

An Architecture for Interactive Query Refinement in Sensor-based Information Fusion Systems

G. CASELLA*, S. K. CHANG**, G. COSTAGLIOLA*, E. JUNGERT***, X. LI** and T. HORNEY**

*Dept. Informatica e Matematica, University of Salerno, ITALY {giocas, gencos}@unisa.it

**Dept. of Computer Science, University of Pittsburgh, USA {chang, flying}@cs.pitt.edu

*** Swedish Defence Research Agency (FOI), Sweden {jungert, tobho}@foi.se

Abstract

An architecture for query execution and interactive query refinement on heterogeneous data from spatially distributed sensors is proposed. The objective is to define a system in which it is always possible to add new data sources and new algorithms for data processing and user query execution. The query language that forms the basis for query formulation is called Σ QL. Ambiguous and vague query can be interactively refined as well. An ontological knowledge base to model fundamental architecture components and their relations is introduced. The method to choose algorithms that can execute user queries based on available data, the Pequiar interactive system, the Reasoner and the user interface are described. The Reasoner automatically or semi automatically carries out an iterative information fusion step by means of rules that are either automatically launched by the Reasoner or defined by the user. If a rule is defined by the user the Reasoner can learn the new rule and use it in later queries. Fusion query examples are also presented. User interface enables the user to compose compound queries from elementary queries using query operators, presents the query results and gives the user a qualitative evaluation of the results.

1. Introduction

In this paper we define an architecture for query execution and interactive query refinement on heterogeneous data from spatially distributed sensors.

The objective of this work is to define a system in which it is always possible to add new data sources and new algorithms for data processing and user query execution. Ambiguous and vague query can be interactively refined as well. The query language that forms the basis for this is called Σ QL and is described in [3],[4].

For example a possible query is to find an object (such as a car) in an area or detect particular events (such as a region covered by a flood). Generally speaking, to

formulate a query such as: “Find cars in a region 5 hours before the region is covered by a flood”, the user needs to specify:

1. the objects to be recognized (cars and flood)
2. the area of interest (the region)
3. the time of interval (when)
4. the relations of interest among the objects (“Find cars in a region” and “5 hours before the region is covered by a flood”)
5. the output format (the retrieved cars and their attributes, or data showing only the retrieved cars, or showing only the cars and the flood, or showing the complete context and so on).

In order to specify items 1 and 2 of a query, the system has to provide the user with maps to let the user indicate the area and predefined objects to be recognized. Furthermore the system should enable the user to define new objects. Note that in other queries one might want to retrieve the areas where something occurs. In this case, the area of interest is given by all the maps known to the system. Maps are stored in a database to support the user interface and objects are described in the ontological knowledge base [1],[6],[7].

The user can interact graphically with the objects and the maps, and provide a time interval (similarly to the case of the sentient map [2]) and all the other information needed to formulate the query. In certain cases this could be done in a query-by-demonstration way or by adding text annotations.

The user interface should give the feedback to the user to let the user know whether the defined query is admissible at that moment or during some later time period, or it cannot be executed until other sensors and/or algorithms are added to the system. To do this the user interface needs to communicate with the ontological knowledge base. The output format can be defined by tagging the elements of interest as “to be presented in the output” or, in more complex cases, using some kind of authoring tool. Once the initial query has been specified, the system needs to complete it with the missing “technical” information such as:

1. what are the characteristics of the objects needed to satisfy the relations (i.e. the query) ?
2. what sensors can be used in that area at that time to recognize these characteristics optimally?
3. what algorithms can be used with the selected sensors for the particular use?

In the above mentioned query, the characteristics of interest to satisfy the relation: (“Find cars in a region” and “5 hours before the region is covered by a flood”) are Position (area covered) and Time for both the car object and the flood object. Hence the relation becomes:

Position_car covered_by Position_flood AND
Timecar= Timeflood -5h

The question now is whether point 1 can be answered automatically. In other words, will the system need to infer the characteristics of interest, or will the user need to specify the characteristics by expressing the relations in more detail? A possible intermediate solution is to add to the ontological knowledge base all the relations of interest in a certain domain. The user has the possibility of browsing and defining the object characteristics from their ontological description while creating the query or, at a higher level, selecting predefined relations between objects from the ontological knowledge base, and the definition of the relation in OKB tells the system what are the values of the characteristics of interest.

Furthermore, once all the information from the user and the system has been gathered a complete Σ QL query can be built. The query processing phase proceeds then in two separate parts:

1. Low level query processing: to recognize all the objects and properties needed to run the high level query.
2. High level query processing: to provide the final result by extracting only the information satisfying the required relations among objects.

In the first part, given the objects, their characteristics, their spatial and temporal position and all the knowledge coming from the ontological knowledge base about objects, sensors, algorithms and conditions, the system is now able to build an initial plan of sensors-algorithms applications. The aim of this first phase is to use, in the best way, all the data sources and algorithms available to execute a user query. Crucial points are the correct management of available data sources, the choice of the most suitable algorithms for query execution and a good presentation of the results.

