K. Gifford, "Fast and effective query refinement," in *Proc. 20th ACM Conf. Research and Development in Information Retrieval (SIGIR97)*, Philadelphia, PA, July 1997.
[21] E. Waltz and J. Llinas, *Multisensor Data Fusion*. Boston, MA: Artech House, 1990.
[22] F. E. White, "Managing data fusion systems in joint and coalition warfare," in *Proc. EuroFusion98—Int. Conf. Data Fusion*, Great Malvern, U.K., Oct. 1998, pp. 49–52.
[23] F. E. Yager, F. E. Fedrizzi, and F. E. Kacprzyk, Eds., *Advances in Dempster-Shafer Theory of Evidence*. New York: Wiley, 1994.
[24] S. K. Chang, G. Costagliola, and E. Jungert, "Multi-sensor information fusion by query refinement," in *Lecture Notes in Computer Science LNCS*. Heidelberg, Germany: Springer-Verlag, Mar. 2002, pp. 1–11.
[25] T. Horney, E. Jungert, and M. Folkesson, "An ontology controlled data fusion process for a query language," in *Proc. 6th Int. Conf. Information Fusion*. Cairns, Australia, July. 8-11, 2003, pp. 530–537.

Shi-Kuo Chang (F'86) received the B.S. degree from National Taiwan University, Taipei, Taiwan, R.O.C., in 1965, and the M.S. and Ph.D. degrees from the University of California, Berkeley, in 1967 and 1969, respectively.

He was a Research Scientist at the IBM T. J. Watson Research Center, Yorktown Heights, NY, from 1969 to 1975. From 1975 to 1982, he was Associate Professor and then Professor with the Department of Information Engineering, University of Illinois at Chicago. From 1982 to 1986, he was Professor and Chairman of the Department of Electrical and Computer Engineering, Illinois Institute of Technology, Chicago. From 1986 to 1991, he was Professor and Chairman of the Department of Computer Science, University of Pittsburgh, Pittsburgh, PA. He is currently Professor and Director of Center for Parallel, Distributed and Intelligent Systems, University of Pittsburgh. His research interests include distributed systems, image information systems, visual languages, and multimedia communications. He has published over 240 and wrote or edited 14 books. His books *Principles of Pictorial Information Systems Design* (Englewood Cliffs, NJ: Prentice-Hall, 1989), *Principles of Visual Programming Systems* (Englewood Cliffs, NJ: Prentice-Hall, 1990), *Symbolic Projection for Image Information Retrieval and Spatial Reasoning* (New York: Academic, 1966), and *Multimedia Software Engineering* (Norwell, MA: Kluwer, 2000), are pioneering advanced textbooks in these research areas. He is the Editor-in-Chief of the *Journal of Visual Languages and Computing* published by Academic Press, and the Editor-in-Chief of the *International Journal of Software Engineering & Knowledge Engineering* published by World Scientific Press.

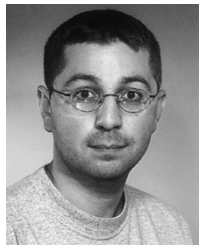
Gennaro Costagliola (M'95) received the Laurea degree in computer science from the University of Salerno, Italy, in 1987, and the M.S. degree in computer science from the University of Pittsburgh, Pittsburgh, PA, in 1991.

He is currently Professor and Director of the Laurea degree courses in computer science at the University of Salerno. His research interests include programming languages, visual languages, parsing technologies, multimedia databases, web technologies.

Dr. Costagliola was guest coeditor of the February 2002 Special Issue of on Querying Multiple Data Sources for the *Journal of Visual Languages and Computing*. He is a member of the ACM and the IEEE Computer Society.

Erland Jungert received the Ph.D. degree in computer science from the University of Linköping, Linköping, Sweden, in 1980.

He has been a Research Staff Member at the Swedish Defence Research Agency (FOI), Linköping, since 1980, except for a short period in 1985–1986, when he was a Visiting Professor at the Illinois Institute of Technology, Chicago. Since 1987, he has been Director of computer science research at FOI and, since 1997, he is also a part-time professor at the Computer Science Department, University of Linköping. His current research interests include query languages, methods for qualitative spatial reasoning and various aspects of GIS. He is a coauthor of one book on spatial reasoning and a co-editor of two other books on visual languages and spatial reasoning. He is also an associate editor of the *Journal of Visual Languages and Computing*.



Francesco Orciuoli was born in Salerno, Italy, on April 6, 1975. He received the Laurea degree in computer science (cum laude) from the University of Salerno, Italy, in 1999.

He is a Research Contractor with the Department of Computer Engineering and Applied Mathematics (DIIMA), University of Salerno), in the area of distributed object infrastructures. His fields of interest are component-based design, compiler building techniques, design patterns, object oriented analysis, design and implementation, relational databases, web services, and UML.