INCREMENTAL QUERY PROCESSING IN INFORMATION FUSION SYSTEMS

by

Xin Li

B.E., Tsinghua University, P. R. China, 1999

M.E., Tsinghua University, P. R. China, 2001

M.S., University of Pittsburgh, USA, 2003

Submitted to the Graduate Faculty of

Arts and Sciences in partial fulfillment

of the requirements for the degree of

Doctor of Philosophy

University of Pittsburgh

2006

UNIVERSITY OF PITTSBURGH

DEPARTMENT OF COMPUTER SCIENCE

This dissertation was presented

by

Xin Li

It was defended on

Feb. 23, 2006

and approved by

Dr. Shi-Kuo Chang

Dr. Alexandros Labrinidis

Dr. Ching-Chung Li

Dr. Janyce Wiebe

Dissertation Director: Dr. Shi-Kuo Chang

ABSTRACT

INCREMENTAL QUERY PROCESSING IN INFORMATION FUSION SYSTEMS

Xin Li, PhD

University of Pittsburgh, 2006

This dissertation studies the methodology and techniques of information retrieval in fusion systems where information referring to same objects is assessed on the basis of data from multiple heterogeneous data sources. A wide range of important applications can be categorized as information fusion systems e.g. multisensor surveillance system, local search system, multisource medical diagnose system, and so on. Up to the time of this dissertation, most information retrieval methods in fusion systems are highly domain specific, and most query systems do not address fusion problem with enough efforts.

In this dissertation, I describe *a broadly applicable query based information retrieval approach* in general fusion systems: user information needs are interpreted as fusion queries, and the query processing techniques e.g. source dependence graph (SDG), query refinement and optimization are described. Aiming to remove the query building bottleneck, a *novel incremental query method* is proposed, which can eliminate the accumulated complexity in query building as well as in query execution. Query pattern is defined to capture and reuse repeated structures in the incremental queries. Several *new techniques for query pattern matching and learning* are described in detail. Some *important experiments* in a real-world multisensor fusion system, i.e. the intelligent vehicle tracking (IVET) system, have been presented to validate the proposed methodology and techniques.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ACKNOWLEDGEMENTS

It took me years of hard work to make this dissertation come into being. However, it is not just the fruit of great efforts of mine. In fact, I can not even start if without the help and support of my advisor, committee members, colleagues, and my family.

The first person I would like to thank is my advisor – Dr. Chang. He is a superb advisor. I can not be luckier to work with him for almost five years. It is with him that I first learn how to do research and how to write a paper. He always blows me away with his amazing intelligence, knowledge, and vision. At every time I fell down, he always encourages me with warm and caring words. For me, he really did much more beyond being an advisor.

I owe special debts to my committee members. They are all great professors, and I have learned a lot from them. Their responsibility ensured the quality of this dissertation. Particularly I would like to thank Dr. Li for his supports and encourages from the very beginning, and thank Dr. Wiebe for teaching me all that I know about knowledge representation. I especially thank Dr. Labrinidis who took tremendous efforts to read the preliminary draft of my dissertation proposal and correct it word by word.

I am also grateful to Dr. Erland Jungert who is the co-director of our project in Swedish Defense Agency. Thank him for providing a solid theoretical and experimental basis for my work.

This dissertation also benefited from the collaboration of many fantastic colleagues. Thank Weiying Dai, Prasad Lakkavaram, Jianhang Xu, Karin Silvervarg (Sweden), Shuyi Shao, and too many others to name for their kind help.

Finally, I would like to thank my parents who encourage me all the time, and my wife – Lihua Fan who gives me the strongest support for every single word I wrote. It is to them I dedicate this dissertation.

1.0 INTRODUCTION

This dissertation studies the methodology and techniques of information retrieval in fusion systems where information referring to the same objects is assessed on the basis of data from multiple heterogeneous sources. For example, in an intelligent surveillance system, three types of sensors – laser radar (LR), infrared video (IR, similar to video but generated at 60 frames/sec), and CCD digital camera are all employed to keep track of vehicles in a ground area. The methodology and techniques utilizing the data from these sources to retrieve information (e.g. "find a red car followed by a truck in 10 minutes in a given area and time") as accurate and complete as possible is the main subject of my research.

I begin this chapter by showing that a wide range of important applications (e.g. multisensor intelligent surveillance system, information retrieval on the World-Wide Web, and medical diagnose system) can be categorized as information fusion systems (section 1.1). Next, the main challenges for the information retrieval in fusion systems are explained in detail (section 1.2). Then the state of the art of the research on information fusion systems is discussed under the framework of the well known JDL model, showing that the bottleneck hindering the usability of fusion systems has been moving from the low-level fusion to the high-level knowledge based reasoning and HCI (human computer interface). Finally, my research is overviewed comparing with related research areas (section 1.4 - 1.6), and a road map is given for the rest of this dissertation (section 1.7).

1.1 INFORMATION FUSION SYSTEM

Information fusion system is a system that can retrieve information from multiple heterogeneous sources, where data referring to same objects are fused for more complete and accurate assessments. The needs for information fusion systems have been originally conceived from military applications. For example, the multi-sensor surveillance system is one of the most widely investigated information fusion systems.

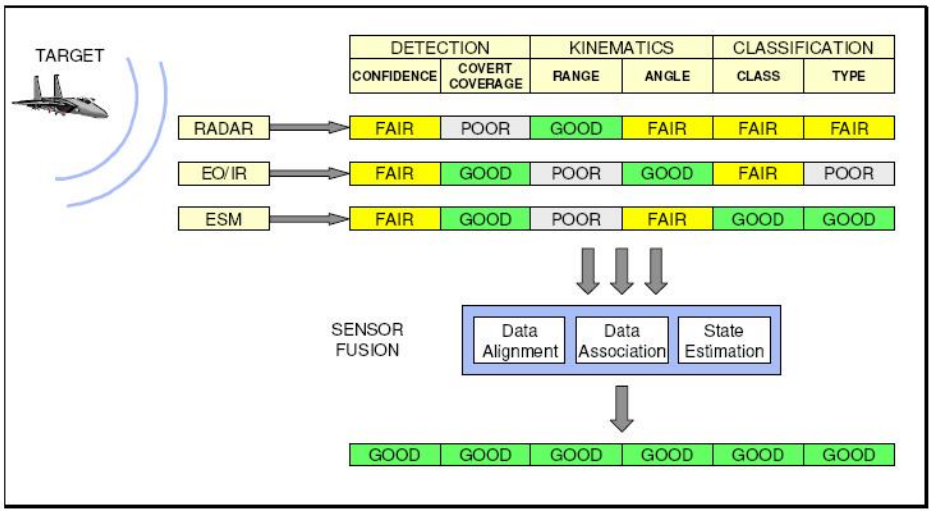

Figure 1: Multisensor Data Fusion in a Battle Field Surveillance System [SBW98]

Example 1.1.1 *An instance of sensor-based information fusion system [SBW98] is illustrated in Figure 1. Three types of sensors (i.e. RADAR, EO/IR, and ESM) are applied to keep track of aircrafts in battle fields. Since individual sensors have their own strength and weakness, the data from single type of sensors usually yields unsatisfied results. However, it could get better output by overlapping (fusing) them.*

In recent years, information fusion becomes a major need in many nonmilitary applications as well as military applications. Particularly, with the overwhelming volume of data available today (e.g. data on the Internet or in large databases), fusion becomes an important technique to improve the quality of information retrieval.

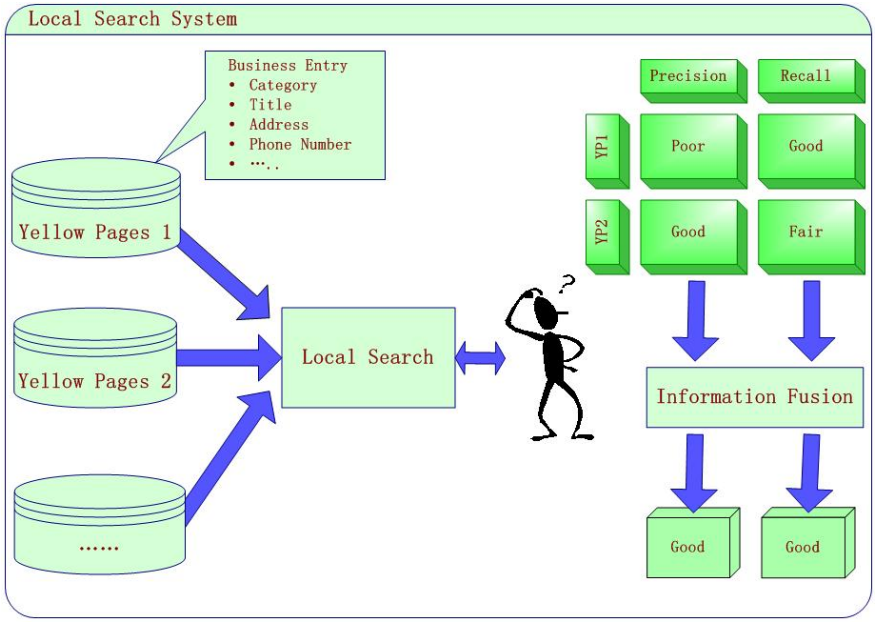

Figure 2: Local Search System

Example 1.1.2 *An online* local search *system [Him05] is a good example of information fusion system in nonmilitary area, which has significant commercial perspectives. Local search is a specialized search engine which works for the same purpose as traditional Yellow Pages (Figure 2). Using it, people can search information about local businesses online instead of reading local newspaper, classified ad circulars, and Yellow Pages directories. To ensure high search quality, most local search systems (e.g. Yahoo Local [Yah], and Google Local [Goo]) can retrieve data from multiple nation-wide Yellow Pages providers. Because data sources have their own characteristics in terms of the precision and recall, the data from different sources are fused to get the maximum accuracy and completeness.*

Another important fusion application class is intelligent medical diagnoses systems, where diseases and abnormalities are identified by the fusion of data from multiple sources (e.g. X-rays, NMR, temperature, IR, biological data) [GSGN+01].

Recently increasing attention to information fusion techniques has been raised in data mining community. Information fusion has been used to improve the quality of raw data prior

to the application of data mining methods, build combined models from multiple processes, and extract more precise data [Tor03]. Many data mining systems can be considered as an extension of information fusion systems.

More applications addressing information fusion problem with considerable attention are robot vision [HL01], data management in sensor network [YG03], automated control of industrial manufacture systems [HL01], environmental monitoring [HL01], emergency management in disaster areas [CDH+02].

To summarize, the explosive growth of information available in the real world has significantly increased the needs for information fusion. Nowadays more and more applications can be categorized as information fusion systems. The information retrieval in fusion systems turns out to be a core problem with the dramatically increasing importance. Although the specific information retrieval and fusion techniques could vary on application to application, there are many fundamental *common characteristics* for the majority of fusion systems. In this dissertation, I will present a broadly applicable solution for information retrieval problem in fusion systems on the basis of these common characteristics.

1.2 CHALLENGES

The complexity of information fusion systems is characterized by the following fundamental reasons:

- A major challenge in information fusion system is lack of methodology and techniques to utilize a large amount of (redundant) data from multiple sources.

  The data sources (e.g. sensors, and World-Wide Web resources) in fusion systems may behave quite differently from the traditional databases. They could have a large amount of data referring to same objects. Frequently, many of them may conflict with each other. It is often extremely hard to make a good use of all the available data. The systems could be simply overwhelmed by too much redundant data. In the worst cases, information retrieved from multiple data sources is even worse than the one from any single data

sources.

- Another challenges in information fusion systems is raised by the difficulty in the collaboration of heterogonous sources.

  Frequently, it could be very cumbersome to build a common understanding among the data sources which may produce information in quite different semantic formats. In many fusion systems, different data sources have quite different terminologies. For instance, in a multisensor fusion system one sensor can classify the detectable mobile ground objects as car, truck, tank, and bus. Meanwhile, the other one performs the classification with non-combat vehicle (i.e. car, bus and truck in one category), tank, and IFV. One more example, in the local search system described in example 1.1.2, Yellow Pages providers usually have different category systems. A business entity – a pizza shop can be listed under the category of fast food, restaurant, or even outdoor delivery in different Yellow Pages. In many cases, it is dramatically complex to build a concrete connection among these data sources and make them consolidate each other.

- With the large quantities of data and the complicated processing over heterogeneous data sources, it becomes a difficult problem to understand the user information needs and present retrieval results properly and efficiently.

  The HCI (Human Computer Interface) has been found as a hard problem in fusion systems. The difficulty comes from two aspects. First, the fusion systems should have a *source-independent* interface for end users. In other words, users are not required to know the detail about data sources. For example, Laser Radar can not detect colors, and CCD camera does not work at night or in a bad weather. A uniform interface is needed which can automatically adapt to these kinds of restrictions without any effort from users. Second, the fusion systems should have a *fusion-specific* interface for end users. The systems may be quite different from traditional information retrieval systems, users may need to know some information such as "Did results from data sources conflict with each other?". It requires that the system can present the detail about data sources in retrieval process. The two aspects seem to be controversial with each other. In some cases, it is very hard to find a solution which can reconcile the both.

1.3 STATE OF THE ART

I begin this section with reviewing the most widely used information fusion model – JDL (Joint Directors of Laboratories) model (section 1.3.1), where fusion-related functions are generalized and categorized into several layers. Then state of the art of the research at each layer is described in detail respectively (section 1.3.2).

1.3.1 The JDL Model

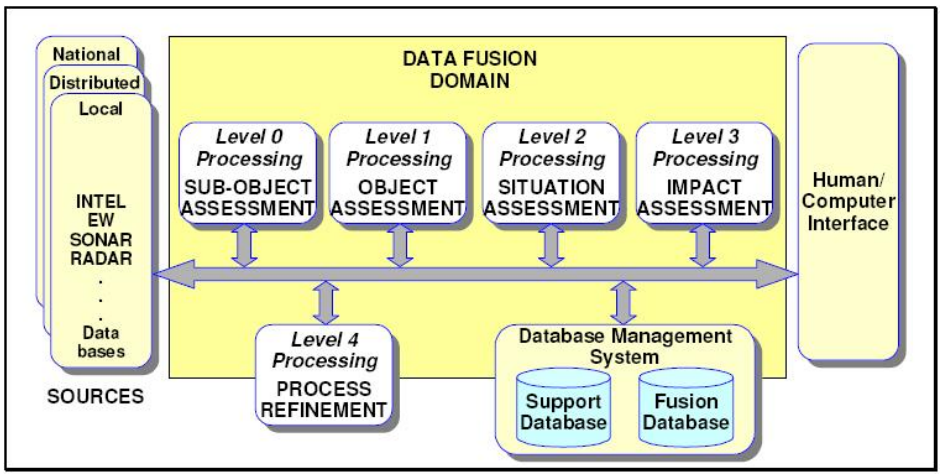

Figure 3: The 1998 Revised JDL Fusion Model [SBW04]

Due to the popularity of fusion applications (section 1.1) and the challenges (section 1.2), information fusion system has been an active research area since early 1970's. Aiming at an overall architecture capable of addressing diverse information fusion systems, F. White proposed the well known JDL model for general fusion system based on the applications developed by DOD (Department of Defense) in 1988 [Whi88]. Then the JDL model has been widely used in the fusion community to categorize the fusion-related functions and systems. Since the model was originally conceived from the military applications, with the dramatically increasing popularity of non-military fusion systems, some of terminologies and concepts (e.g. threat refinement) become not obvious in many applications today. For this

reason, the JDL model has been refined recently in 1998 [SBW98][1].