The execution of the plan on the real data might modify the plan itself in order to get more precise results. The plan uses sensor data fusion [5] operations to make all the data converge to a complete final result. In the second part the query is run against the recognized items. An interactive system called Peculiar is used to resolve the ambiguity in the high level queries.

According to this approach the Σ QL query is seen as logically divided into two distinct parts: the modifying plan and the relational constraints (WHERE clause).

Alternatively, the Σ QL query processing phase might take care simultaneously of the sensor information and of the relational constraints. However the query will have to change during its execution because of the plan adaptation. This makes the whole system much more complex and less structured.

In our approach we consider two kinds of algorithms. The first ones are recognition algorithms and their goal is to detect something from available data. The second ones are fusion algorithms, their goal is to fuse recognition algorithm results or other fusion algorithm results to obtain more information. We consider the case in which more than one algorithm is suitable to execute the user query and we present a method to choose the best.

Finally we describe guidelines to create a user interface to help a user during query definition and result interpretation.

In section 2 we present an ontological knowledge base to model fundamental architecture components and their relation. In section 3 we describe a method to choose algorithms that can execute user queries based on available data. In section 4 we illustrate and briefly describe the system architecture. The Reasoner and reasoning approaches are discussed in Section 5. Section 6 presents some fusion query examples. In section 7 we describe the user interface and finally in Section 8 we give the conclusions.

2. The Ontological knowledge base

The ontology is used to model fundamental architecture components and their relations. A similar approach is presented in [6],[7]. All concepts used in the architecture with their properties and relations are modelled in the ontology.

In this section all components are described:

Data Source

The concept “Data Source” models a generic entity that can provide data, for example a database containing information collected from a sensor placed on the territory. A Data Source has a position, expressed in global coordinates (for example longitude and latitude), that points the place where the sensor is. Data Source has also an active range that represents the area where data are acquired and is expressed in relation of the data source position. Data source produces a sequence of Perception Source (see Figure 1), which corresponds to a certain sensor data type such as CCD, IR, LASER. An algorithm may input only a certain sensor data type, or a set of data type, and so can work only with data sources that provide such type of data.

Context Source

The background or proper context in which an object may occur and that is subject to a query in Σ QL can be seen as a data source used to enhance the outcome of the query. This was similarly discussed in [3]where the

“local view” and the “global view” were introduced as means for query refinement. Here the two views are jointly called Context Source. An illustration of the use of Context Source to enhance a query result is given in the type I task query example where basically the Global View is used to determine whether there are any objects in the proximity of an already retrieved object. As the number of types of queries extends other sources are required as well, which is illustrated in Figure 1.

Query input

Query input includes information delivered by the users including such information as area-of-interest (AOI), (time) interval of interest and requested object types. In this source, AOI may be used to determine the extension of the context source. Second to this the Meta Data source is used to determine whether there are any data available from any source in Perception Source that correspond to AOI and where Perception Source corresponds to, e.g. data from a sensor. The latter type of sources can be seen as primary sources that are always used as query input.

Dependency Tree

The Dependency Tree that is generated internally to each query can also be used as a data source in cases where queries about query result are of concern. This is illustrated by the task III query type that asks for pre-fusion sub-results that may be of extreme type.

Query Output

The result of any given query can be used in an iterative query; this is illustrated by the task I query type. Altogether, the concept of Data Source becomes much more complex when applied to iterative queries generated either in dialogue with or automatically by Pequiar. Although this may cause difficulties in composing iterative queries it also extends the number of possible queries, which in turn, makes Σ QL more adaptive to complex problems related to, for instance, higher orders of information fusion. That is, in applications like situation and impact analysis and for this reason it becomes possible to talk about context sensitive object assessment where context refers not just to the proper context but to all the data sources given in Figure 1.

Perception Source

This concept models data type that a sensor produces. It is introduced to associate data sources and algorithms to process data; a data source outputs a particular sensor-data-type, and an algorithm can work on a particular data type (or a set of them) in input. A Perception Source has a name and other additional information. As additional information we consider, for example minimal and maximal rate of data sources that provide this data type. Examples of Sensor Data Type are IR, CCD, LASER, TEXT, etc.

Data-Unit

This concept models a single data unit. A Data Unit has a type “Perception Source” that describes what kind of sensor has produced it, and a timestamp that shows when it has been produced. Timestamp can be expressed in relation to GMT.

Algorithm

The algorithms have a name that identify them unambiguously and are of two types: “recognition algorithm” or “fusion algorithm”.

Recognition Algorithm

The aim of the recognition algorithms is to detect one or more ‘characteristics’ in the data types to be processed. The data type (or the set) in input is a Perception Source and is produced from a “Data Source”. By the term “characteristic” we mean, for example, characteristic-shape (a car), characteristic-color (red object), characteristic-motion (objects that move), characteristic-crash (objects that crash), characteristic-text (a particular sentence in a document or in a image), etc. Every characteristic belongs to the concept “Recognizable Characteristic”.

