Iterative Information Fusion using a Reasoner for Objects with Uninformative Belief Values

S. K. Chang[1] and Erland Jungert[2]
[1]Dept. of Computer Science, University of Pittsburgh, USA (chang@cs.pitt.edu)
[2]Swedish Defence Research Agency (FOI), Sweden (jungert@foi.se)

Abstract - *We describe an approach for iterative information fusion using a context-dependent Reasoner called Pequliar. The system basically consists of a query processor with fusion capability and a Reasoner with learning capability. The query processor first performs a query to produce some initial results. If the initial results are uninformative, then the Reasoner guided by the user creates a more elaborate query by means of some rule and returns the query to the query processor that executes it and returns a more informative answer. Rules may be initially specified by the user and subsequently learned by the Reasoner. Examples of iterative queries are drawn from multi-sensor information fusion applications.*

1 Introduction

Reasoning about data of uncertain type is necessary in a growing number of complex applications and especially for such applications where the basic data sources correspond to sensors of multiple type and where the generated data are of heterogeneous type. Data from such sensor sources are always associated with some level of uncertainty. To gain acceptable results from the analysis of data from multiple sensors multiple sensor data fusion is required [9]. For this reason, information, that for the most part is of spatial and temporal type need to be acquired from the sensors. It has been shown, e.g. in [4], that this can be made by means of a specific query language. Subsequently it will be demonstrated how the output from the query language can be used as input to a spatial/temporal reasoning tool that can resolve the uncertainty problems. Basically, the Reasoner creates a more elaborate query by means of some rule and returns the query to the query processor that executes it and returns a more expanded answer. The Reasoner described in this paper is called Pequliar and can be described as a post-query-language-reasoner. The name "Pequliar" is an acronym for ***P***(ost)+***e*** +***QU***(ery) ***L***(anguage)+***iar***+***R***(easoner).

Pequliar can be seen as a Reasoner for objects determined through queries in a query language. In particular, the query language used is ΣQL [2], [4]. This query language has the ability to respond to queries where the input data sources are sensors that generate images and where the data is of uncertain type. More specifically, in ΣQL these uncertainties are associated with belief values that must be subject to further interpretation either by the user or as here by the Reasoner. Since we are here talking about multiple sensor data sources as input to the query language, it must be possible to fuse the information from the various sensors prior to the usage in the Reasoner; a capability that is available in ΣQL. The eventual output from Pequliar, that is rule driven, should help the user to get a better understanding of the external environment subject to the analysis carried out by the complete system. The query language and the Reasoner is developed to support a number of applications, such as data mining of primarily unstructured data [11], and for higher levels of information fusion, i.e. situation and impact analysis [9], [5].

As the object information is associated with various kinds of uncertainties and as the object information from a single sensor may or may not correspond to the same objects as acquired from other sensors this will have an impact on how sensor data fusion is carried out. To make it possible for the user to interpret the uncertainty level of an identified object the query language must somehow produce a measure of the actual uncertainty level that can be associated with the object information. In ΣQL this measure is called a belief value. However, this does not always create a result that is simple to interpret. One way to overcome this problem is, as already has been mentioned, to apply the query output to a Reasoner and to combine it with various types of background information that for the most part is context dependent, i.e. if the objects to be reasoned about correspond to vehicles then most likely the background information is geographical. Furthermore, this will also include information about the past behavior of the vehicles as well as the behavior of other objects. Besides being rule driven the reasoning process also requires information from

an ontology. Results from the Reasoner can, however, be handled in various ways, that is, if the result is feasible it is returned to the user, if not there must be a request sent to the sensors for more information followed by a rerun of the query. Besides this, background information may be accessed from other sources like geo-databases. Basically, Pequliar will take care of the query results that may be difficult to interpret without considering the background information. Consequently the Reasoner must also include some element of fusion since data from new data sources are integrated into the process. In Pequliar metadata is a necessary means for determination of the background information that will be needed in the reasoning process.