Figure 3 shows the 1998 revised JDL model. It conceptualized the entire fusion system as three layers: *Sources*, *Data Fusion Domain*, and *HCI*. In the data fusion domain, the fusion processes are categorized into five levels of sub-processes and database management system:

- **Level 0 – Sub-Object Data Assessment**: estimation and prediction of observable object status on the basis of the attribute-level data association and characterization.

  For example, in multisensor surveillance system (example 1.1.1), employing more than one sensors to assess the *color* of a aircraft is a Level 0 process. In local search system (example 1.1.2), a process to determine the *address* of a business entity from multiple Yellow Pages is also a Level 0 process.

- **Level 1 – Object Assessment**: estimation and prediction of the entity states on the basis of object-level association and characterization.

  For example, in multisensor surveillance system (example 1.1.1), employing more than one sensors to assess a aircraft is a Level 1 process. In local search system (example 1.1.2), a process to assess a business entity from multiple Yellow Pages is also a Level 1 process.

- **Level 2 – Situation Assessment**: estimation and prediction of relations among entities.

  For example, in multisensor surveillance system (example 1.1.1), checking with multiple aircrafts to see if they are in a safe distance is a Level 2 process. In local search system (example 1.1.2), a process to identify whether a business is embedded in another one (e.g. a restaurant embedded in a hotel) is also a Level 2 process.

- **Level 3 – Impact Assessment**: estimation and prediction of the effect of the situation of planed actions by participators.

  For example, in multisensor surveillance system (example 1.1.1), if enemy aircrafts are found too close to ours, the possible impact could be that it is prohibited to fire at them from the ground.

[1]The model is proposed in 1998 [SBW98]. The figure 3 is from [SBW04] by the same authors in 2004.

- **Level 4 – Process Refinement** (an element of Resource Management): adaptive data acquisition and processing to support information retrieval request. As an element of resource management, the goal of this level process is to achieve the maximum informative output using the minimum resource.
- **Database Management System**: It includes fusion database and support database.

As mentioned by A. N. Stainberg et al in [SBW98], the JDL model should not read as a processing model e.g. do level 1 fusion first, then level 2, 3, 4. In fact, a fusion system may or may not contain all these levels of fusion. For example, consider the local search system described in example 1.1.2, the level 3 fusion is not obvious since the participators simply don't have any actions.

1.3.2 State of the Art

The technologies related to information fusion systems are rapidly evolving. There is much concurrent ongoing research in each level of fusion processes specified in the JDL model.

The most mature area of information fusion is Level 0 and 1 processing - using multi-source data to determine the features and entity. For example, in intelligent surveillance systems, determining features such as the position and velocity of an object based on multiple sensor observations is relatively an old problem. Gauss and Legendre developed the method of least squares to determine the orbits of asteroids [Sor70]. Generally speaking, many effective fusion methods have been developed according to the type of information to be fused, e.g., aggregation functions on numerical and categorical data, combination operations on probability and possibility distributions, fusion operators in fuzzy sets and so on [Tor03].

Level 2 and 3 fusion (situation assessment and impact assessment) are currently dominated by knowledge-based methods such as rule-based inference systems. These areas are relatively immature [HL01]: there are numerous prototypes, but few robust, operational systems. The rules frequently used on these levels are developed on the basis of the voting method, Bayesian inference, Dempster Shafer's method, and so on [HM04]. Compared to human performance, the current approaches can only provide very limited functions and

there is still much work to do on these levels.

Level 4 processing, which dynamically controls and improves the performance and operation of an ongoing information fusion process, has the least maturity. Frequently, this problem is addressed in the scenario with single data sources (e.g., single sensor operation and control). However, very few feasible approaches are proposed for systems with the multiple heterogeneous data sources, multiple objects, and complex external constraints [HL01].

Finally, HCI in information fusion systems is immature. As studied by Waltz and Llinas [WL90], the overall effectiveness of an information fusion system is greatly affected by the efficacy of the HCI. Unfortunately, advances in understanding of human information needs and how information is processed have not progressed as rapidly. There is still a lot to learn about interaction between human and computer during in this process [HHT01].

According to what we studied about the maturities of current research on different levels in the JDL model, the bottleneck hindering the usability of complex fusion systems are migrating from the low-level fusion processes to the high-level fusion related operations such as knowledge based reasoning, resource management, and HCI.

1.4 OVERVIEW OF OUR APPROACH

The research discussed in this dissertation mainly focuses on the higher level fusion processes (i.e. Level 2-4 and HCI in the JDL model) with the assumption that the lower level fusion algorithms and operations are already available.

Basically, we propose a query-based approach for information retrieval in fusion systems: user information needs are interpreted as queries; the operations of information retrieval and fusion are fulfilled by query processing; the retrieval outputs are presented back to user as query results. Therefore the main subject of my research is to *develop a solution to process the queries effectively and efficiently.*

I first narrowed my study area down to a specific information fusion system – the Intelligent VEhicle Tracking (IVET) System (see chapter 3), where I developed a robust, operational prototype system on the basis of real sensor data and end users. Similar to the

execution plan for query processing and optimization in database theory, *Source Dependency Graph (SDG)* is proposed to formulate the queries in the fusion systems. As illustrated by several examples, the SDG proves to be a powerful tool to generate efficient operation sequences in various situations with multiple data sources. Based upon the SDG, the query refinement and resource management can be performed either by dynamically updating the space of interest (SOI) and time of interest (TOI) or continuously eliminating the unnecessary operation nodes (β – Pruning). Verified by the experiments, the proposed techniques can effectively get rid of redundant data referring to same objects, and consequently elevate the system performance. On the basis of all the work described above, the query optimization problem is formally formulated and proved being a NP hard problem.

In order to achieve the maximum informative results using the minimum time and resources, the response time and resource utilization are analyzed during the entire process of query handling. Query building is identified as the most time consuming process which dominates the whole response time, because freqently the complexity to construct and process a query is accumulated while multiple objects are retrieved from multiple sources. Aiming at resolving this kind of accumulated complexity, I propose the *incremental query* in which a complex information retrieval request is broken down into several iterations. In each iteration, the user constructs an elementary query in a simple and uniform format. Using query operators, multiple elementary queries are composed into a (complex) compound query. System resource can be utilized much more evenly and efficiently by performing the query building and query execution in parallel. Query optimization can be carried out once the final complex query has been developed. Demonstrated by the survey on user query manner and the experiments on the prototype system, the incremental query proves to be a promising approach having many obvious advantages over traditional information retrieval approaches in fusion systems.

When a complex query is broken down into a chain of simple elementary queries connected by query operators, the repeated structures in query processing become much more feasible to be detected and reused. In order to describe the reusable templates of the incremental queries (i.e. repeated structures), a broadly applicable concept *query pattern* is proposed. First the formal definition of query pattern is described. Then I study the typical

tasks in general information fusion systems, and formulate all of them into my query pattern schema. Finally, a *post query reasoner* equipped with pattern matching and learning capability is proposed to apply query patterns into the processing of incremental queries. The matched query patterns can be detected at each time when users finish an iteration in incremental queries. Meanwhile the most frequently used queries can be abstracted and learned as query patterns. Studied from the experiments on the prototype system, the proposed query pattern and its related techniques can effectively facilitate the information retrieval in fusion systems.

1.5 CONTRIBUTIONS OF THE DISSERTATION

Up to the time of this dissertation, many works has been done on the high level fusion processes (i.e. level 2-4 and HCI in the JDL model) towards information fusion systems with a certain degree of intelligence. Several recent approaches advocate the agent based fusion systems where heterogeneous data sources 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 usually depend on ad-hoc, "hardwired" programs to implement them. In general, although information fusion systems have been widely studied in many research domains, it is not clear what is a good way to develop a general solution on the basis of the common characteristics of information fusion process.

- The first contribution of this dissertation is a *broadly applicable query-based approach* for information retrieval in fusion systems.

  Unique from other approaches, our solution offers a uniform framework to handle diverse information needs in various scenarios. Under this framework, a concrete, robust, and operational multi-sensor information fusion system (i.e. the IVET system) has been designed and implemented. It is conceivable that the proposed query processing techniques (e.g. SDG, query refinement and optimization) can be easily applied in many other applications.

- The second contribution of this dissertation is a *novel incremental query method* that can effectively eliminate the accumulated complexity of query processing in the multi-object and multi-source situation.

  For a long time, the usability of fusion systems has been hindered by the lack of an efficient way to interpret user information needs with the collaboration of multiple data sources. The major problem is that the complexity is accumulated and quickly becomes intractable with respect to the complicated retrieval requests. For this reason, I proposed the incremental query process where the information retrieval requests are broken down into serveral simple iterations in a uniform format. The complexity in every phase of query processing i.e. building, refinement, execution, and visualization has been successfully isolated within the single iterations.

- Another major contribution of this dissertation is *the techniques to formulate and apply query patterns*.

  Query pattern, which is usually introduced to handle the reusable templates of user information needs, has been investigated in several query-based information retrieval systems. In our approach, query pattern is a natural extension of the incremental query. Because the query is formulated into a chain of elementary queries and query operators, the repeated structures i.e. query patterns are much more feasible to be detected and reused. A post query reasoner is proposed to match and learn query patterns, which lends itself to a general solution to achieve a certain degree of intelligence in fusion systems.

1.6 RELATED RESEARCH AREAS

In order to eliminate possible confusions between my research domain and the similar ones, namely data management in sensor network (section 1.6.1) and information integration (section 1.6.2), I briefly clarify their differences as well as commonalities in this section.

1.6.1 Data Management in Sensor Network

Our research is different from data management in sensor network, although both deal with data from multiple sources. A sensor network consists of a large number of sensor nodes, in which individual sensor nodes are connected to other nodes in their vicinity through a wireless network [PK00]. While in our research, data sources may or may not be sensors. Even in the case they are sensors, all these sensors are supposed to be connected with different sensor analysis systems, and all information will be fused into central computation nodes. On the contrary, in most sensor networks, there is no assumption for such central nodes. Besides this, sensor nodes in sensor network often refer to a specific kind of sensors, which have limited computation and storage capabilities: a node has a general-purpose CPU to perform computation and a small amount of storage space to save program code and data [PK00]. However, the sensors in our research could be any types such as radar, sonar, and so on.

Of course, information fusion is also a problem in sensor network: since sensor data might contain noise, fusion from several sensors could help to get more accurate and complete result. However, it is far less important than other problems. For example, since sensors in sensor network are usually not connected to a fixed infrastructure, data routing and collecting turns out to be a crucial problem [YG03]. While sensors use batteries as their main power supply, the preservation of power becomes another important problem [SVGM00]. Obviously in a sensor network, all above problems have much more significance than fusion problem.

It is worth noting that there also are some query techniques employed in information retrieval in sensor networks [YG03]. However, these techniques mainly focus on the problems about data collection from sensor nodes and preservation of energy.

1.6.2 Information Integration

Information integration is an active research topic raised recently both by AI and database communities, which currently focuses on combining data distributed on the internet. A simple example is: from hundreds of online book stores, by the information integration approach, the prices of a specific book can be listed together, and the cheapest one can be marked. Much

progress has been made in terms of developing conceptual progress and algorithmic frame works, query optimization, wrap constructing, and object matching [DM03, GMPDQ+97].

It is not surprising that information fusion shares several common issues with information integration, since they both retrieve information from heterogeneous sources. However, one of most significant differences is that the formats of retrieving data and the behaviors of data collection are much more controllable in information fusion systems. In other words, in the fusion applications, the semantics of the inputs are usually already known so that it is easy to be mapped to a central ontology knowledge base. On the contrary, ontology matching and mapping is the biggest challenge in information integration, since various external data sources use different ontology schemas [DMD+03].

More about the comparison between information fusion and information integration is discussed in section of the related work on query system (section 6.2).

1.7 OUTLINE

The rest of this dissertation is organized as follows: the next chapter describes the fusion system and information retrieval problems (i.e. queries) in it, which I consider in this dissertation. Then my major experimental system – intelligent vehicle tracking (IVET) system and basic query processing techniques are described in chapter 3. Chapter 4 explains the incremental query in detail. Next, the query pattern is described in chapter 5. Chapter 6 reviews related research and discuss how our approach advances the state of the art. Finally, the dissertation is summarized, and the future direction is discussed in chapter 7.

Parts of this dissertation have been published in peer-reviewed conferences and journals. In particular, basic query processing techniques (Chapter 3) are partially published in DMS 2002 and 2004 [CDH+02, CCC+04]. Incremental query processing (Chapter 4) is partially published in DMS 2004 [LC04]. Some preliminary work on query pattern and its application (Chapter 5) is partially described in a journal paper in *information fusion* [CJL06]. The application of our approach in a different research domain (i.e. user profiling in e-Learning) is partially published in DMS 2005 [LC05].

2.0 PROBLEM DEFINITION

This chapter defines the information fusion system and query which I consider in this dissertation. First, a variety of information fusion systems are generalized into a common model (section 2.1). Next, the query language that can represent any information retrieval needs in the model is defined (section 2.2). Finally, the metrics for evaluating the query processing are described (section 2.3).

2.1 INFORMATION FUSION SYSTEM

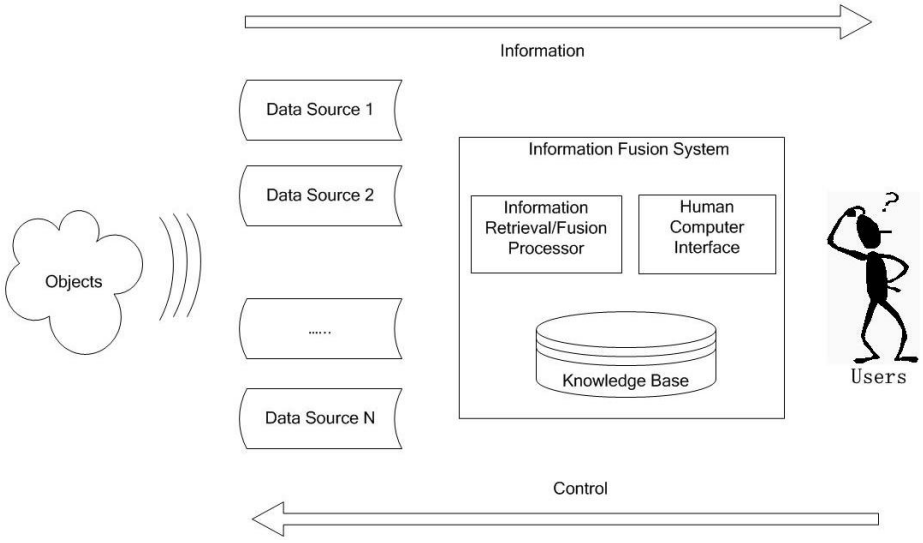

Figure 4: Model of Information Fusion System

The key commonality underlying the diverse fusion systems is that they retrieve informa-

tion referring to same objects on the basis of data from multiple heterogonous data sources. Based upon this point, I generalized the information systems by the model shown in figure 4. It includes the following key components:

- **User**: the client who needs retrieving information about objects from data sources.
- **Objects**: the entities in the real world in which users are interested.
- **Data Sources**: the providers that can supply data about objects. They could be sensors, databases, web resources, and so on.
- **Human Computer Interface**: the interface that connects the user and system. It accepts user information request and presents the retrieval result.
- **Information Retrieval/Fusion Processor**: the module that performs operations towards the goal of information retrieval. It controls the fusion processes, manages system resources, and utilizes the knowledge in knowledge base.
- **Knowledge Base**: the module that stores the supporting knowledge.