Fusion Algorithm

The aim of a fusion algorithm is to fuse information produced by other algorithms (fusion and / or recognition algorithms) to obtain better results. Fusion algorithm’s input is then the output of a set of algorithms. As recognition algorithms, fusion algorithms have a set of “Recognizable Characteristics” that represent what the algorithm can recognize from the input data.

Recognizable Characteristics

The introduction of the extended set of data sources enables the system to answer queries about proper-background and similarity. Proper background is a construct, which can be used to ask the system if a certain object that has been found is present in a proper background, e.g. a bus is properly situated on a road but it is not proper to find it in the middle of a lake. Similarity, in this case, means similarity between objects, i.e. is a given object type similar to another object type. For example, is a car similar to a vehicle or a bus similar to a building?

This concept of recognizable characteristics models the set of characteristics (or events) that the data fusion and recognition algorithms can detect. A particular recognizable characteristic is the “object relationship”. For example the system can detect “two objects that crash” or if some objects are proper in which backgrounds (proper-background) and which objects are similar to which other objects (similarity).

Objects that can be searched for and background types in the ontology, allow us to define object relations such as is-proper-in-background between “Recognizable Object” (the base concept for objects that can be

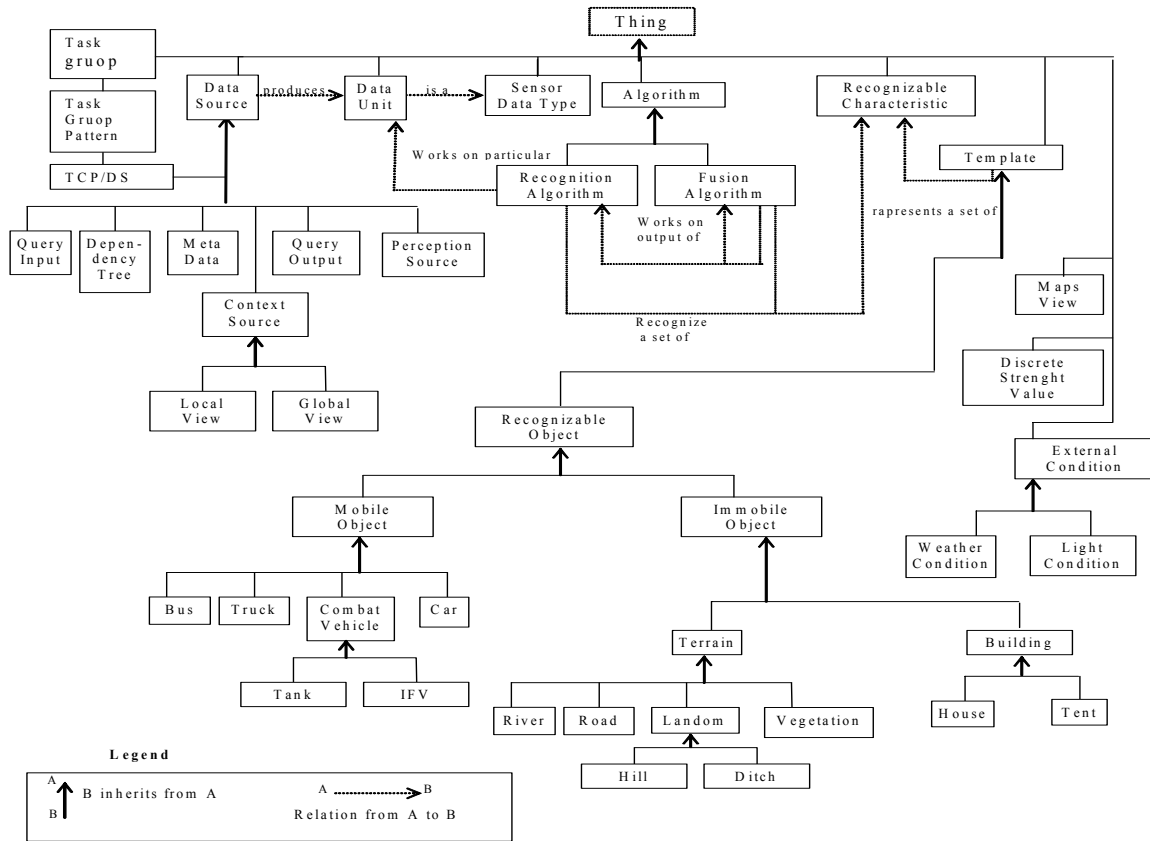


Figure 1. The ontological knowledge base

searched for) and “Terrain” (the base concept for backgrounds). Now, for each “Recognizable Object” an instance of the proper-in-background relation is created to each Terrain concept describing a background where it is proper to find it. It is, of course, possible to do the reverse, i.e. define a relation is-NOT-proper-in-background instead and create instances of that relation between “Recognizable Object” instances and Terrain instances where the object is not proper in the background. With either definition, it is now easy for the system to check if it is proper to find a certain object in the background where it was found.