We describe an approach for iterative information fusion using a context-dependent Reasoner called Pequliar. Section 2 discusses the system design. The system basically consists of a query processor and a Reasoner where the query processor has fusion capability and the Reasoner learning capability. As described in Section 3, the query processor first performs a query to produce some initial results. If the initial results are uninformative, then the Reasoner guided by the user creates a more elaborate query by means of some rule and returns the query to the query processor. The query processor executes it and returns a more informative answer. Rules may be initially specified by the user, and successful rules are learned by the Reasoner. Section 4 discusses reasoning issues. Section 5 and Section 6 present iterative information fusion based upon examples drawn from multi-sensor information fusion applications. Further research issues are discussed in Section 7.

2 System Design

2.1 System Overview

The system consists mainly of a query processor and a Reasoner. The query processor first performs a query to produce some initial results. If the initial results are uninformative then the Reasoner guided by the user creates a more elaborate query by means of some rule and returns the query to the query processor. The query processor executes it and returns a more informative answer. Rules may be initially specified by the user and subsequently learned by the Reasoner.

The name of the Reasoner - Pequliar - is obviously related to the word "*peculiar*". This word "*peculiar*" has several different meanings and in the context of this paper means "one of a kind", "singular" or "to be an object with uncertain and/or uninformative belief values". The system is therefore intended to handle objects that are one-of-a-kind, singular, or with uncertain and/or uninformative belief values. To accomplish this, any information peculiar to an object in the application domain must be stored in the ontology, and any information peculiar to spatial/ temporal reasoning must be stored in the rule base, see Figure 1.

2.2 Iterative Query Construction

Each user's 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 ΣQL. The composition rule is implicit, and the user does not need to know the underlying query language ΣQL for the nested query. What is required is an understanding of the meaning of different query operators. The main purpose of the user interface is therefore to provide a visual interface for the user to compose compound queries from elementary queries using query operators. A preliminary study of the user interface can be found in [10].

2.3 The Query Processor

The result of a ΣQL-query is generally the object types requested by the user including also the attribute and status values of these objects, if requested. Examples 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 ΣQL is that the type attribute is associated with a belief value. Other attributes and status values may have belief values associated to them 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.

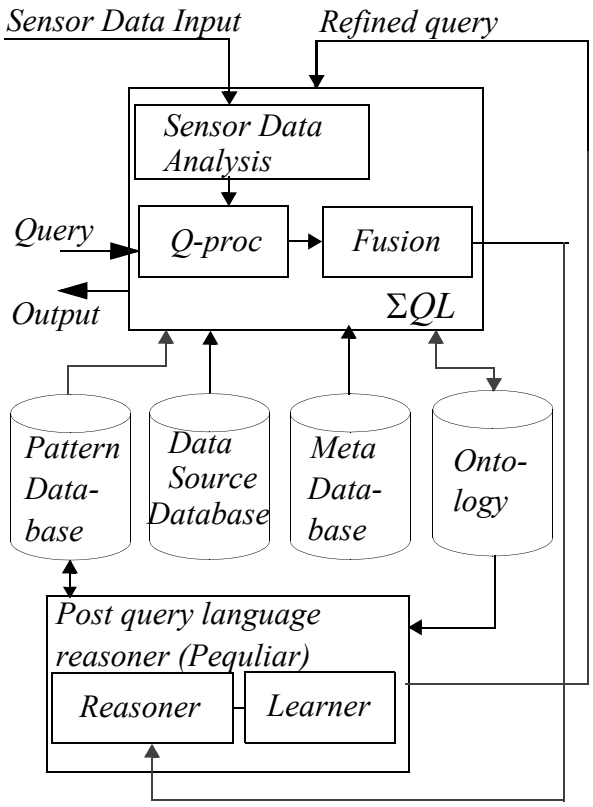

Figure 1. The system diagram.