The model described above can formulate a variety of fusion systems. For example, both the multisensor surveillance system in example 1.1.1 and the local search system in example 1.1.2 are it instances. Because of its popularity and importance, the subsequent chapters are all discussed under the frame of this model.

2.2 QUERY IN THE INFORMATION FUSION SYSTEM

Under the framework of the fusion system model described in section 2.1, we proposed a query-based approach: user information needs are interpreted as queries; the operations of information retrieval and fusion are fulfilled by query processing; the retrieval outputs are presented back to user as query results.

The query language used here is a SQL-like query language that supports the fusion-related operations, which is called ΣQL. Compared with SQL, it equips with the following new features:

- **Multi-Source Support**: A single object can be marked as being retrieved from multiple data sources by simply putting all the sources in the "FROM" clause. In this case, the fusion operations will be automatically employed if the same objects are observed by multiple data sources.
- **Clustering Support**: A new key word "CLUSTER" is introduced so that the objects sharing the same characteristics (e.g. having the same time stamp) can be clustered. In fact, clustering is a technique originally used in pattern classification to form subsets of similar objects. In information fusion systems, it is employed to eliminate the redundant data referring to same objects.

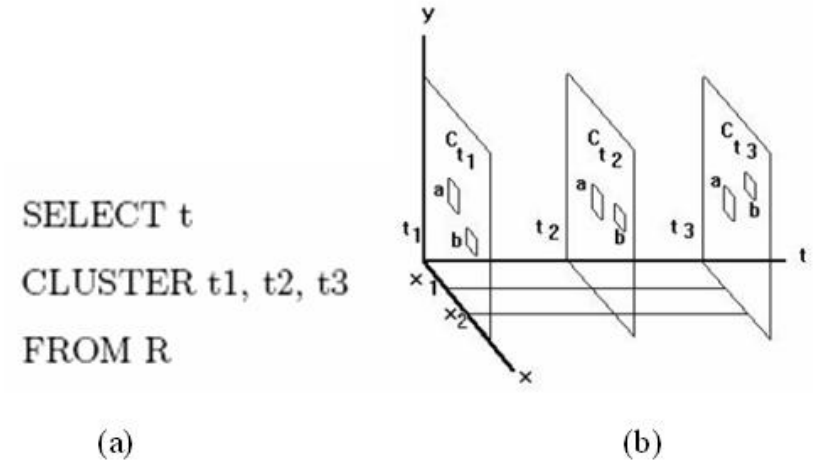

Figure 5: Semantics of "CLUSTER"(a. Query in ΣQL; b. the Semantics)

A simple example is illustrated in figure 5 to explain the semantics of the keyword "CLUSTER". In figure 5(a) a query proposed to simply retrieve three subsets of objects which sharing the same time from data source "R". Shown in figure 5(b), the actual operation is cutting three slices along the time dimension. Noting that any attributes could be set as the dimension, it means the objects can be clustered as any way we need. What follows are two examples of queries in ΣQL:

Example 2.2.1 *In the aircraft surveillance system described by example 1.1.1, user wants to find two F-16 Fighting Falcons which are close enough and both are heading to the north*

with a speed more than 1000 mph in given time and area of interest (i.e. TOI and AOI).

The request can be formulated in ΣQL as follows:

SELECT $object$
CLUSTER ALIAS $object_i, object_j$
FROM
  SELECT t
  CLUSTER *
  FROM $RADAR, EO/IR, ESM$
WHERE $object_i.type =$ "$F16FF$" AND $object_j.type =$ "$F16FF$"
    AND $Distance(object_i, object_j) < \delta$
    AND $object_i.direction =$ "$north$" AND $object_i.speed > 1000$
    AND $object_j.direction =$ "$north$" AND $object_j.speed > 1000$
    AND $object_i.t = t_{given}$ AND $Inside(object_i, AOI)$
    AND $object_j.t = t_{given}$ AND $Inside(object_j, AOI)$

This is a typical sensor-based temporal/spatial query. The data from sensors can be organized by any order such as time or location. First, the data is retrieved by the sequence of time:

SELECT t
CLUSTER *
FROM $RADAR, EO/IR, ESM$

Then the data is clustered by objects. Two objects in the time slice of t_{given} are retrieved based on the criteria described in *where* clause. The fusion operations are employed automatically whenever the observations from different sensors are associated with same objects. The threshold δ is determined by the knowledge base that interprets the term "close enough" based on the context.

Example 2.2.2 *In the local search system described by example 1.1.2, user wants to find a Starbucks Cafe within 5 miles of a given address, which has a public parking lot nearby.*

The request can be formulated in ΣQL as follows:

SELECT $object$

CLUSTER ALIAS $object_i, object_j$

FROM $YP_1, YP_2, \cdots, YP_n$

WHERE $object_i.title =$" Starbuck Cafe"

AND $object_j.category =$"Public Parking Lot"

AND $Distance(object_i, address_{given}) < 5$

AND $Distance(object_i, object_j) < \delta$

The data is first clustered by objects, which is business entities in this example. Then the objects are retrieved based on the criteria described in *where* clause. Similar as the example 1, fusion operations are evoked automatically, and the threshold δ is determined by the knowledge base that interprets the term "nearby" based on the context.

2.3 EVALUATION OF QUERY PROCESSING

The main goal of my research is to process queries (described in section 2.1) in the information fusion systems (defined in section 2.2) accurately and efficiently. In our approach, the following metrics are employed to evaluate the performance of the query processing:

- **Correctness**

  The accuracy of information retrieval is a crucial issue in fusion systems. Generally speaking, the potential errors mainly come from two aspects: the uncertainty caused by low level recognition/fusion algorithm, and the inappropriate knowledge-based reasoning. From the view of the high level fusion processes, the correctness is a fundamental requirement. In other words, there is no compromise solution on the correctness at this level. In our approach, we assume that the low level processes are trustworthy. The correctness of high level processes can be manually verified based upon observation on the output of low level fusion/recognition operations. In the experiments described in the following chapters, the correctness of query processing is primarily verified manually.

- **Processing Time**

  Processing time is a major metric to evaluate a query processing. In our approach, the processing time is defined as the time from the users starting to give their requests until they get the correct results. It is slightly different from the traditional response time, which is from the time the users submit requests until they get the correct results. The rationale behind this difference is that unlike the traditional query approaches, we take the users and HCI as active subjects in our research.

- **System Resource**

  Although our experiments are all simulated on PC, the real fusion systems are frequently used in the scenarios with various constraints on system resources, e.g., the portable devices with limited computing power and memory. So the utilization of system resources is another major metric to evaluate a query processing. The main goal is to achieve the maximum informative results using the minimum resources. In practice, we monitor the usage of the system resources i.e. CPU and memory in our query processing experiments.

2.4 SUMMARY

In this chapter, I defined my main research subject – query processing in information fusion systems. The information fusion systems in which we are interested are described by a general fusion model. The user queries in the model can be formulated by ΣQL, which is a SQL-Like query language supporting fusion operations. Several criteria are also discussed to evaluate performance of the query processing.

3.0 IVET QUERY SYSTEM

Intelligent VEhicle Tracking (IVET) System is our major experimental system, not only because it is a joint project with FOI (Swedish Defense Agency) where the real sensor data and end users are relatively easy to access, but also it is an intelligent surveillance system which is one of the most important and widely investigated information fusion systems.

In this chapter, the IVET system is described. Since it is a complex system involving many aspects of the ongoing work in our research group, I only describe the components related to my work in detail. First, the system is overviewed (section 3.1). Next, the *Ontology Knowledge Base* (OKB), which is closely related to my work, is described (section 3.2). Then I propose basic query processing techniques (section 3.3 – section 3.5). Finally an experimental system and the experiments on it are described to demonstrate the usability of the proposed techniques (section 3.6).

3.1 SYSTEM OVERVIEW

The IVET system is a sensor-based information fusion system where data from different types of sensors, including laser radar (LR), infrared video (IR, similar to video but generated at 60 frames/sec), and CCD digital camera are fused to keep track of vehicles in a given ground area. Figure 6 shows some sample images captured by these three kinds of sensors. Although they were aimed at a same place, the images look quite different (i.e., heterogeneous data inputs).

In a preliminary analysis of the above described sensor data, it is found that data from a single sensor always yields poor results in object recognition. For instance, the target

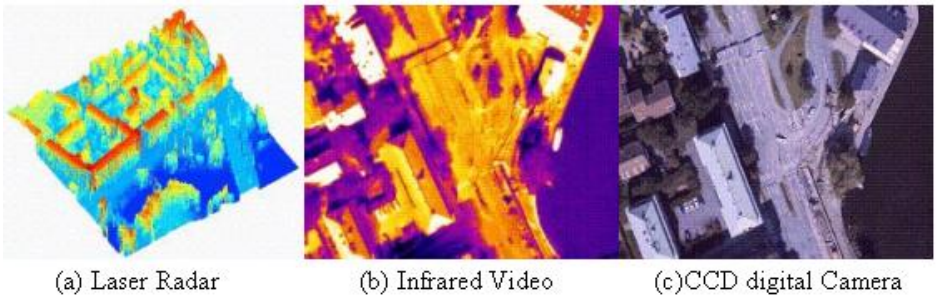

Figure 6: Images from Different Types of Sensors Observing Same Ground Area

object may be partially hidden by an occluding object such as tree, rendering certain type of sensor ineffective. Therefore, it is of great importance to fuse the data from as many types of sensors as possible.

The diagram of the IVET system is shown in Figure 7. For each source (laser radar, infrared video, and CCD digital camera), there is a specific image analysis system hosting recognition algorithms to retrieve object from the raw data. User requests are translated into queries through user interface. Query is basically interpreted into an operation sequence (i.e. execution plan), and the result will be returned when all the operations are finished.

As an overview, the IVET system is a typical fusion system following the classic JDL model (Figure 3). Here I list the main subsystems as well as their counterparts in the JDL model.

- **User Interface** accepts the input of users and presents the query results (i.e., HCI in the JDL model).
- **Query Interpreter** generates the query execution plans on the basis of the user requests (i.e., level 4 process in the JDL Model).
- **Knowledge System** is a logical inference system based on the ontology knowledge base, which refines the query execution plan (i.e., level 2 and 3 processes in the JDL model).
- **Fusion Module** performs a pre-installed fusion algorithms which are determined by the query interpreter and knowledge system (i.e., level 1 process in the JDL model).

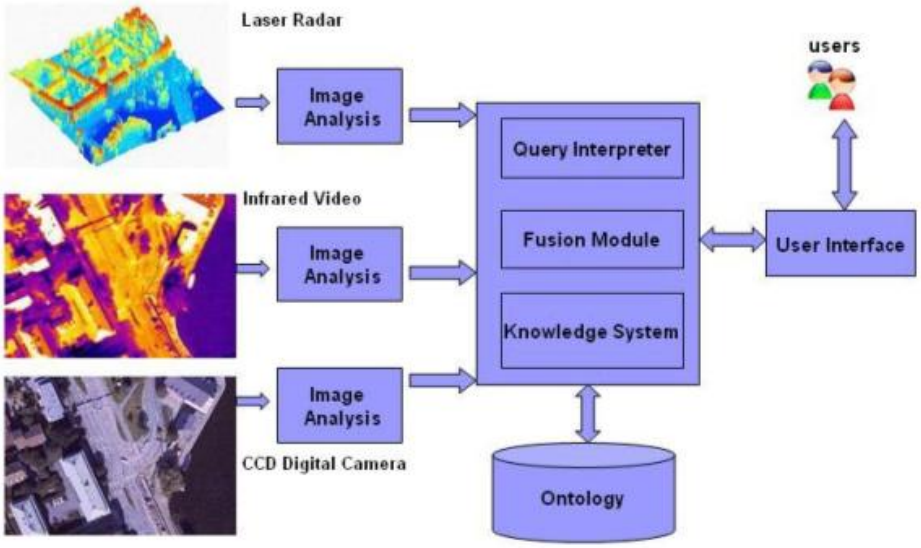

Figure 7: Diagram of the IVET query system

- **Image Analysis System** is an image analysis system specialized for each data source which performs low level object recognition (i.e., level 0 process in the JDL model).
- **Ontology Knowledge Base** encodes a shared understanding among different subsystems.

3.2 ONTOLOGY KNOWLEDGE BASE

Because the information fusion systems are mainly dealing with heterogeneous data from multiple sources, the ontology knowledge that encodes a common understanding cross data sources plays a crucial role in the fusion process. In this section, first I explain why the ontology is needed (section 3.2.1). Next, the related ontology approaches in information retrieval systems on heterogeneous data sources are briefly described (section 3.2.2). Finally, the ontology knowlege base in our system is described in detail (section 3.2.3 – 3.2.4).

3.2.1 Role of the Ontology

Generally speaking, the main purpose of ontology knowledge base is encoding a shared understanding in some specific domains, which can be agreed among different parties (people, computers or systems). In the fusion systems, the ontology is introduced to resolve the heterogeneity of data from different sources. For example, a multitude of data sources are employed in a fusion system which aims at a uniform data access for end users. However, the sources use different terminologies to describe same objects. Frequently it is very cumbersome to maintain a common understanding among them unless there is a systematic ontology schema which can establish an efficient connection with various terminologies and definitions.

Basically, the specification of an ontology comprises a vocabulary of terms, each with a definition specifying its meaning. As mentioned by D. L. McGuinness in [McG02], ontologies can be classified into different categories, ranging from general domain independent knowledge to domain specific knowledge:

- **Representation or meta-ontologies** conceptualize the knowledge representation formalisms.
- **Upper-level ontologies** define general-level descriptive terms that form the foundation for knowledge representation. For example, space, time or object are domain independent terms that apply to all domains.
- **Domain ontologies** represent specific knowledge concepts. For example, weapons or missiles are specific terms of the military domain.