Template

This concept models a concrete example of something that a user can be interested in. It has a set of “Recognizable Characteristic” (for example if the template represents a car, the recognizable characteristic is the shape, if the template represents “a red car that is moving”, the recognizable characteristics are the shape-color-motion). A source file or other information can be associated to a template, for example “car-shape.jpg” can be associated to a car shape or RGB(FF,00,FF) to a color. A User interface will use templates to help a user in formulating the queries.

Recognizable Object

This concept model the objects that can be recognized by the recognition algorithms. Recognizable objects further divided into mobile and immobile objects.

External Condition

This concept models the external conditions in which a sensor has collected data. External conditions are “Weather Condition” and “Light Condition” and have values belonging to the “Discrete Strenght Value” concept.

Maps View

This concept models all the information required from the user interface to enable a user to select the territory of interest.

Figure 1 shows a graphical representation of the ontology modeled in a hierarchical manner known as ontology tree. The hierarchy has the ultimately general concept called thing at the top. All other concepts inherit directly or indirectly from thing. This means that “everything is a thing”. The concept thing has no properties and no relations; it just acts as the parent of all other concepts. The hierarchy is organized so that more specialized concepts appear further down the inheritance chain.

When the ontological structure has been created it is populated with instances becoming a knowledge base called ontological knowledge base.

3. The Query Execution Planner

The aim of the query execution planner is to choose the more suitable algorithms, available to the system, to execute user queries in relation to available data. When data-sources and algorithms for data processing have been chosen a representation of this choice is transferred to the Σ QL query builder. To describe the query execution planner we use a wide hierarchical structure given in [3]. We represent the query execution process with a tree for every area with data. A single node is given in Figure 2.

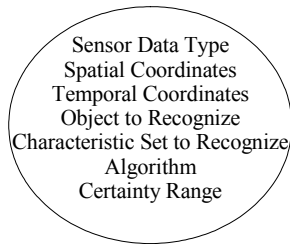


Figure 2. A node in sensor dependency tree.

The term “Sensor Data Type” represents the type of data against which we execute the query. The term “Spatial Coordinates” and “Temporal Coordinates” refer to the area and time in which data are retrieved. The term “Object to Recognize” represents the object (or the event) that we want to recognize and the term “Characteristic Set to Recognize” represents the characteristics that we need to recognize to detect the “Object to Recognize”. An arrow from node A to node B denotes that the output of A is the input of B. We present the steps to build the tree, called sensor dependency tree, reminding that all the information about entities (algorithm, sensors, external conditions) come from the ontological knowledge base modeling and storing them.

Step 1: The system analyzes a user query to create tree leaves. Every leaf represents a Data-Source that can provide data about the area and the time interval to which the user is interested. In particular, every leaf represents a Data-Source with an active range that intersects the area of interest for the user and that can provide Data-Units with a timestamp that belongs to the time interval to which the user is interested.

Step 2: The system processes a user query and extracts the characteristics to be recognized to execute the query. In relation to these characteristics the system tries to find an available algorithm that is able to detect these characteristics and so to execute the query. If there exists a unique algorithm to execute the query the system transfers all the necessary information to the Σ -Query Plan Builder to execute it.

The aim of the Σ -Query Plan Builder is to build a Σ QL query in relation to the available data and

algorithms as presented in [4]. If there exists more than a pair (algorithm, data source) available to execute the query, the system chooses the one from which the best results are attended on the base of the algorithm certainty range and external condition in which data are retrieved. An approach to make this choice is presented in [6]. If there is not any pair (algorithm, sensor) to execute the query the system executes step 3.

Step 3: The system considers all the possible user query sub-sets and for every subset it tries to find an algorithm to execute in relation to the available data. For example if a user is interested in finding “all red cars in an area” a subset of this query is to “find all cars”, another subset is to find “all red objects”. For every subset if there exists more than a pair (algorithm, data source) available to execute the query the system chooses the one from which the best results are attended. For every partial result obtained the system tries to find a fusion algorithm that is able to improve that partial result. Improving results means to execute the most part of the user query and then to recognize the largest number of characteristics.

The system tries to improve results until the user query is totally executed or there are not other available algorithms. Finally the system transfers all the necessary information to the Σ -Query Plan Builder to execute the queries (or partial queries).

Here we present a short example in which we suppose that the user is interested to find all “red cars that are moving” in a particular area and in a particular time interval. In this query “Recognizable Characteristic” are Shape (car), Color (red) and Property – Movement (moving object). Furthermore we suppose that the user is interested in an area called “locality” and in a particular time interval called “period”. Considering a given Data Sources and Algorithm sets available we have the sensor dependency tree in figure 3.

Step 1: To find an object located in *locality* in the time interval *period* considered, there are available data from three perception sources, i.e. in this case sensor types. The perception sources are IR, CCD and Laser.

Step 2: To find a particular moving of a given color in the system there are no algorithms that can work on the available data source types.

Step 3: In the system there exists an algorithm, “alg 123”, that can detect a particular shape in data provided from a LASER Data Source. In the system there exists an algorithm, “alg 185”, which can detect a colored shape in data provided from a CCD. Finally there exists a fusion algorithm, “alg 13”, to execute user query fusing output of recognition algorithm 123 and 185.