2.4 Iterative Information Fusion by Belief Value Adjustment

New, and hopefully more informative, belief values will be achieved through a reasoning step in Pequliar that includes generation of new and more elaborate query 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 meta-data. 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 Reasoner has come to a conclusion in its process a new and elaborate query is created and executed.

3 The Pequliar 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* **x** *Object* **x** *Attributes* **x** *Time* **x** *Location* **x** *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 searchers 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, these rules constructed by the user are remembered, forming part of the scenario.

4 Some Reasoning Issues

The Pequliar Reasoner should be able to carry out a number of different tasks related to a number of different applications that generally are of spatial and/or of temporal character. Applications where the Reasoner may be invoked to support the users may, for example, 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,
- high level information fusion, e.g. situation analysis.

Determination of the result of these operations is carried out by the Reasoner by means of the learned rules in combination with the meta-data and the available context information. In this way new and more comprehensive queries, of which some may be recursive, can be created from templates in the rule-base. These queries are then executed in the ΣQL 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.

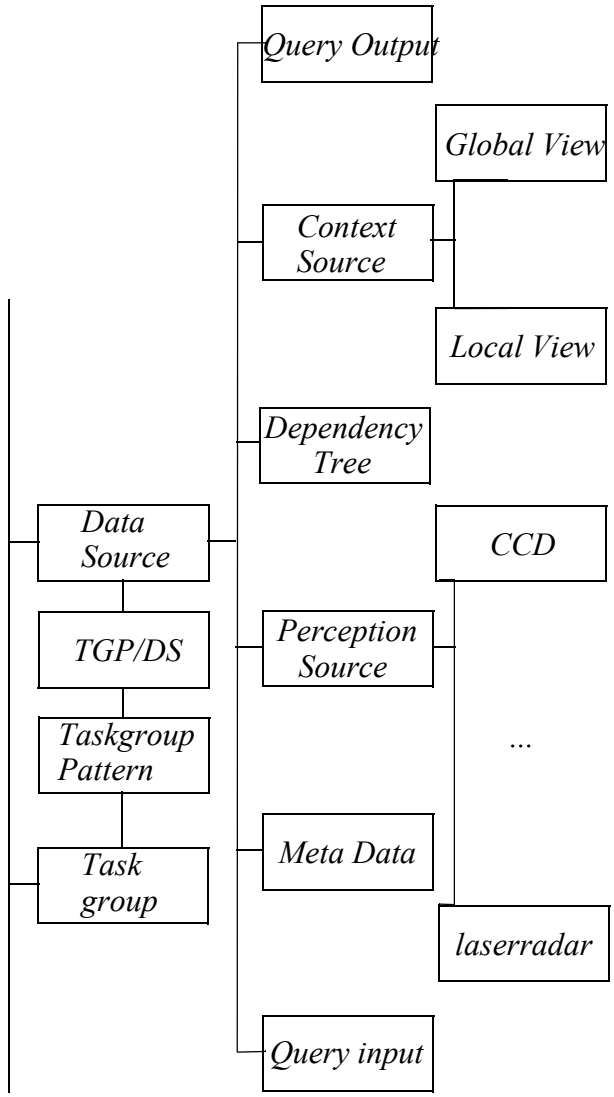

Figure 2. Data sources, task groups and their relations as part of the ontology, to be used for context sensitive object assessment in the Pequliar system.

4.1 Context Sensitive Object Assessment

The background or proper context in which an object may occur and that is subject to a query in ΣQL can be seen as a data source used to enhance the outcome of the query. This was similarly discussed in [3] where the local and the global views 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, see section 5.1, 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 2.

Query input includes information delivered by the users including such information as area-of-interest, (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. 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, section 5.3, that asks for pre-fusion sub-results that may be of extreme type. Finally, 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 Pequliar. Although this may cause difficulties in composing iterative queries it also extends the number of possible queries, which in turn, makes ΣQL much more adaptive to complex problems related to, for instance, the higher orders of information fusion, i.e. 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 2. The structure in Figure 2 is an extension to the ontology described in [6], which is related to work discussed in [7], [8] that in both cases discuss work where the aim is to develop ontologies for situation awareness. Context Source, refers to the actual geographical background while Perception Source refers to the sensor data sources. Dependency tree is the partial query result from a particular sensor data source. The remaining data sources are self-explanatory.