In the IVET system, there are all these three kinds of ontology knowledge. A crucial point in practice is that we need to strictly keep all the domain specific knowledge into domain ontology, because for when our approach is applied into different application areas, the changes on the ontology knowledge should only reside within the domain ontology. In other words, the knowledge and related techniques for the general fusion process should be able to keep unchanged across different fusion applications.

3.2.2 Related Research on Ontology

The ontology is widely investigated in fusion community to deal with heterogeneous data sources. In fact, the heterogeneity of data sources can be addressed at the structural, syntactic, and semantic levels. In this context, the ontology is mainly used to construct efficient mappings among data sources to resolve the potential conflicts at all these levels. Frequently the data sources are equipped with their own ontology. In this case, the task is to construct a global ontology which can provide mappings among local ontologies.

In [WVV+01], H. Wache et al. reviewed the ontology solutions of twenty five information retrieval systems on multiple heterogeneous data sources and categorized them into the following three groups:

- **Single Ontology Approach**: a single global ontology provides a shared vocabulary for the specification of data. In this case, all information sources are directly related to the global domain ontology. For example, SIMS model proposed by Y. Aren et al. in [AHK98] employs a single ontology to construct the semantic mapping among data sources.
- **Multiple Ontologies Approach**: the semantics of a data source is described by its own ontology. Because there is no common vocabulary, the inter-ontology mapping is needed to identify semantically corresponding terms of different source ontologies. For example, the early version of OBSERVER system proposed by E. Mena et al. in [MIKS00] manages local ontologies, and provides mechanisms to map different ontologies without an uniform global schema.
- **Hybrid Approach**: similar as multiple ontologies approach, information sources are described by local ontologies. However, a global ontology is constructed to describe the semantics of the common terminologies among local ontologies. For example, in the KRAFT system proposed by A. Preece [PHG+99], the data sources are handled by autonomous agents equipped with various local ontologies, a global shared ontology is constructed to build an efficient mapping among these local ontologies.

The detail comparison about these three ontology approaches is discussed in [WVV+01]. Generally specking, the single ontology is the simplest one, but with the least flexibility.

The multiple ontologies approach could have the most flexibility since there is no need to maintain a set of common terms. However, in practice, usually it is extremely difficult to construct an efficient connection without a global agreement among data sources. From this point of view, the hybrid approach constitutes a good compromise, which is also used in our IVET system. In our system, the data sources could have their own ontology; however, they are all mapped to a global, shared ontology knowledge base. What follows is a detail description of the global ontology knowledge base used in the IVET system.

3.2.3 The Representation of Ontology Knowledge

The ontology knowledge, which describes the basic concepts and their relations in our system, is represented by a semantic network. In fact, semantic network is used in a wide range of applications for knowledge representation, the more detail about it can be found in [Rus03]. To keep the reasoning as simple as possible, all concepts are organized in a relative simple tree based infrastructure instead of a complex graph, which is called Ontology Knowledge Base (OKB). Each concept is a node in the OKB. The arc in the OKB tree represents *subtype* relation between concepts, denoted as *IS_A*. The rules are represented in first order logic. The formal definitions are described as follows.

Definition 3.2.1 (Ontology Concept) *A ontology concept* OC *is defined as a 3 tuple:*

$$OC =< term, parent, description >$$

In which,

- ***term*** *is the name of OC;*
- ***parent*** *is the super concept of OC, i.e.* $IS\_A(OC, parent)$*; if parent is empty, denoted as EMPTY, it means that OC represents the root node of OKB tree;*
- ***description*** *is the description about OC;*

Definition 3.2.2 (Ontology Knowledge Base) *An ontology knowledge base OKB is a set of ontology concepts in which the parents of all concepts are either in OKB or empty, i.e.*

$$\forall OC \in OKB, IS\_A(OC, P) \Longrightarrow (P \in OKB)\ OR\ (P \equiv EMPTY)$$

Literally, the ontology knowledge base contains a complete closure of concepts in tree structures. For more detail about the representation schema of the ontology knowledge in our system, please refer to [CCC+04, HJF03].

3.2.4 The OKB in the IVET system

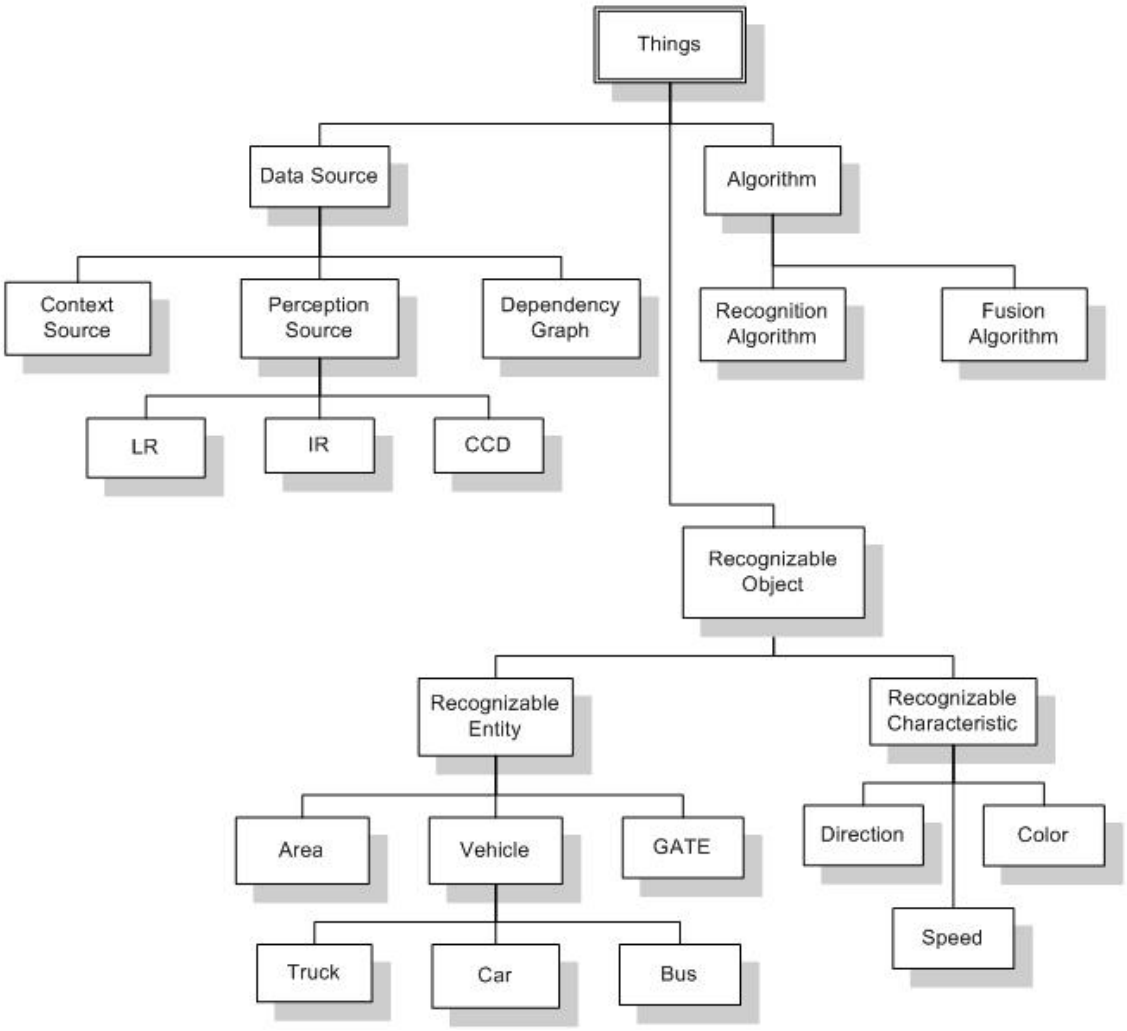

Figure 8: A Portion of the OKB

A portion of the OKB in our system is simplified and illustrated in figure 8. All the concepts are organized 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. The detail about OKB structure is explained as follows:

- 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 8), which corresponds to a certain sensor data type such as CCD, IR, and LR.

- Context Source

  The background or proper context in which an object may occur can be seen as a data source used to enhance the outcome of the query.

- 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, which will be explained in detail in section 3.3.

- Perception Source

  This concept models data type that a source 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, LR, and so on.

- Algorithm

The algorithms must have a name that identify them unambiguously, e.g., "FusionAlgorithm12", or "RecognitionAlgorithm17". It includes two type of algorithms:

1. Recognition Algorithm

   The aim of the recognition algorithms is to detect one or more 'characteristics' in the data types to be processed. Every characteristic belongs to the concept "Recognizable Characteristic".

2. 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. Similar as recognition algorithms, fusion algorithms have a set of "Recognizable Characteristics" that represent what the algorithm can recognize from the input data.

- Recognizable Object

  The concept of recognizable object models everything that the system can detect. It includes recognizable entity and recognizable characteristics.

- Recognizable Characteristics

  The concept of recognizable characteristics models the set of characteristics (or events) that the data fusion and recognition algorithms can detect. For example, it includes 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), and so on.

- Recognizable Entity

  This concept models the entities that can be recognized by the recognition and fusion algorithms. In our system, recognizable entities further divided into area, vehicle, gate, and so on.

A more complete description of the OKB in our system can be found in [CCC+04].

3.3 SOURCE DEPENDENCY GRAPH

In this section, the source dependency graph (SDG) is described, which is a basic tool to facilitate fusion-based query processing and optimization. First, the definition of the SDG is described (section 3.3.1). Then, a query processing example is presented to illustrate how the SDG works (section 3.3.2).

3.3.1 Definition of the SDG

In database theory, query processing and optimization are usually formulated with respect to a query execution plan where the nodes represent the various database operations to be performed [JC84]. In fusion-based query processing, a concept similar to the query execution plan is introduced. It is called the source dependency graph, which is a graph in which each node P_i has the following parameters:

- obj_type_i is the object type to be recognized
- $source_i$ is either the information source or an operator for fusion, union or combination
- $recog\_alg_i$ is the object recognition/fusion algorithm to be applied
- $time_i$ is the estimated computation time of the recognition/fusion algorithm in seconds
- $recog\_cr_i$ is the certainty range [min, max] for the recognition of an object
- $norecog\_cr_i$ is the certainty range [min, max] for the non-recognition of an object
- soi_i is the space-of-interest for object recognition/fusion (usually an area of interest)
- toi_i is the time-of-interest for object recognition/fusion (usually a time-interval of interest)

These parameters provide detailed information on a computation step to be carried out in fusion-based query processing.

If the computation results of a node P_1 are the required input to another node P_2, there is a directed arc from P_1 to P_2. Usually we are dealing with source dependency trees where the directed arcs originate from the leave nodes and terminate at the root node. The leave nodes of the tree are the information sources such as laser radar, infrared camera, CCD camera and so on. They have parameters such as (none, LR, NONE, 0, (1,1), (1,1), sqoi, tqoi, soiall,

toiall). Sometimes we represent such leave nodes by their symbolic names such as LR, IR, CCD, etc. The intermediate nodes of the tree are the objects to be recognized. For example, suppose the object type is 'truck'. An intermediate node may have parameters (truck, LR, recog315, 10, (0.3, 0.5),(1,1), sqo_i, tqo_i, soi_{all}, toi_{all}). The root node of the tree is the result of information fusion, for example, a node with parameters (truck, ALL, fusion7, 2000, (0,1), (0,1), sqo_i, tqo_i, soi_{all}, toi_{all}) where the parameter ALL indicates that information is drawn from all the sources.

3.3.2 Query Processing Example

Query processing in fusion systems can be accomplished by the repeated computation and updates of the source dependency graph. During the each iteration one or more nodes are selected for computation. The selected nodes must not be dependent on any other nodes. After the computation, one ore more nodes are removed from the source dependency graph. The process then iterates until there is no nodes in the tree. What follows is a simple example to illustrate how it works.

Example 3.3.1 *Query: Find a truck in given time and area from laser radar, infrared camera, and CCD camera.*

The corresponding query can be formulated as follows:

SELECT object
CLUSTER ALIAS OBJ1
FROM
    SELECT t
    CLUSTER *
    FROM LR, IR, CCD
WHERE OBJ1.type = 'truck' and OBJ1.time = t_{given} and Inside(AOI, OBJ1)

By analyzing the initial query, the source dependency graph T1 is constructed (Figure 9), where the sources are laser radar (LR), infrared (IR) and charged couple device camera (CCD). Next, we select some of the nodes to compute. For instance, all the three source

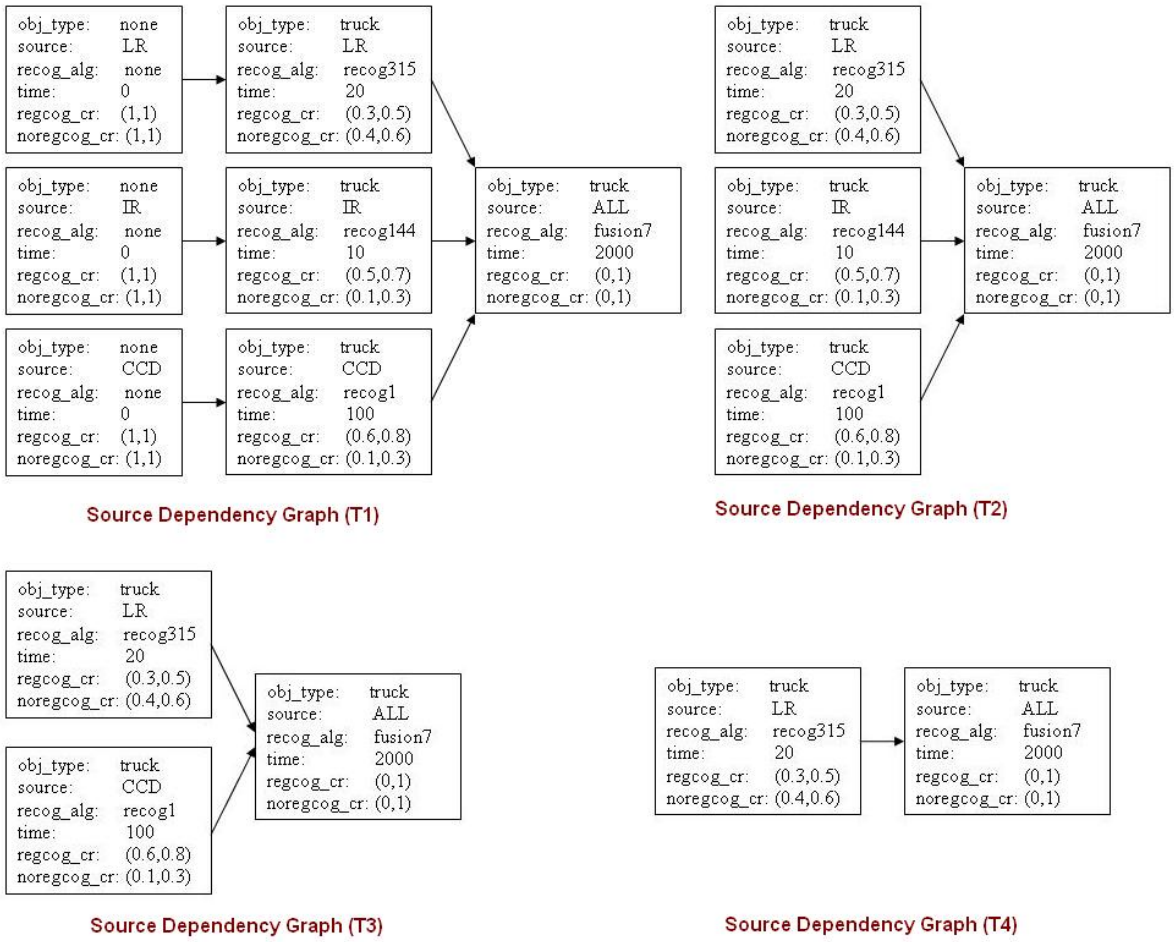

Figure 9: Source Dependency Graph

nodes can be selected, meaning information will be gathered from all three sources. After this computation, the processed nodes are dropped and the updated source dependency graph T2 (Figure 9) is obtained.