If the system can not execute the user query totally the plain builder creates more than one tree. Every tree represents the execution of a query subset.

In the following example we suppose that the system does not possess any fusion algorithm to improve the results of the recognition algorithms. Then we have the sensor dependency tree of figure 4. In this case the system can only find cars and red moving objects in

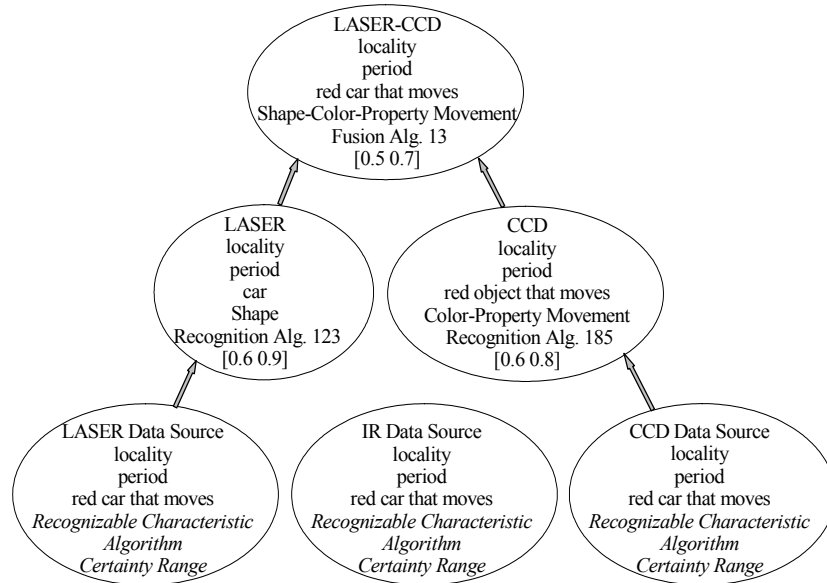


Figure 3. A sensor dependency tree

locality. The Output Manager presents these partial results to the user.

4. The Architecture of the Σ QL System

In this section we present the architecture to define and execute queries on heterogeneous data. Every piece of information about the sensors, available data sources and the algorithms are stored in the ontological knowledge base. The ontology also stores object instances in which the user can be interested, the user interface presents these instances as templates. A user defines queries through this interface. The user interface uses an Input Manager and an Output Manager to help the user in query definition and to present his/her results.

The Input Manager uses the Map Multilevel View Manager to show the user a representation of the territory in which he/she can define a query. When a query has been defined it is transferred to the Query Execution Planner. The Query Execution Planner creates the sensor dependency tree(s) as described above.

If the query cannot be executed in its entirety the system calculates what is the best part of the query that can be executed and notifies this part to the user. Using the Query Planner results the Σ -Query Plan Builder builds the Σ -query or queries using the Σ QL language and then the Query Executor execute/s it/them.

The result of a Σ QL-query is generally the object types requested by the user including also the recognizable characteristic such as attribute and status values of these objects. Example of attributes are color, size etc. while the status of an object generally corresponds to such characteristics as position, orientation or speed etc..

The difference between an attribute and a status value is basically that an attribute is not subject to change in the short range of time, that is, the color of a vehicle may change but not within the time frame of concern to the user. Status values may change within a very short time frame that may be less than seconds; consider for instance position and speed.

Common to all the information returned by Σ QL is that the type attribute is associated with a belief value. Other attributes and status values may have belief values as well but this is less common. For the most part, such belief values are given to indicate to what extent the result of a query can be believed. In the most general case the belief values are just given for the object types and from each type of sensor data and eventually there is also a belief value given as a result of the fusion process that takes place for the majority of the queries; this is due to the use of multiple sensor data sources. Cases when fusion is excluded may occur just for simple and trivial queries. Other information from the query processor that might be of concern for the Reasoner are for instance the quality of the data in the area of interest given by the user.

To determine the source data quality for a certain area of interest the corresponding meta-data will be required. This must, however, be subject to further research.

Updated, and hopefully more informative, belief values will be achieved through a reasoning step in Pequilar that includes the generation of new and more elaborate queries that will be executed subsequently.

Input to this reasoning step is mainly the output, that may include the dependency tree information, from the query processor, the context information and metadata (see also the data sources in Figure 1). Secondary to this is the applicable ontological information. The meta-data is

used to select the portion of the context information that corresponds to the area of interest (AOI). Once the

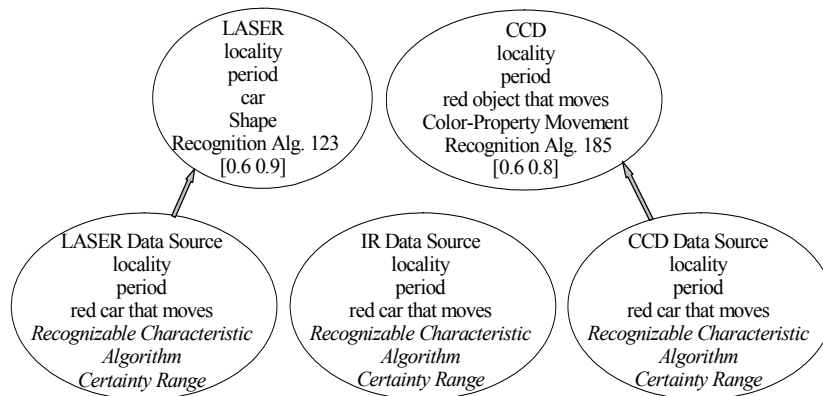


Figure 4. Processing steps.