4.2 Deriving Query Patterns from Query Paths

As mentioned above, the user formulates a query by composing compound queries from elementary queries using query operators. But the problem is that for a compound query, this may make the query building time consuming and boring. An approach to speed up this process is proposed based upon two concepts: *query path* and *query pattern*. A query path $\boldsymbol{p}$ is a sequence of queries (Q_0, Q_1, Q_2, ……, Q_n) in the iterative query formulation process. Basically a query path shows the track of an interactive query construction from Q_0 to Q_n. In each step, a query operator and an elementary query are applied on the previous one. While a query pattern is generalized from query paths, some constant values in the queries may be marked as variables to indicate that these values are exchangeable while doing a pattern match. Query patterns are important because they could cover most of the frequently used queries and make them simple. In our approach, a set of query patterns $S_{pattern}$ is stored and maintained during the query processing. At the initial state, there are no query patterns in $S_{pattern}$. When one query is finished, the

query path will be shown and the user is asked to choose the variables and input a linguistic tag for it. A new query pattern is generated and added to $S_{pattern}$. In query formulation, the query patterns are retrieved dynamically and then selected by the user. Therefore the Pequliar Reasoner can learn new patterns, and use the patterns to provide short cuts for the user to formulate queries.

5 Tasks for Information Fusion

In what follows we describe the typical tasks for information fusion, which can be grouped into three types. The reasoning process as carried out here depends on whether the belief values that are output from any user query has got a value that is uninformative. Three different types of task queries have been identified. The first type is concerned with how to improve the result of the original query by considering other aspects of the sensor data sources, such as for instance similarity among requested objects, but also proximity between the objects. The second type is merely concerned with how to associate different object instances with each other while the third group concerns queries that need to inspect the dependency tree from the last user query. All these three task types are related in that they will result in new elaborate queries that are part of the iterative information fusion process that also includes learning of rules in that process. Here we also demonstrate the three task types with some relevant examples of elaborate ΣQL queries.

5.1 Type I Tasks

These tasks require the generation of new and elaborate ΣQL queries that basically depends on certain conditions among the spatial objects normally found in sensor data.

1) 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 and *similarity* to the ontology; the Reasoner creates new elaborate queries from the templates.
2) Have there been any other objects in the proximity of the retrieved object that are of similar or of equal type as the retrieved object?
   The Reasoner creates new elaborate queries from the templates.
3) Is the present background (context) information of the proper type relative to the type of the retrieved object?
   Context refers to the geographical background in the AOI and the Reasoner creates new elaborate queries from the templates.
4) Has this object type been previously observed in this background context?
   This query is similar to 3 but requires involvement of earlier output instances from ΣQL, i.e. the IOI must be involved as well.
5) Is it possible that the quality of the background information may have influenced the query result?
   For instance is there any missing data in the AOI. This could be determined from a particular ΣQL query. For a lot of missing data the risks of not finding a particular object increases.
6) Is the retrieved object partly hidden by some other object that is part of the context?
   The context refers to the local background that must be in high resolution. This allows the Reasoner to create a new elaborate query.
7) Are there any other object types in the proximity of the retrieved object that may have any *impact* of some kind on the found object?
   This query is similar to task type 6 but here the object may have a location that is too close to some other object which will have some more or less serious consequences on the primary object.

5.2 Type II Tasks

This task type requires generally the invocation of a particular function that basically is concerned with the resolution of the association problem that may occur in single cases, that is between a pair of registered objects, or as a part of a tracking task, including a time sequence of the registered objects.

8) Can the retrieved object be associated to an earlier single observation?
   This query type requires the solution of the association problem.
9) Can the retrieved object be part of an existing track.
   This task type is a recursive variation of query type 8 that includes the application of the association problem but will not be further elaborated here.