We can then select the next node(s) to compute. Since IR has the smallest estimated computation time, it is selected and recognition algorithm 144 is applied. The source dependency graph T3 is shown in Figure 9.

In the updated graph, the IR node has been removed. We now select the CCD node because it has much higher certainty range than LR and, after its processing, select the LR

node. The source dependency graph T4 is shown in Figure 9.

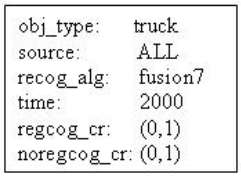

Figure 10: Source Dependency Graph (T5)

Finally the fusion node is selected. The graph T5 has only a single node shown in figure 10. After the fusion operation, there are no unprocessed (i.e., unselected) nodes, and query processing terminates.

3.4 QUERY REFINEMENT

In database research, query refinement is one of the most widely investigated problems, and many mature techniques can also be used in our system. In this section, I briefly describe two basic, fusion-based query refinement techniques on the source dependency graph – the dynamic SOI/TOI (section 3.4.1) and β-Pruning (section 3.4.2).

3.4.1 Dynamic SOI and TOI

Most fusion systems deal with the temporal/spatial data, where user information needs usually come up with a specific range of space or time. As discussed before, they are denoted as space of interest (SOI) and time of interest (TOI). Because in fusion systems multiple data sources describe overlapping sets of objects, frequently there is a large amount of redundant data. The dynamic SOI/TOI is an important technique which can effectively get rid of uninterested data. The main idea is: in the source dependency graph, the SOI/TOI of the unprocessed nodes can be dynamically updated based on the result of processed nodes. For example, in figure 9 from T2 to T3, the node "recog144" was selected to compute. It

applied on the full range of SOI/TOI, and detected that only specific SOI/TOI ranges contain meaningful objects. When it is finished, using dynamic SOI/TOI technique, the SOI/TOI of unprocessed nodes in T3 will be updated to these specific ranges at run time. As a result, the following computation will be performed within the smaller ranges of space and time. In many cases, this technique can significantly improve the performance of query processing.

3.4.2 β – Pruning

The β-pruning also comes from a straightforward idea. Because in fusion systems information referring to same objects are retrieved from multiple sources, there is a large mount of repeated computation. For example, when a query as simple as "find a truck" is applied on three data sources, the same retrieval process has to be performed for three times on different sources. In source dependency graph, these repeated computations are represented by nodes with all the same specifications except the data source. A group of this kind of nodes is named *node cluster*. All the nodes in a node cluster are specified to retrieve same objects but from different sources. The first processed node in a *node cluster* is named α *node*, the rest nodes are β nodes. This could change dynamically: after the α node is computed, it will be removed from the cluster and the next node to process will become α node again.

The rule of *beta*-pruning is: if the α node returns the object with a certainty above a threshold, the rest nodes i.e. β nodes will be pruned from the source dependency graph without any computation. For example, in figure 9 from T2 to T3, the nodes "recog315", "recog144", and "recog1" are in one node cluster. The node "recog144" was α node since it is first computed. Suppose that it returns the "truck" objects with a certainty higher than the pre-defined threshold. According to the rules of β-pruning, the β nodes "recog315" and "recog1" will be directly remove form the source dependency graph.

Literally, the β-pruning can be interpreted this way: for a arbitrary information retrieval problem in fusion systems, if one data source can provide some information for sure, there is no need to use other sources to retrieve the same thing any more. In many cases, this technique can significantly eliminate unnecessary computations.

3.5 QUERY OPTIMIZATION

In the previous sections we explained the query processing steps and query refinement techniques. In this section the optimization problems related to query processing are described.

Suppose that we have the source dependency graph such as T1 of section 3.3. For the recognition algorithm recog315, we have the following certainty range: $P(recog315 = yes|X = truck, Y = LR) = (0.3, 0.5)$, and $P(recog315 = no|X = truck, Y = LR) = (0.4, 0.6)$, where X=truck, Y=LR means that there is a truck in the frame which is obtained by LR. If the input data has certainty range (a,b) and the recognition algorithm has certainty range (c,d), then the output has certainty range (min(a,c), min(b,d)). The optimization problem can be stated as follows: Given the source dependency graph, we want to recognize the object 'truck' with the certainty value above a threshold. Our goal is to minimize the total processing time. In other words, the optimization problem is as follows:

Minimize

$$\sum_{j=1}^{N} \sum_{i=1}^{N} \delta_{ij} \cdot T_i$$

where

$$\delta_{ij} = \begin{cases} 1 & \text{if algorithem i run at } j_{th} \text{ oder;} \\ 0 & \text{otherwise.} \end{cases}$$

T_i= processing time of algorithm i.

subject to

$\sum_{i=1}^{N} \delta_{ij} \leq 1$(for j_{th} order, at most one algorithm can run.)

$\sum_{j=1}^{N} \delta_{ij} \leq 1$(for every algorithm, it can be at most in one order.)

Max$(C^N \times \Delta_N) \geq \theta$ (θ is the certainty threshold.)

where

C^1=(c(ALG_1, priori certainty),. . .,c(ALG_N, priori certainty))

$\Delta_1 = (\delta_{11}, \ldots, \delta_{N1})$.

$C^2 = (c(ALG_1, C^1 \times \Delta_1), \ldots, c(ALG_N, C^1 \times \Delta_1))$

$\Delta_2 = (\delta_{12}, \ldots, \delta_{N2})$.

...

It is a 0-1 integer linear programming(0-1 ILP) problem, which has been widely investigated [Sch98, CLRS01]. It has a dual problem – maximize the certainty value for the object truck under the condition that the total processing time is within the time limit. The problem can be formulated as follows:

Maximize $max(C^N \times \Delta_N)$

where

$$\delta_{ij} = \begin{cases} 1 & \text{if algorithem i run at } j_{th} \text{ oder;} \\ 0 & \text{otherwise.} \end{cases}$$

where

C^1=(c(ALG_1, priori certainty),...,c(ALG_N, priori certainty))

$\Delta_1 = (\delta_{11}, \ldots, \delta_{N1})$.

$C^2 = (c(ALG_1, C^1 \times \Delta_1), \ldots, c(ALG_N, C^1 \times \Delta_1))$

$\Delta_2 = (\delta_{12}, \ldots, \delta_{N2})$.

...

subject to

$\sum_{i=1}^{N} \delta_{ij} \leq 1$(for j_{th} order, at most one algorithm can run.)

$\sum_{j=1}^{N} \delta_{ij} \leq 1$(for every algorithm, it can be at most in one order.)

$\sum_{j=1}^{N} \sum_{i=1}^{N} \delta_{ij} T_i < T$(T is the maximum time that we can bear.)

where

T_i = processing time of algorithm i.

In the above optimization problems we have not considered the space of interest soi when we formalize the problem. If we put it in, the formulation of the problem becomes more complicated:

F^1= (t(ALG_1,initial soi),...,t(ALG_N, initial soi))

A^1 =(a(ALG_1,initial soi),...,a(ALG_N, initial soi))

$\Delta_1 = (\delta_{11}, \ldots, \delta_{N1})$.

$F^2 = (t(ALG_1, A^1 \times \Delta_1), \ldots, t(ALG_N, A^1 \times \Delta_1))$

$A^2 = (a(ALG_1, A^1 \times \Delta_1), \ldots, a(ALG_N, A^1 \times \Delta_1))$

$\Delta_2 = (\delta_{12}, \ldots, \delta_{N2})$.

$\ldots$

$\sum_{k=1}^{N} F^k \times \Delta_k$ is the total running time.

Now we need to change the goal function to

Minimize

$\sum_{k=1}^{N} F^k \times \Delta_k$

Note:

$a(ALG_i, soi)$ is the output soi after using algorithm i on the input soi.

$A^1 \times \Delta_1$ is the output soi after using the first algorithm.

$F^k \times \Delta_k$ is the running time of the kth order algorithm.

$t(ALG_i, soi)$ is the running time of the ith algorithm on soi .

The 0-1 integer linear programming (0-1 ILP) problem is a well-known NP-Hard problem [Sch98, CLRS01]. In other words, there is no polynomial time algorithms so far which can be guaranteed to find the minimum processing time or maximum certainty value given the constrains. In practice, we use approximation algorithms with polynomial complexity to provide some semi-optimized results.

3.6 EXPERIMENTAL SYSTEM AND EXPERIMENTS

To investigate the usability of proposed query processing techniques (i.e. SDG, query refinements etc.), an experimental system has been implemented, which can process queries on real sensor data. To distinguish this system from the following experimental systems, I name it system V1.0. I begin this section by the introduction of the experimental system (section 3.6.1). Then the experiment on it is described in detail (section 3.6.2).

Table 1: Experimental System V1.0

Item	Detail
Title	Experimental IVET System V1.0
Goal	A system which can work on the real data and fusion algorithms to process user queries
Challenges	Integrate several components (e.g. low level fusion/recognition, OKB, query interface, etc.) and make them work together.
Designed by	Xin Li

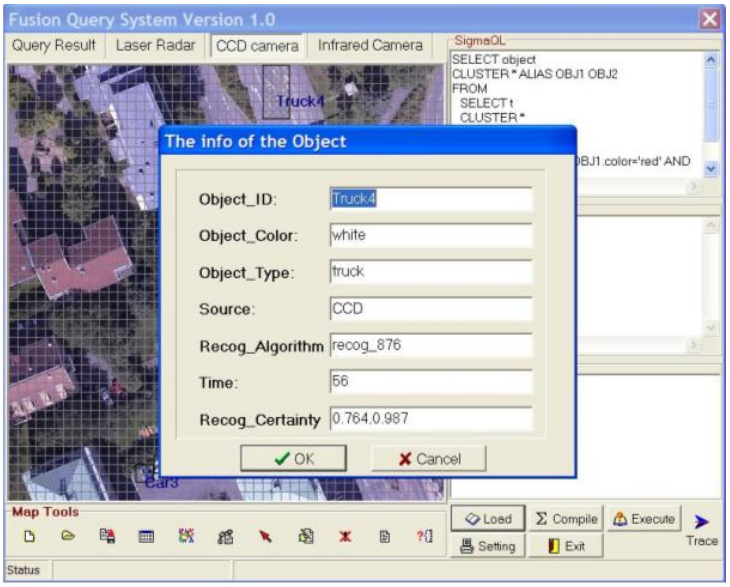

Figure 11: Objects recognized by the recognition algorithms have seven attributes

3.6.1 Experimental IVET System V1.0

Table 1 shows serveral important attributes of the system V1.0. It is the first system in which I have implemented and integrated all components in the system diagram (figure 7). The system can manage various recognition and fusion algorithms for different sensors. The

tree-structured OKB is stored in a database, and a simple knowledge inference system is created based on it. Users can input the queries using a graphic interface. All the techniques described in previous sections are implemented in the system.

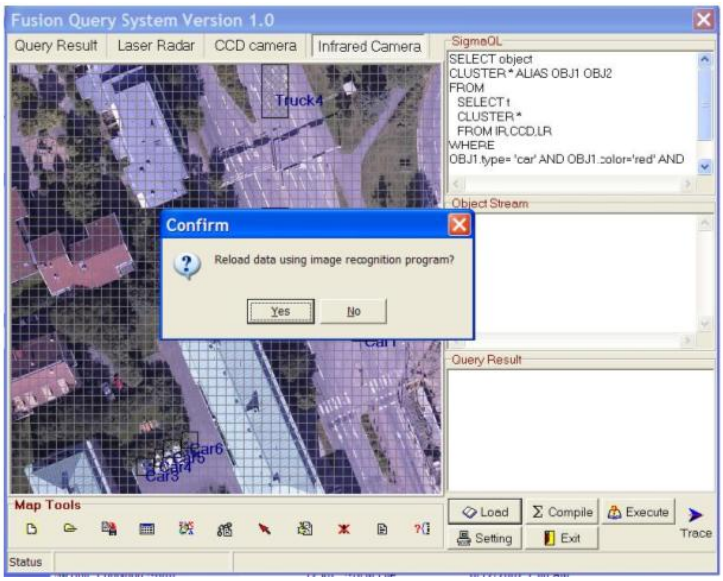

Figure 12: Recognition algorithms can be applied dynamically

The main interface of system is shown in Figure 11. The sensors used are laser radar (LR), infrared video (IR), and CCD digital Camera (CCD).

There are separate image analysis systems working for different sensors. Each time after a recognition algorithm is applied and some objects identified, these objects can be displayed. Each object has seven attributes: object id, object color, object type, source, recognition algorithm, estimated processing time and certainty range for object recognition (Figure 11). There are also hidden attributes including the parameters for the minimum enclosing rectangle of that object, the spatial and temporal coordinates of the query originator, the space of interest and time of interest, and the certainty range for non-recognition of object.

As shown in Figure 12, the recognition algorithms can be applied dynamically to an area in the image, and the recognized objects are displayed. Figures 13 and Figrue 14 illustrate the construction of a query and the results of processing the query, respectively.

The main window in Figure 13 illustrates the visual construction of a query. The user

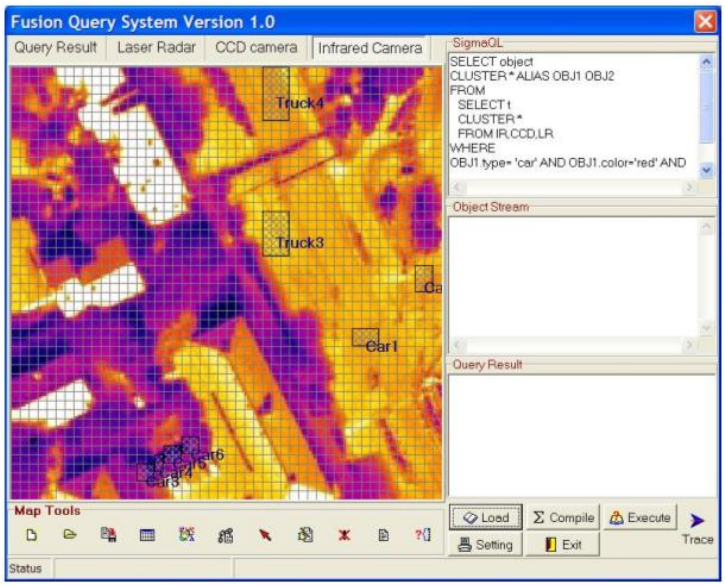

Figure 13: Construction of a query. The resultant query is shown in the upper right window

drags and drops objects and enters their attributes, and the constructed query is shown in the upper right window. The objects in the dependency tree are shown as an object stream in the middle right window.