Reasoner has come to a conclusion in its process a new and elaborate query is created and executed; thus quite often producing an adjusted belief value that better mirrors the situation in focus, that is a more informative belief value may have been achieved.

The query results (partials or totals) and their quality are managed by the Output Manager that present them to the user. Result quality is given based on relation to the original user query, the certainty range of the algorithms used and the external conditions. The output manager gives also to the user the chance to view data from which query results have been obtained.

5. The Reasoner

The Reasoner accepts the output from the Query Processor, and either selects a reasoning rule by itself or by input from the user. The output of the query processor is a collection of entities that are the results of query processing, such as “trucks” recognized by the Query Processor. The Reasoner selects an applicable rule from the following space S , which is the Cartesian product of the sub-spaces including sources, objects to be recognized, attributes of objects, time, location and spatial/temporal/semantic relations. In other words, $S = Source \times Object \times Attributes \times Time \times Location \times Relations$

For example, the Reasoner may need to pick a rule that is applicable to a different source, to recognize a certain type of objects with attributes in a certain range, in a certain time interval, for objects in a certain spatial location, and satisfying certain relations. The Reasoner searches the rule base to select an applicable rule. The rule could be a query template, which is then substantiated and sent back to the Query Processor. If the Reasoner cannot select a rule by itself, either because the rule base is not yet populated or because the space S is not properly defined, the Reasoner can accept input from

the user. The next time, such rules constructed by the user is remembered, forming one part of the scenario.

Pequiliar should be able to carry out a number of further operations related to a number of different applications that generally are of spatial and/or temporal nature. Examples of applications where the Reasoner need to be involved may include:

- tracking of objects
- solution of the association problem
- aggregation of objects
- prediction of future object behavior in space and over time
- determination of complex object relationships

Determination of the result of these operations is carried out by the Reasoner by means of the learned rules in combination with the metadata and the available context information i.e. information from all the data sources. In this way new and more comprehensive queries can be created from templates in the rule base. These queries are then executed by the Σ QL query processor. This may lead to a situation that requires a second invocation of the Reasoner that takes place after the comprehensive query have been processed. In this way the Reasoner will be able to learn from the generation of the elaborate queries. However, in the above more comprehensive applications it may not be sufficient to just run a comprehensive query but also to take a step further and perform higher level information fusion, e.g. situation analysis but this is outside the scope of this work and must be subject to further research efforts.

6. Examples of Iterative Query Refinement for Information Fusion

In what follows we describe the typical tasks for iterative information fusion, which can be grouped into three types. The reasoning process as carried out here

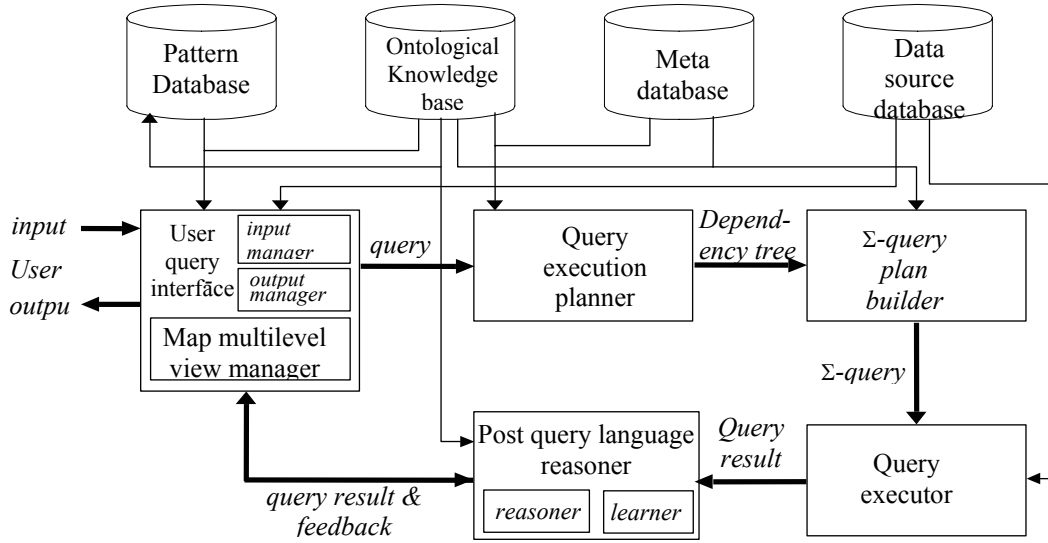


Figure 5. Architecture of the Sigma System

depends on whether the belief values that are output from any user query has got a value that is uninformative or simply that the user wants to extend the original query in some way.