5.3 Type III Tasks

This task type requires only investigation of the result of the various sub-queries of user defined queries, that corresponds, in most cases, to an inspection of the content of the dependency tree

10) 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; only a check of the dependency tree is required.
11) Did the result of some of the sensors (data sources) contradict each other?
    Does not require any new ΣQL query; only an inspection of the dependency tree.
12) Which sensors (data sources) were used to answer the query?
    Does not require any new ΣQL query; only an inspection of the dependency tree.
13) Are there any attributes or status values of the retrieved object that *in particular* could have diverted the outcome of the primary query.
    This means that the attribute did not in any case contribute to the query result; contrary it could

have contributed to another outcome of the query.

6 Examples of Iterative Queries

In this section a set of iterative queries generated by the Reasoner will be shown. These queries are examples of the three task types introduced in Section 5. Most of these queries require two iterations although it is possible to refine query formulation into three or even four iterations, depending upon how the query formulation is done by the user. In the query examples *relation* refer to topological relations between a pair of spatial objects of which AOI is also considered being a spatial object.

Example 1: The following ΣQL query corresponds to Query 1 of Task Type 1. Initially, the user tries to find "trucks" in a certain area of interest. This corresponds to the light grey colored ΣQL 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".

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

```
Select object_k.type, object_k.position, object_j.type
cluster * alias object_k
from PerceptionSource
where relation(AOI, object_k) = 'inside'
   and object_k.type = 'truck'
   and object_j.t = object_k.t
   and distance(object_k, object_j) < δ
   and similar(object_k, object_j)
   and object_j in
       Select object_i.type, object_i.position
       cluster * alias object_i
       from PerceptionSource
       where relation(AOI, object_i) = 'inside'
          and object_i.t = t_given
          and object_i.type = 'truck'
```

Example 2: The following ΣQL query corresponds to Query 2 of Task Type I. As in the previous example, initially the user tries to find "trucks" in a certain area of interest. This corresponds to the light grey colored ΣQL query. The user may then decide to guide the Reasoner to apply a more elaborate query as shown below to find similar objects in previous time periods. 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 in previous time periods".

Query: *Have there been any other objects in the proximity of the retrieved object that are of similar or of equal type as the retrieved object?*

```
Select object_k.type, object_k.position, object_k.t,
       object_j.type, object_j.position
cluster * alias object_k
from PerceptionSource
where relation(AOI, object_k) = 'inside'
  and distance(object_k.position,object_j.position)
      < δ
  and similar(object_k.type, object_j.type)
  and t_start < object_i.t < object_k.t
  and object_j in
      Select object_i.type, object_i.position
      cluster * alias object_i
      from PerceptionSource
      where relation(AOI, object_i) = 'inside'
         and object_i.t = t_given
         and object_i.type = 'truck'
```

Example 3: task type I query 3

Query: *Is the present background (context) information of proper type relative to the type of the retrieved object?*

```
Select object_j. type,object_j.position,object_p.type
cluster * alias object_p
from ContextSource
where relation(object_p, object_j) = 'inside'
   and properbackground(object_p.type,
                       object_j.type)
   and object_j in
       Select object_i.type, object_i.position
       cluster * alias object_i
       from PerceptionSource
       where relation(AOI, object_i) = 'inside'
          and object_i.t = t_given
          and object_i.type = 'bus'
```

Example 4: task type I query 4

Query: *Has this object type been previously observed in this background context?*

```
Select   object_p.type, object_i.type,
         object_i.position
cluster * alias object_p
from ContextSource
where object_j.type = object_p.type
   and object_i.type = object_k.type
   and t_start < object_i.t < object_k.t
   and object_i in
         Select object_m.type, object_m.t,
                object_m.position
          cluster * alias object_m
          from PerceptionSource
          where relation(AOI, object_m) =
                'inside'
             and t_start < object_m.t < t_given
             and object_m.type = 'bus'
   and object_j in
         Select object_l.type
```

```
cluster * alias object_l
from ContextSource
where relation(AOL, object_l) =
        'partlyoverlap'
   and object_k in
       Select objedct_n.type,
              object_n.position
       cluster * alias object_n
       from PerceptionSource
       where
          and relation(AOI, object_n) =
              'overlap'
          and object_n.t = t_given
          and object_n.type = 'bus'
```