In Figure 14 the lower right window shows the query results. When a query is being executed, its dependency tree will change dynamically. Figure 15 displays the same information as that of the object stream, but in a format more familiar to end users. It shows the dependency tree on the left side of the screen, and the selected node with its attributes on the right side of the screen. In the next step, both the dependency tree and the query may be changed, as illustrated in Figure 16. As shown in Figure 17, the information of optimization can be shown in a pop-up window.

The original query shown in the upper right window of Figure 13 is as follows:

SELECT object
CLUSTER * ALIAS OBJ1 OBJ2
FROM

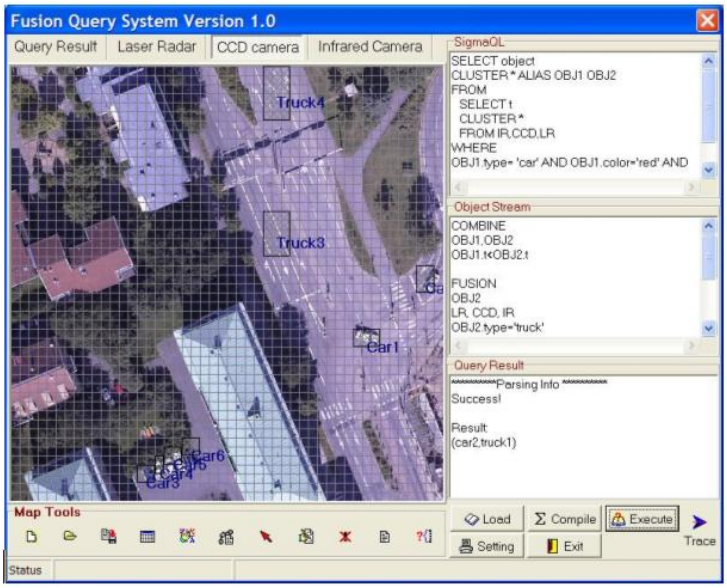

Figure 14: Visualization of Query processing. The result of query was shown on the right bottom window

SELECT t
CLUSTER *
FROM video_source
WHERE OBJ1.type = 'car' AND OBJ1.color = 'red' AND
OBJ2.type = 'truck' AND OBJ1.t < OBJ2.t

The corresponding Object Stream is:

COMBINE
OBJ1,OBJ2
OBJ1.t=OBJ2.t

OBJ2

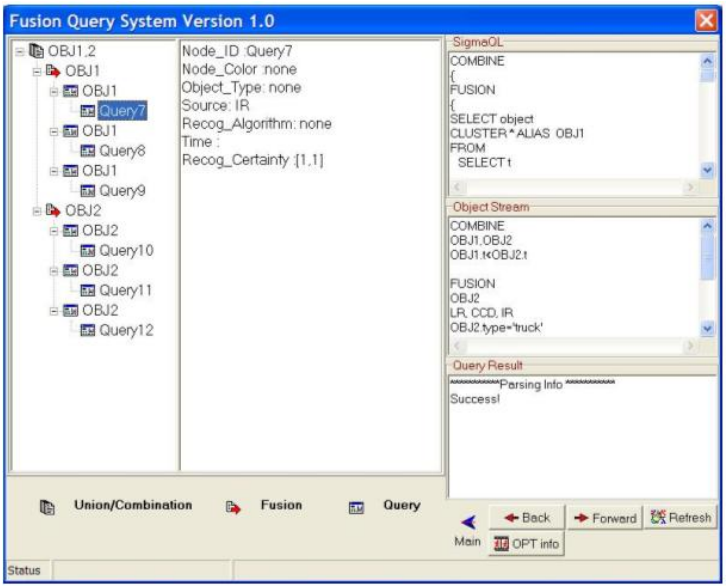

Figure 15: The dependency tree (left) and a selected node (right). We can trace the Query processing step by step

video_source

OBJ2.type = 'truck'

OBJ1

video_source

OBJ1.type = 'car'

OBJ1.color = 'red'

The Result Set is:

Car1, Truck3

Car2, Truck4

i.e., either Car1, Truck3 or Car2, Truck 4 is the retrieved result.

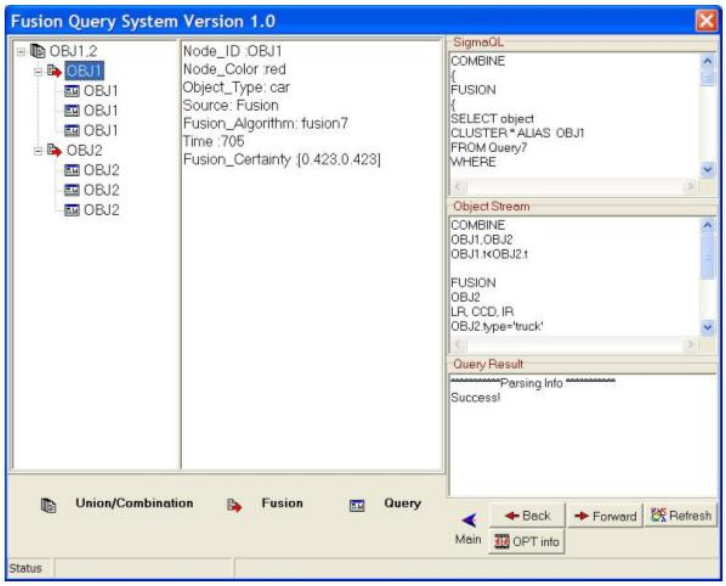

Figure 16: The next step of query processing after the step shown in Figure 15. Both query and dependency tree are changing step by step

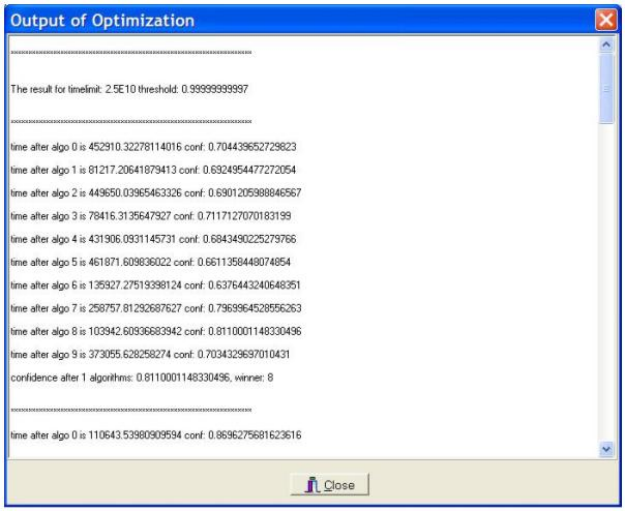

Figure 17: The information of optimization can be shown in a pop-up window.

3.6.2 Experiments on System V1.0

Table 2: Experiment 1

Item	Detail
Title	Experiments on System V1.0
Goal	Verify the usability of the basic query processing techniques (i.e. SDG, Dynamic SOI/TOI, and β – Pruning).
Description	Participators are asked to perform queries expressed in natural language using the experimental system V1.0. Query is interpreted to SQL-like Query by users, and the result is visualized on the map which is easy to be verified.
Designed by	Xin Li
Participators	Eight End Users

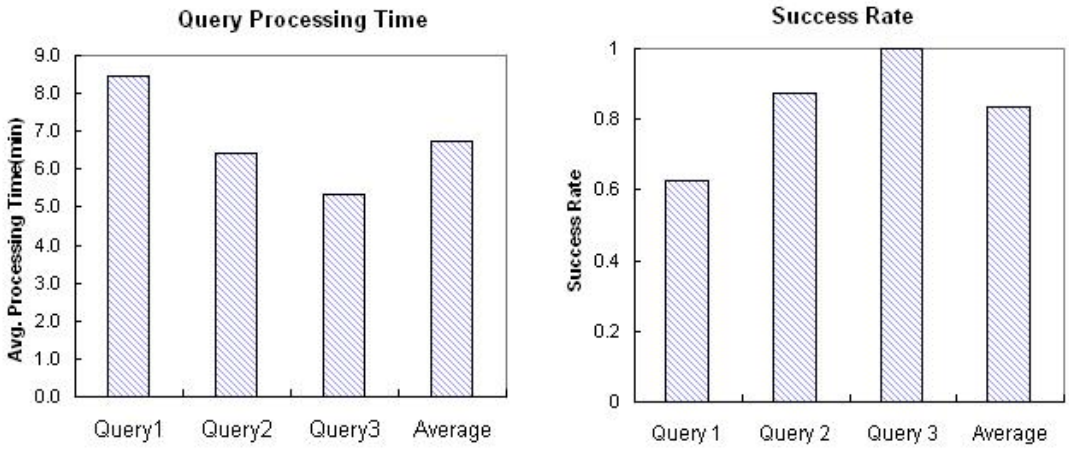

Figure 18: Result Analysis of Experiment 1

In order to analyze the performance of proposed query processing techniques, I designed an experiment on system V1.0. Table 2 lists some important attributes of this experiment. Eight end users have kindly participated in this study. Basically, the experiment includes the following three steps:

- **Step1**: assuming users have some pre-knowledge on SQL language, we give them a very quick tutorial (30 min) on the query language we are using by showing several examples using our system.
- **Step2**: we carefully choose three most frequently issued queries as benchmark queries (see Appendix A), and tell users the queries in English. Then ask them to perform these queries one by one using the experimental system.
- **Step3**: we monitor the time that users spend on each query and the system resource usage during the process.

A threshold δ of response time (i.e. 15 min) is defined: if users can finish a query (from formulating a query until getting a satisfied result) within δ time, it is defined as a successful case, otherwise it is a failure. Figure 18 shows the average processing time and success rate on each benchmark query and the average of all.

Demonstrated by this experiment, the proposed query processing methodology including SDG, query refinements, and optimization prove to be a feasible approach in real fusion systems.

3.7 SUMMARY

In this chapter, I describe our primary experimental system – the IVET system, which is a typical sensor-based fusion system. In particular, the ontology knowledge base in the system, which closely relate to my works, is described in detail. Several important, fusion related query processing techniques such as source dependency graph (SDG), dynamic SOI/TOI, β-Pruning, and query optimization are discussed in detail. Verified by the experimental system, the proposed query processing techniques are feasible and effective in the real fusion system. All these works serve as a solid foundation for our further research.

4.0 INCREMENTAL QUERY PROCESSING

I begin this chapter by analyzing the experiment results described in the last chapter, showing that the bottleneck of query processing in the IVET system is query building instead of query execution (section 4.1). Motivated by this fact, I propose an incremental query method to eliminate the accumulated complexity in query building as well as query execution (section 4.2 – section 4.4). Verified by a survey on user query manner, the incremental query method is more preferable than the direct query method in various scenarios of fusion systems (section 4.5). An experimental system has been implemented to demonstrate the usability of the proposed incremental query method (section 4.6).

4.1 MOTIVATION

To investigate usability of the IVET system, the experiment results on system V1.0 are further analyzed. Figure 19(a) shows the system CPU/memory usage during processing of a complex spatial/temporal query. Depending on the workload, query processing can be divided into two phases, marked as Phase A and Phase B:

- **Phase A**: The characteristics of this phase is low workload on CPU/memory for the system (CPU Usage $< 30\%$; Memory Usage $< 10\%$). The system is almost idle because it is accepting users input, checking syntax of query or doing other light-load tasks.
- **Phase B**: CPU/memory becomes much busier because the system is executing query and visualizing result.

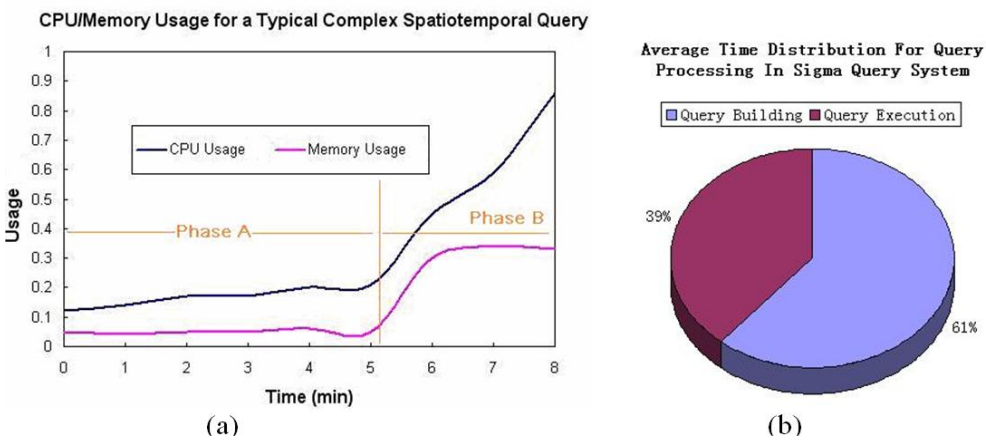

Figure 19: Experiment Result on the System V1.0

Figure 19(b) shows the statistical result from the user's point of view. The *query building* phase begins from inputting a query to submitting a query, followed by the *query execution* phase. Results from the view of the system (Figure 19(a)) and the view of the user (Figure 19(b)) both consistently lead to the following conclusion: for the system the users spend more time building a query than executing a query. In other words query building is the more significant phase in the processing of complex spatial/temporal queries. On the other hand the system is almost idle while the user is composing a complex query.

This explains why the optimization for query execution (section 3.5) does not significantly improve usability of the system. Query execution is actually just a small part of the entire query processing task. Query building is the real bottleneck hindering the overall usability of the fusion system.

To remove the query building bottleneck, query interface plays a crucial role because it is a tool to help a user build complex queries. In fact, query interface is an active research area evolving fast in recent years. As related research, there are mainly two types of query interfaces: textual interfaces and non-textual interfaces. The system V1.0 shows an example of a textual interface, in which a user may input textual queries into the system directly. The advantage of the textual interfaces is that it can support very complex

spatial/temporal queries. However query languages are difficult to learn for ordinary users. Query building is also error prone and time consuming if users have to input textual queries directly. Not surprisingly, there are many non-textual query interface used to decrease users mental load in various areas. For each non-textual query interface, usually there is a visual query language supporting it. Many approaches for visual query languages have been proposed [AEG03, NS02, LC95]. In these approaches the picture elements conveying different meaning are connected together to form a query. For example in [AEG03] a simple query can be represented by a data input element and a data output element connected by a filter symbol. A filter also can connect different simple queries to form a more complex query. In [NS02] iconic symbols that have various meaning are composed by spatial relations (such as horizontal or vertical concatenation) to form a query.