6.1. Type I Queries

These queries require the generation of new and elaborate Σ QL queries.

Example: Are there any other objects in the proximity of the retrieved object that are of similar or of equal type as the retrieved object?

Proximity refers to the AOI or an extension of it and similarity to the ontology; the reasoner creates new elaborate queries from the templates.

The following Σ QL query corresponds to Task Type 1. Initially, the user tries to find “trucks” in a certain area of interest. This corresponds to the light grey colored Σ QL query. If the results are uninformative, the user may guide the reasoner to apply a more elaborate query as shown below. After the elaborate query is processed, the user is satisfied with the results. He can then tell the reasoner to remember the rule under a certain user-assigned task description, such as “objects similar to trucks”.

A simple way of handling similarity would be to define two objects as similar if they have a common ancestor in the ontological hierarchy no more than i steps up the inheritance chain, where e.g. $i=1$ or $i=2$. Assuming that $i=2$ then a Tank and a Car would be similar according to the ontology in Figure 2, whereas a Truck and a River is not.

```

Select objectk.type, objectk.position, objectj.type
cluster * alias objectk
from PerceptionSource
where relation(AOI, objectk) = ‘inside’
and objectk.type = ‘truck’
and objectj.t = objectk.t
and distance(objectk, objectj) <  $\delta$ 
and similar(objectk, objectj)
and objectj in
Select objecti.type, objecti.position
cluster * alias objecti
from PerceptionSource
where relation(AOI, objecti) = ‘inside’
and objecti.t = tgiven
and objecti.type = ‘truck’

```

6.2. Type II Queries

These queries require generally the invocation of particular functions that basically are concerned with the resolution of the association problem that may occur in just a single case or as a part of a tracking task. Examples of such functions are the determination of whether topological relations are true or false.

Example: Can the retrieved object be associated with an earlier single observation?

This query type requires the solution of the association problem.

The corresponding query is:

```

Select objectk.type, objectk.t, objectk.position,
objectj.type, objectj.t, objectj.position
cluster * alias objectk
from PerceptionSource
where relation(AOI, objectk) = ‘inside’
and distance(objectk.position, objectj.position) <  $\delta$ 

```

```

and tstart < objectk.t < objectj.t
and associate(objectk, objectj)
and objectj in
  Select objecti.type, objecti.position
  cluster * alias objecti
  from PerceptionSource
  where relation(AOI, objecti) = 'inside'
    and objecti.t = tgiven
    and objecti.type = 'truck'

```

6.3. Type III Queries

This task type requires only investigation of the result of the various sub-queries of the user defined query, that is in many cases an inspection of the content of the dependency tree.

Example: Did any sensor (data sources) contribute to the result in any extreme way? This refers to the single belief values from the various sensor related sub-queries and require only a check of the dependency tree.

The corresponding query is:

```

Select objectk.type, objectk.sensor, objectk.belief
  value, objectk.position
cluster * alias objectk
from DependencyTree
where qualitative-difference(objectk.belief-value,
  objectj.belief-value) = 'large'
  and objectk.position = objectj.position
  and objectk.type = objectj.type
  and objectk.t = objectj.t
and objectj in
  Select objecti.type, objecti.position
  cluster * alias objecti
  from PerceptionSource
  where relation(AOI, objecti) = 'inside'
    and objecti.t = tgiven
    and objecti.type = 'truck'

```

7. The User Interface

Each query is considered a compound query that can be constructed in several iterations. In each iteration the user constructs an elementary query, which is essentially a quintuple: <Object, Source, Time, Space, Direction>. Using query operators the elementary queries can be composed into a nested query in Σ QL. The composition rule is implicit, and the user does not need to know the underlying query language Σ QL for the nested query. What is required is an understanding of the meaning of different query operators. The main purpose of the user interface [8] is therefore to provide a visual interface for the user to compose compound queries from elementary queries using query operators, to present the query results and to give the user a qualitative evaluation of the results. To perform a "qualitative evaluation" means to show to the user the sub-set of the query that has been executed (if it has not been possible to execute the query

totally), and to give him/her information about the quality of the algorithms involved in the query execution.

The user interface must be adaptive in the sense that if new algorithms or new data sources are added to the system, new characteristics can be detected and the user interface must be able to show these new possibilities to the user. The user interface must help the user in query formulation based on the available data and algorithms. In fact it is useless for the user to define a query that cannot be executed. If the user knows this he/she may decide to define a simpler, but still useful, query.

We describe the user query formulation process through an example. Let us suppose that the user wants to execute the following query: "Find cars in a region 5 hours before the region is covered by a flood". We describe now the steps to be followed to define the query and the interactions between the user and the system:

1 The user selects the area in which he wants to execute the query, and the time interval of interest. To do this in a visual way the interface can present a territorial map.

2 The system checks if there is any available information in the spatial area and temporal interval of interest.. To do this the system checks if there are data sources with an active range that intersects the area in which the user is interested. If there are available data sources the system checks if they can provide data-units with timestamps belonging to the period in which the user is interested.