Example 5: task type query 5

Query: *Is it possible that the quality of the background information may have influenced the query result?*

```
Select object_p.sensor
cluster * alias object_p
from PerceptionSource
where relation(object_p.background, object_j) =
        'inside'
and missingdata(object_p.background) > 50
      /* more than 50% of the background data
       may be missing */
and object_p.t = t_given
and object_j in
    Select object_i.type, object_i.position
    cluster * alias object_i
    from PerceptionSource
    where relation(AOI, object_i) = 'inside'
       and object_i.t = t_given
       and object_i.type = 'bus'
```

Example 6: task type I query 6

Query: *Is the retrieved object partly hidden by some other object that is part of the context?*

```
Select object_p.type, object_p.position,
       object_j.type, object_j.position
cluster * alias object_p
from ContextSource
where relation(AOI, object_p) = 'partlyoverlap'
   and ((object_p.type = 'terrain-feature')
      or (object_p.type = 'building')
      or (object_p.type = 'natural-object))
   and ((relation(object_p, object_j) =
          'partlyoverlap')
     or (relation(object_p, object_j) = 'beside'))
   and object_p.t = object_j.t
   and object_j in
      Select type
      cluster * alias object_i
      from PerceptionSource
      where relation(AOI, object_i) = 'inside'
         and object_i.t = t_given
         and object_i.type = 'bus'
```

Example 7: task type I query 7

Query: *Are there any other object types in the proximity of the retrieved object that may have any impact of some kind on the found object?*

```
Select object_k.type, object_k.position,
       object_j.type, object_j.position
cluster * alias object_k
from PerceptionSource
where relation(AOI, object_k) = 'inside'
   and distance(object_k.position,
                object_j.position) < δ
   and object_k.type = 'dog'
   and non-coexistant(object_k, object_j)
   and object_k.t = object_j.t
   and object_j in
      Select object_i.type
      cluster * alias object_i
      from PerceptionSource
      where relation(AOI, object_i) = 'inside'
         and object_i.type = 'cat'
         and object_i.t = t_given
```

Example 8: task type II query 8

Query: *Can the retrieved object be associated to an earlier single observation?*

```
Select object_k.type, object_k.t, object_k.position,
       object_j.type, object_j.t, object_j.position
cluster * alias object_k
from PerceptionSource
where relation(AOI, object_k) = 'inside'
   and distance(object_k.position,
                object_j.position) < δ
   and t_start < object_k.t < object_j.t
   and associate(object_k, object_j)
   and object_j in
      Select object_i.type, object_i.position
      cluster * alias object_i
      from PerceptionSource
      where relation(AOI, object_i) = 'inside'
         and object_i.t = t_given
         and object_i.type = 'truck'
```

Example 10: task type III query 10

Query: *Did any sensor (data source) contribute to the result in any extreme way?*

```
Select object_k.type, object_k.sensor,
       object_k.belief-value, object_k.position
cluster * alias object_k
from DependencyTree
where qualitative-difference
          (object_k.belief-value,
          object_j.belief-value) = 'large'
   and object_k.position = object_j.position
   and object_k.type = object_j.type
   and object_k.t = object_j.t
   and object_j in
      Select object_i.type, object_i.position
```

```
    cluster * alias object_i
    from PerceptionSource
    where relation(AOI, object_i) = 'inside'
      and object_i.t = t_given
      and object_i.type = 'truck'
```