Mentioned by G. Costagliola in [CDOP02], all these approaches can be classified into two groups depending on the connecting styles of elements: connection-based such as [AEG03] and geometric-based such as [NS02]. The foundation of the connection-based query languages is E-R (Entity-Relation) diagram, which is widely used in database design. But unfortunately for naíve users who have no knowledge on of databases and query languages, it is not easy to understand E-R diagrams. The most severe problem for both approaches is: similar to the text query interface, when a query is getting becomes complex, all these diagrams will become much more difficult to build and manage, especially in multi-object and multi-source scenarios.

Aiming to resolve this kind of accumulated complexity in multi-object and multi-source situations, I propose incremental query in which a complex information retrieval request is constructed incrementally through several separated iterations. The more about the incremental query is discussed in detail in the following sections.

4.2 AN EXAMPLE OF INCREMENTAL QUERY

Before I introduce the formal definition of incremental query, an example is described here to illustrate the basic concepts in incremental query such as elementary query, compound

query and query operators. For example, we have the following query:

Example 4.2.1 *Query: find a red car followed by a truck within 10 minutes from Laser Radar and CCD camera.*

As we discussed before, the above request can be formulated into a query based on the fusion operations as follows:

```
SELECT object
CLUSTER ALIAS OBJ1, OBJ2
FROM
    SELECT t
    CLUSTER *
    FROM LR, CCD
WHERE OBJ1.type='car' AND OBJ1.color='red' AND OBJ2.type='truck' AND
          OBJ1.time<OBJ2.time AND OBJ1.Time>OBJ2.time-10min
```

In the above query, the OBJ1 (red car) and OBJ2 (truck) are retrieved from two sources at same time. For end users, frequently multiple objects/ multiple sources make the query formulation become complex. However, it will be much easier to construct if we re-organize the query in the following equivalent way:

```
(2) SELECT object
(2) CLUSTER t ALIAS OBJ1,
(1)     SELECT object
(1)     CLUSTER t ALIAS OBJ2
(1)     FROM LR, CCD
(1)     WHERE OBJ2.type=truck
(2) FROM LR, CCD
(2) WHERE OBJ1.type=car AND OBJ1.color=red AND
(2)     OBJ1.time<OBJ2.time AND OBJ1.Time>OBJ2.time-10
```

Through this way, the query could be *incrementally* built in two iterations. The query for the first iteration is the statements marked with (1) – *selecting all the truck objects from LR and CCD*. The statements with mark (2) are the query for the second iteration. Besides the object, time, space and some properties needed to be specified as same as first step, there is another thing to specify for the second step – the relation to the previous one. In other words, the second iteration can be divided into two parts: one part is building a simple query – *selecting all the red cars from LR and CCD*; the other one is choosing a relation operation to combine these two queries. For users, the relation between different queries could be treated as an operator that connects two queries. In this example, the query operator formulates a temporal relation – *Objects in query (1) appears before Objects in (2) within 10 minutes*.

To summarize, as shown by this example, the incremental query can break a complex query into several iterations. In the each iteration, user only needs to construct a simple query, which is called *elementary query*. Elementary queries can be connected by various relations such as temporal, spatial, and so on. This kind of relation is described by *query operator*. A complex query can be represented by a chain of elementary queries connected by query operators, which is called *compound query*. The formal definitions of these concepts are described in the following section.

4.3 FORMAL DEFINITIONS

In this section, I formally define the concepts of elementary query, query operator, and compound query.

Definition 4.3.1 (Elementary Query) *An* elementary query *EQ is a basic form of spatial/temporal query, which can be defined as a 5 tuple: EQ:= < OBJ, SOURCE, AOI, TOI, PREDICATE >*

- *OBJ is type of object to be retrieved, such as car, hurricane and so on.*
- *SOURCE is data source, such as a video camera.*

- *TOI is time of interest, which is represented by a range of time.*
- *TOI is time of interest, which is represented by a range of time.*
- *PREDICATE is a unary predicate on the objects which specifies constraints on the properties of the objects to be retrieved, such as a cars direction of movement.*

Example 4.3.1 *A simple spatial/temporal query-"find red cars which passed toll gate A from 4:00am to 6:00am from surveillance video" can be represent by an elementary query Q:*

Q=< "car" , surveillance video, toll gate A, 4:00am-6:00am, (object.color = "red") AND (object.moving = TRUE)>

However, a complex query may involve more than one object, so it cannot be simply represented as an elementary query. For example, a query – "find a red car followed by a truck within 10 minutes from a video source given AOI and TOI" cannot be described in terms of an elementary query. But it can be represented as a compound query composed from two elementary queries. What follows are definitions towards a formal description of compound query.

Definition 4.3.2 (Query Operator) *A query operator QO is a binary predicate P on two objects X and Y.*

QO :=P(X, Y)

Query operators can specify relations between objects. With respect to the predicate P, Query operators are categorized into three groups:

1. Temporal query operator if P describes a temporal relation between X and Y.
2. Spatial query operator if P describes a spatial relation between X and Y.
3. Direction query operator, otherwise.

Example 4.3.2 *Query operator τ: τ(object1,object2) = (object1.time > object2.time) AND (object1.time < object2.time + 10min)*

τ is a temporal query operator, which is literally explained as "Object1 appears before object2 within 10 minutes".

Definition 4.3.3 (Compound Query) *A compound query CQ is a form of complex spatial/temporal query which could include multiple objects. It can be defined as a three tuple recursively: CQ :=<q, τ, c> In which,*

- *q is an elementary query.*
- *τ is a query operator.*
- *c is either an elementary query or a compound query.*

Example 4.3.3 *The complex query mentioned before – "find a red car followed by a truck within 10 minutes from a video source by given AOI and TOI"can be represent by a compound query CQ:*

$$CQ = < q1,\ \tau,\ q0 >$$

In which,
q1=<"car", video, AOI, TOI, (object.color="red") AND (object.moving = TRUE) >
τ(object1, object2) = (object1.time > object2.time) AND (object1.time < object2.time + 10min)
q0=<"truck", video, AOI, TOI, object.moving = TRUE >

Using elementary query and compound query, a complex query can be represented as a nested compound query which composed by a chain of elementary queries and query operators. The following function ResultSet reveals the semantic this kind of composition:

Definition 4.3.4 (ResultSet) *A ResultSet is a function on either an elementary query or a compound query to a set of objects:*

$$ResultSet(Q) := \begin{cases} \{object | (object.type = obj) AND (object.source = source) \\ \quad AND (object.location \in AOI) AND (object.time \subset TOI)\} \\ \text{if } Q \text{ is an elementary query, } Q = <obj, source, AOI, TOI, direction>; \\ \\ \{object | (object \in ResultSet(q_0) AND \\ \quad (\exists X \in ResultSet(q_1), \tau(object, X))\} \\ \text{if } Q \text{ is a Compound Query, } Q = < q_0, \tau, q_1 >. \end{cases}$$

Example 4.3.4 *Query: Did any sensor (data source) contribute to the result in any extreme way?*

$Q_0 = <$ *"truck", PerceptionSource, AOI,* $object_i.t_{given}$, *TRUE* $>$

τ*(objectj, objectk) =*

(OR (qualitative-difference($object_k$*.belief-value,*

$object_j$*.belief-value) = "large"*

AND $object_k.position = object_j.position$

AND objectk.t = objectj.t),

($object_k.t = object_j.t$

AND $object_k.position = object_j.position$

AND $object_k.type \neq object_j.type$*))*

$Q_1 = <$ *"truck", Source Dependency Graph (SDG), AOI,* $object_k.t = t_{given}$, *TRUE* $>$

In the above expressions, "TRUE" means that the PREDICATE in Q_0 is always satisfied because no attribute value has been asked for. In Q_1 truck is part of the query. In this case this means there is no change in object type.

4.4 INCREMENTAL QUERY PROCESSING

As discussed above, a crucial problem for information retrieval in multi-object, multi-source scenarios is that information needs are often too complex to be formulated by users. The reason is that complexity of a query is usually accumulated when multiple object types are retrieved from multiple sources in a single query. Using the incremental query schema, every query by users is considered a compound query that can be constructed in several iterations. In the each iteration, the user constructs an elementary query. Using query operators, multiple elementary queries can be composed into a nested query. The composition rule is implicit, and the user does not need to know the underlying query language 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.

Compared to other query approaches, our incremental query method has several evident advantages for fusion systems:

1. Accumulated complexity is reduced because the query building steps are isolated from each other. Each step is concerned only with one object. The query operators are the only connections between consecutive steps.
2. Visualization of the interface becomes much easier because there is no need to display a whole complex query at one time, but just the elementary queries and the query operators separately.
3. System resources such as CPU and memory can be utilized evenly because query building and query execution are divided into small pieces and could be processed in pipeline style.
4. Some repeated structures in a query become easier to detect and reuse, because a complex query turns out to be a long chain of elementary queries and query operators.
5. Query optimization can be carried out once the final complex query has been developed.

Using incremental query, every query issued by users is uniquely identified by sequences of elementary queries and query operators, which is formally defined as *query path*.

Definition 4.4.1 (Query Path) *A query path QP is a sequence of queries*

$$QP := (Q_0, Q_1, \ldots, Q_n)$$

In which,

$$Q_i =< q, \tau, Q_{i-1} >,\ q \text{ is an elementary query and } \tau \text{ is query operator.}$$

Query Path $QP := (Q_0, Q_1, \ldots, Q_n)$ describes the track of an incremental query building from Q_0 to Q_n. In each step, a query operator and an elementary query are applied on the previous one.

Definition 4.4.2 (Complete) *The predicate Complete is a unary predicate on Query Path* $QP := (Q_0, Q_1, , Q_n)$,

Complete(p) $\Longleftrightarrow$ *Q_0 is an Elementary Query, and each Q_{i+1} is constructed from the previous Q_i by applying a query operator.*

A complete query path is a history of incremental query building from an elementary query to a complex query.

Definition 4.4.3 (SubPath) *The predicate SubPath is a binary predicate on Query Path $QP_1 = (Q_0, Q_1, Q_2, \ldots, Q_n)$ and $QP_2 = (Q'_0, Q'_1, Q'_2, \ldots, Q'_m)$, $n < m$*

$$SubPath(QP_1, QP_2) \Longleftrightarrow \exists s, 0 \leq s \leq m-n, \forall i = 0, 1, \ldots, m, Q_i = Q'_s + i$$

Literally, $QP_1 = (Q_0, Q_1, Q_2, \cdots, Q_n)$ is a subpath of $QP_2 = (Q'_0, Q'_1, Q'_2, , Q'_m)$ if only if $(Q_0, Q_1, Q_2, \cdots, Q_n)$ is a subsequence of $(Q'_0, Q'_1, Q'_2, \cdots, Q'_m)$.

The following rules hold with SubPath relation:

$$SubPath(X, Y) \cap SubPath(Y, Z) \Longrightarrow SubPath(X, Z)(transitivity)$$
$$SubPath(X, Y) \cap SubPath(Y, X) \Longrightarrow X \equiv Y(anti - symmetry)$$

Observed from the definitions of the query operator and compound query, some restrictions are applied to our query representation because the query operator is limited to the binary predicate and no memory manipulation capacity is provided along the query path. However, noting the well known trade off between the expressiveness and computational tractability in knowledge representation [LB85], with these restrictions the queries become much easier to handle. Furthermore, this trade-off makes it possible to apply more advanced techniques such as query pattern matching and retrieving into our method.

4.5 SURVEY ON USER QUERY MANNER

With incremental query, now users have two options to create a query: 1) directly input a SQL-like query (e.g. the method used in system V1.0); 2) incrementally build a query using elementary queries and query operators. In order to find out which one is more preferable for users, I have designed an experiment based on the user survey shown in Appendix B. The important attributes of this experiment are listed in Table 3.

Table 3: Experiment 2

Item	Detail
Title	User Query Manner Survey
Goal	Assess the user preference on incremental query and compound query.
Description	Participators are asked to finish a survey shown in Appendix B. Several queries are provided in natural language. Participators are required to formulate them into incremental query (i.e. elementary queries and query operations) and compound query. Time spent on each method is recorded and the preference is surveyed.
Designed by	Xin Li
Participators	Twenty Three Graduate Students in CS3650, University of Pittsburgh.

In the experiment, surveyees are asked to create three queries in fusion systems using the both methods. The time they spent on each one is recorded, and their preferences are explicitly collected.

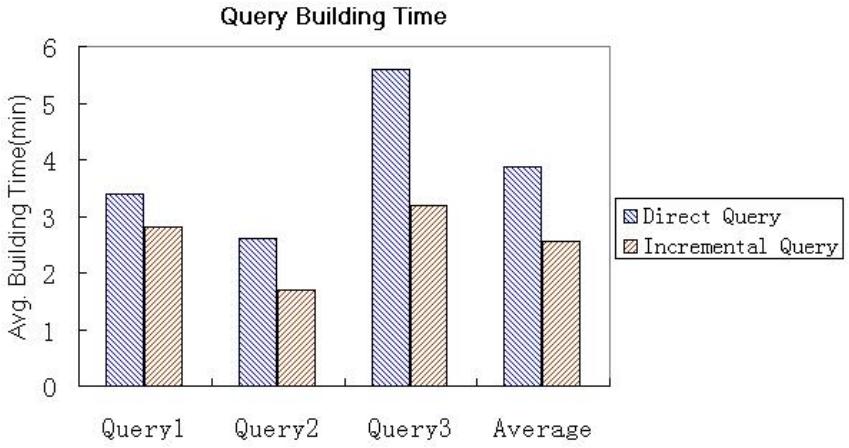

Figure 20: Query Building Time (Direct Query vs. Incremental Query)

Figure 20 shows the average time on three sample queries, it consistently shows that

the incremental query building costs less time than the direct query building. Meanwhile, shown in figure 21(a) the majority of surveyees (52%) also explicitly prefers the incremental query method. As a conclusion, the incremental query proves to be a better way for users to formulate a query in fusion systems.