3 By using the Input Manager and the Map Multi Level View Manager, in a visual way, the system shows the areas in which there are no available data. At this point the interface presents the user with all the recognizable characteristics based on the algorithms and data sources available to the system to permit him/her to define a query that may be executed. For example, if the user wants to find a particular shape, the interface presents him/her with shape templates stored in the ontology knowledge base, gives the user the chance to paint a personal shape using an integrated editor or to import a shape from a file. In a similar way the interface must implement strategies to include in the query definition every recognizable characteristics. Finally the interface must enable the user to consider more than one Recognizable-Characteristic. For example, a colored-shape or two crashing-shapes. In the previous example the user must select a car shape and a moving gray object (flood). Finally the user must specify that the car is in the area five hours before it is covered by the flood.

4 By considering the data and algorithms available the system shows the areas where the query can be executed in its entirety, and areas where it can be executed only partially.

5 To improve the query results the Reasoner may need to pick an applicable rule from the rule base. If the Reasoner cannot select a rule by itself, the interface enables the user to specify a new rule.

6 The system displays the query results. For the areas where the query has been executed in its entirety the output is textual, for example “At 5.15 p.m. in locality x a car has been detected, locality x has been covered by the flood at 10.15 p.m.”. Furthermore the system must enable the user to view data from which the results have been obtained. In the case where the query has been executed only partially (for example the system detected some cars but could not detect the complete flood) the user must be informed exactly when the query was completed. Finally the results are presented visually. For the previous example the interface shows the image of the retrieved cars and their attributes, or data showing only the retrieved cars, or showing only the cars and the flood, or showing the complete context and so on. What the interface shows is determined by the available data and the query result.

The Map Multi Level View Manager (MMLVM) allows the interface to present territorial maps to the user and to give a visual representation of query results. The maps are presented using different ways or levels.

We consider the following levels:

Level 1: At this level MMLVM presents maps containing only immutable territorial characteristics, for example mountains or streets. To do this the system uses GIS maps. MMLVM must permit different zoom operations when presenting these maps.

Level 2: At this level the MMLVM projects information about available sensors to the system on the maps of level 1.

Level 3: When the user has specified a time interval of interest the MMLVM projects on the maps of level 2 information on external condition.

Level 4: The query results are showed on the maps of level 1.

The interface uses level 1 to enable the user to specify the area of interest in the first step of the query definition. Then the interface uses level 2 to project information from available sensors on the specified area and level 3 to project on the maps information on external conditions. Finally Level 4 is used to present query results visually.

8. Conclusion and Future Work

In this paper an architecture for interactive query refinement for a sensor-based information fusion system has been described. The architecture is based on the Σ QL query language and includes an elaborate ontology and its knowledge-base. The ontological knowledge system supports the query processing by selecting the most appropriate sensors and sensor data analysis algorithms in a sensor data independent way, which means that the users are not involved in the sensor and algorithm selection process. The ontology also include means for determination of the most relevant objects with respect to the applied queries, i.e. the existing object patterns that actually are used for the recognition of these objects. The

query process is basically controlled by the dependency tree that controls the different steps in the query execution.

The Sigma System also includes a reasoner called Pequiliar that is referred to as a post query language reasoner. The purpose of this Reasoner is to automatically or semi automatically carry out an iterative information fusion step by means of rules that are either automatically launched by the Reasoner or defined by the user. In the latter case the Reasoner can learn the new rule and use it in later queries. A rule corresponds to a refined query described in terms of a generic pattern. Through the refined queries a better support for the higher levels of information fusion can be achieved in particular for situation awareness and impact analysis.

References:

- [1] D.L. McGuinness “Ontologies for Information Fusion” , Proceedings of the International Conference on Information Fusion 2003 (Fusion'03), Cairns, Australia, July 8-11, pp 650-657
- [2] S.K. Chang, "The Sentient Map", Journal of Visual Languages and Computing, Academic Press, Vol. 11, 2000, 455-474.
- [3] S.K. Chang, E. Jungert and G. Costagliola, “Multi-sensor Information Fusion by Query Refinement”, Proc. of 5th Int'l Conference on Visual Information Systems, Hsin Chu, Taiwan, March 2002, pp. 1-11.
- [4] S.K. Chang, E. Jungert and G. Costagliola, “Queryng Distributed Multimedia Databases and Data Sources for Sensor Data Fusion”, to appear in IEEE Transactions on Multimedia, 2004.
- [5] Handbook of Multisensor Data Fusion, D.L. Hall & J. Llinas (Eds.) CRC Press, New York, 2001.
- [6] T. Horney, ”Design of an ontological knowledge structure for a query language for multiple data source”, FOI-R-0498-SE, May, 2002, ISSN 1650.1942, Scientific Report.
- [7] T. Horney, E. Jungert, M. Folkesson, “An Ontology Controlled Data Fusion Process for Query Language”, Proceedings of the International Conference on Information Fusion 2003 (Fusion'03), Cairns, Australia, July 8-11.
- [8] J. Nielsen, “Usability Engineering”, Morgan Kaufman, New York, 2001