Example 11: task type III query 11

Query: *Did the final result contradict the result from any of the sensors (data sources)?*

```
Select object_k.type, object_k.sensor,
       object_j.type, object_j.position
cluster * alias object_k
from DependencyTree
where object_k.position = object_j.position
  and object_k.type ≠ object_j.type
  and object_k.t = object_j.t
  and object_j in
    Select object_i.type, object_i.position
    cluster * alias object_i
    from PerceptionSource
    where relation(AOI, object_i) = 'inside'
      and object_i.t = t_given
      and object_i.type = 'truck'
```

Example 12: task type III query 12

Query: *Which sensors (Perception Sources) where used to answer the query?*

```
Select object_k.sensor
cluster * alias object_k
from DependencyTree
 where object_k.position = object_j.position
   and object_k.type = object_j.type
   and object_k.t = object_j.t
   and object_j in
     Select type
     cluster * alias object_i
     from PerceptionSource
     where relation(AOI, object_i) = 'inside'
       and object_i.t = t_given
       and object_i.type = 'truck'
```

7 Discussion

In this work a reasoning system for objects with uninformative belief values has been introduced. The Reasoner can be viewed as a post query language processor, since the input to the Reasoner is the output from ΣQL a query language, primarily, developed for data from multiple sensors. An important characteristic of ΣQL is that it allows sensor data fusion and includes an ontology that allows application of queries in a sensor data independence way, that is a user does not have to know which sensors that are involved when a query is answered. In particular, to allow the extension of the queries the query processor has been extended with respect to the data sources. This is mirrored by the ontology that demonstrates how the context information as well as the dependency tree information can be used as input to the refined queries. This is done in the same manner as the perception sources, which were introduced in the original approach to ΣQL, are used. The extended query technique is, finally, demonstrated with a number of elaborate queries that demonstrates the technique with three different task type queries that correspond to patterns that can be determined by the Reasoner or alternatively by the user. This illustrates how the Reasoner eventually can learn how to automatically apply the more elaborate queries.

In connection to this work there are also other means that need to be studied further, such as generation of even more elaborate queries to support applications related to, e.g. spatial data mining.

References:

[1] Chang, S.-K. and Jungert, E.: Human and system directed fusion of multimedia and multimodal information using the σ-tree data model. Proceedings of the 2nd International conference on Visual information systems, San Diego, CA, December 15-17 (1997) 21-28.

[2] S.-K. Chang, G. Costagliola, E. Jungert, Spatial/ Temporal Query Processing for Information Fusion Applications, Advances in Visual Information Systems, R. Laurini (Ed.), Lecture Notes in Computer Science 1929, Springer Verlag, Berlin, 2000, pp 127-139.

[3] S.-K. Chang, G. Costagliola, E. Jungert, Multisensor Information Fusion by Query Refinement, Recent advances in Visual information systems, S.-K. Chang, Z. Chen, S.-Y. Lee (Eds.), Lecture Notes in Computer Science 2314, Springer Verlag, Berlin, 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, London, 2001.

[6] Horney, T., Jungert, E., Folkesson, M., *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.

[7] Matheus, C. J., Kokar, M., Baclawski, K., *A Core Ontology for Situation Awareness*,Proceedings of the International Conference on Information Fusion 2003 (Fusion'03), Cairns, Australia, July 8-11.

[8] McGuinness, D. L., *Ontologies for Information Fusion,* Proceedings of the International Conference on Information Fusion 2003 (Fusion'03), Cairns, Australia, July 8-11, pp 650-657.

[9] E. Waltz and J. Llinas, Multi-sensor data fusion, Artect House, Boston, 1990.

[10] K. Silvervarg, E. Jungert, *Aspects of a visual user interface for spatial/temporal queries*, proceedings of the 9th international conference on Distributed Multimedia Systems, Miami, September 24-26, 2003,pp 287-293.

[11] S. Chawla, S. Shekhar, W.-L. Wu, U. Ozesmi, *Modeling spatial Dependencies for mining geospatial data: An introduction*, H. Miller, J. Han (Eds.), Geographic Data Mining and Knowledge Discovery (GKD), Taylor and Francis, 1999.