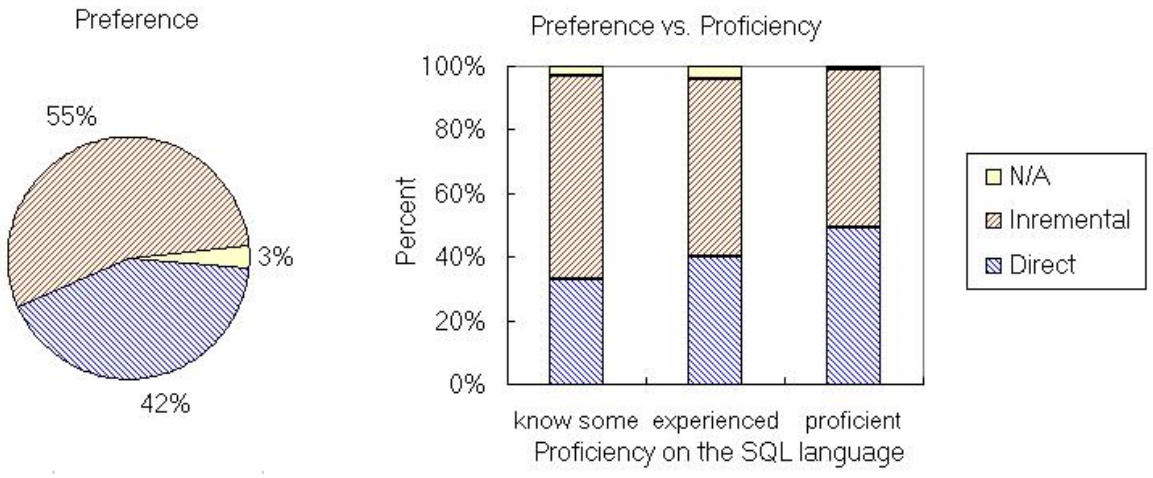

Figure 21: User Preference Survey((a): overall preference; (b): preference vs. proficiency)

The relation between the preference and the proficient on query language (i.e. SQL language) is also studied in this experiment. As shown in figure 21(b): the more proficient on SQL query language (and related concepts) users are, the more preferable on the direct query method they indicate. It is conceivable that for the users who know nothing about query languages the incremental query will be much more preferable than the direct query.

4.6 EXPERIMENTAL SYSTEM AND EXPERIMENTS

To investigate the usability of the proposed incremental query method, an experimental system named V2.0 has been designed and implemented on the basis of system V1.0 (see section 3.6.1). The new system can support the incremental query building (i.e. *elementary query* and *query operator*) through a visual query interface. I begin this section by the

introduction of the system (section 4.6.1). Then the experiment on it is described in detail (section 4.6.2).

4.6.1 Experimental IVET System V2.0

Table 4: Experimental System V2.0

Item	**Detail**
Title	Experimental IVET System V2.0
Goal	An incremental query system based on the experimental system V1.0 (shown in Table 1)
Challenges	Query Interfrace that visualizes elementary query and query operators.
Designed by	Xin Li

Table 4 lists some important attributes of the system V2.0. The main challenge is to design a query interface for the incremental query. The incremental query makes query visualization feasible because the complexity of a query is separated within single iterations. However, it is still a difficult problem to visualize the elementary query and query operators.

The main user interface of system V2.0 is shown in Figure 22. Similar as the system V1.0, on the left of main window there are some objects shown on a map. The red rectangle on the map shows the AOI (Area of Interest) for the current query. As defined in section 4.3, an elementary query has five variables needed to specify: Object, Source, Time (TOI), Space (AOI) and Direction. The Space (AOI) is shown on the map; other four items are on the panel. For the right part of main window, from top to bottom there are elementary query, query operator, and four functional buttons. The buttons are *back*, *query*, *restart* and *forward*. User can switch between different iterations using *back* and *forward* button. The result of current query will be displayed on the map if button *query* is pressed.

When a query operator button (i.e. *temporal*, *spatial*, and *direction*) is pressed, a separate window will be shown to let user create a new query operator. Figure 23 shows the interfaces for temporal and spatial query operators. And also, users can input text to create these operators directly.

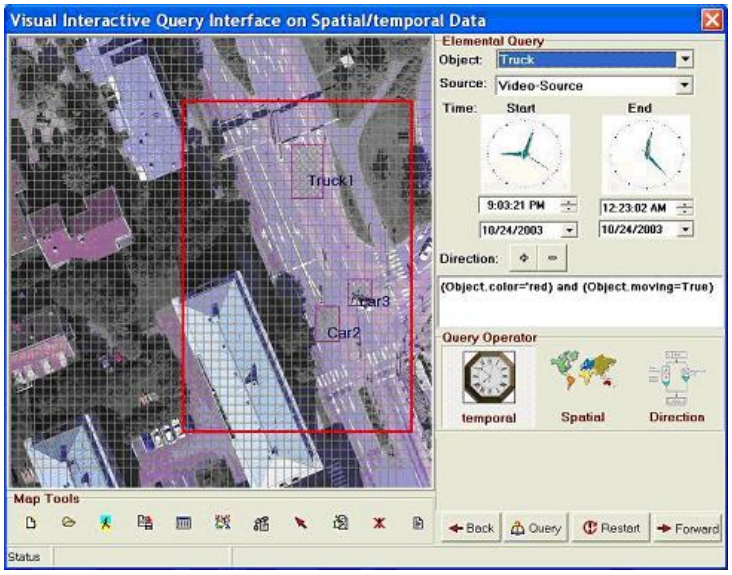

Figure 22: Query Interface of the System V2.0

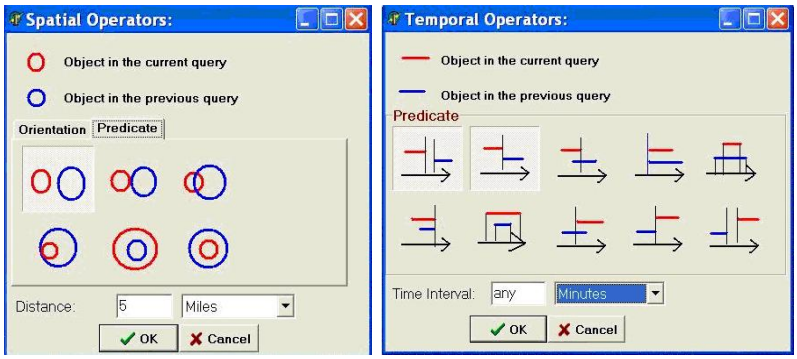

Figure 23: Query Operators

4.6.2 Experiments on System V2.0

In order to analyze performance of the proposed incremental query method, I designed an experiment on system V2.0. Table 5 lists some important attributes of this experiment. Eight end users have kindly participated in this study. Similar as the experiment on system

Table 5: Experiment 3

Item	Detail
Title	Experiments on System V2.0
Goal	Verify the usability of the incremental query processing techniques (i.e. complex query is composed by elmentary queries and query operator).
Description	Participators are asked to perform queries expressed in natural language using the experimental system V2.0. Users are required to input queries incrementally using elementary queries and query operators. The result is visualized on the map which is easy to be verified.
Designed by	Xin Li
Participators	Eight End Users

V1.0 (see section 3.6.2), the experiment includes the following three steps:

- **Step1**: assuming users have no pre-knowledge on incremental query, we give them a very quick tutorial (30 min) by showing several examples using our system.
- **Step2**: we carefully choose three most frequently issued queries as benchmark queries (see Appendix A), and tell users the queries in English. Then ask them to perform these queries one by one using our system.
- **Step3**: we monitor the time that users spend on each query and the system resource usage during this period.

A threshold δ of response time (i.e. 15 min) is defined: if users can finish a query (from formulating a query until getting a satisfied result) within δ time, it is defined as a successful case, otherwise it is a failure. To compare with the performance of the direct query, I plot the time and success rate of this experiment together with the experiment 1 (see table 2) in figure 24. Compared to the direct query method, the average query processing time and

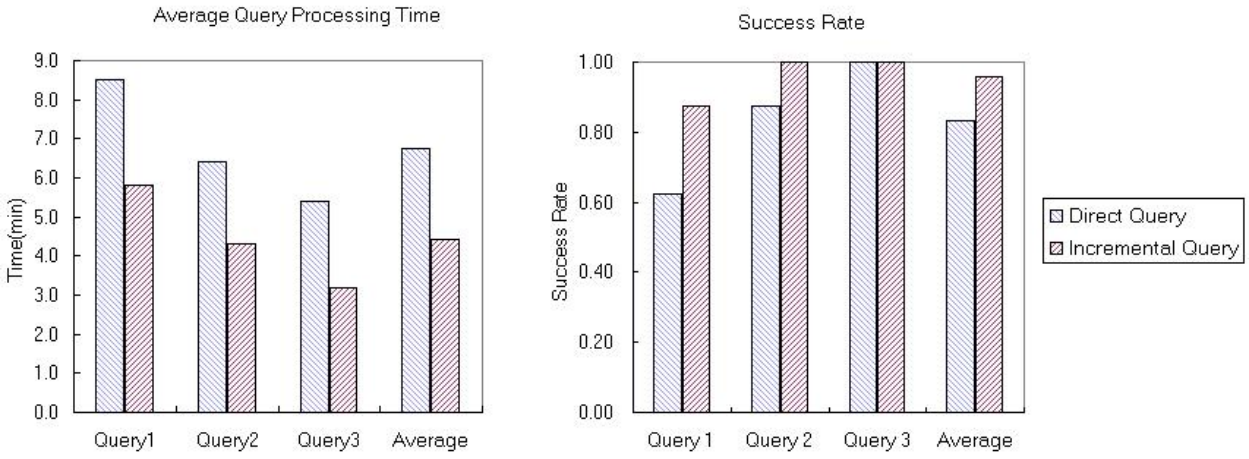

Figure 24: Result Analysis of Experiment 3 (Processing Time and Success Rate)

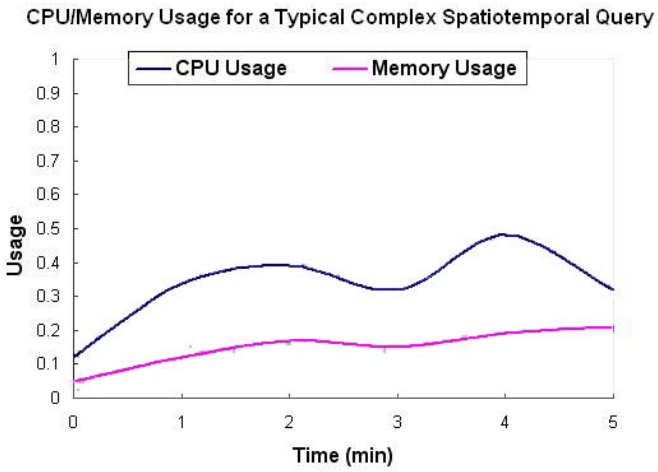

Figure 25: Result Analysis of Experiment 3 (CPU and Memory Usage)

system failure rate in the incremental query have improved 35.3% and 15.6% respectively.

Figure 25 shows the system CPU/Memory usage of the incremental query method on the same query as the one in figure 19(a). It is evident that the system resources are utilized more efficiently by the incremental query. The rationale is that using incremental query

system can execute previous query iterations at the same time when users input the current one. In other words, query execution and query building can be performed in parallel. More important, the maximum needs for CPU and memory dramatically decreased comparing to the direct query. This could have significant value when the fusion system is used on the portable devices which have limited system resources.

4.7 SUMMARY

In this chapter, a new method of incremental query is proposed. It is motivated by the query building bottleneck found through result analysis of the experiment described in chapter 3. Using the incremental query, a query is broken down into several simple and uniform elementary queries connected by query operators to resolve the accumulated complexity. The incremental query language, interface, and processing techniques are described in detail, which have been verified by two experiments, i.e., user query manner survey (Experiment 2) and query processing experiments on the system V2.0 (Experiment 3). The results of both experiments consistently indicate that the incremental query has evident advantages over the direct query in fusion systems.

5.0 QUERY PATTERN

Query Pattern is important, not only because it can capture and reuse the repeated structures in complex queries, and hence make them easy to build and process, but also it can help our approach make a significant progress towards a broadly applicable solution for query processing in general fusion systems.

I begin this chapter by describing several repeated structures in user queries which have been spotted in practice (section 5.1). It is conceivable that it could significantly improve system performance and lower user mental load if they can be retrieved and reused by any means. Based on these observations, the formal definition of query pattern in our system is described (section 5.2). Then some important query patterns are presented using the proposed schema – the observed typical tasks in general fusion systems are studied, and formulated as the query patterns (section 5.3). Next, in order to apply the query pattern into our incremental query processing, I design a post query reasoner equipped with the capability of pattern matching and learning. Several important techniques for pattern matching and learning are described in detail (section 5.4). Finally, the experiments have been done to demonstrate that the proposed techniques on query pattern can effectively improve the system usability and lower user mental load in the process of query handling (section 5.5).

5.1 MOTIVATION

Due to the diversity of the fusion applications, there is a wide range of information needs in fusion systems. However, in practice we observe that many information requests from users

come up with very similar structures. What follows are some examples of queries which have similar structures as the benchmark queries described in Appendix A.

Example 5.1.1 *The benchmark query–"Find a red car followed by a truck within 10 minutes" has the following similar queries:*

- *Find a bus followed by a car within 5 minutes.*
- *Search for a car which follows a red truck in 1 minute.*
- *Search for a red truck entered the parking lot within 3 minutes before the gate closed.*
- ...

Despite of some minor variations, the commonality among these queries is: they all have two object variables X and Y and a time variable T, and the corresponding relation is that X follows Y within time T.

Example 5.1.2 *The benchmark query–" Find a black car in the area 1 which entered the parking lot through Gate A during 6:00pm – 9:00pm." has the following similar queries:*

- *Find a truck currently in area B which was spotted in area A at 7:00am today.*
- *Search for a moving bus which parked in area A during the night.*
- *Search for a car in area A which previously parked in area B for three consecutive days.*
- ...

Despite of some minor variations, the commonality among these queries is: they all have two object variables X and Y, and the corresponding relation is that X is associated with Y, i.e., X and Y are same object in different observations.

Example 5.1.3 *The benchmark query–" Did the results of some sensors conflict with each other (i.e. different types referring to same object) while finding a truck passed Gate B at 7:00pm? " has the following similar queries:*

- *Did the results of some sensors conflict with each other while finding a red car in the area B?*
- *Did the results of some sensors conflict with each other while finding a truck currently in area B which was spotted in area A at 7:00am today?*

- *Check for the conflicts among the data sources while searching for a red car followed by a truck within 10 minutes.*
- ...

Despite of some minor variations, the commonality among these queries is: they all have a object variable X, and the corresponding query is whether or not there are conflictions among different data sources while retrieving the information about object X.

Because in incremental query schema a complex query is broken down into a chain elementary queries connected by query operators, the repeated structures become much more obvious to detect and reuse. It is conceivable that it could significantly improve the system performance and lower the user mental load if these repeated structures can be learned and reused by any means. Query pattern, which is proposed to capture and reuse these repeated structures, is discussed in detail in the following sections.

5.2 DEFINITION OF QUERY PATTERN

On the basis of the incremental query, query patterns can be easily generalized from a set of similar query paths in which some constants are marked as variables with appropriate scope. Through this way, a single pattern can cover a group of similar queries by instantiating the variables according to the scope and context. Formally, a query pattern is defined as follows.

Definition 5.2.1 (Query Pattern) *A **query pattern** QPN is defined as a triple: QPN:=* $< QP, S, L >$ *where*

- $QP = (Q_0, Q_1, \ldots, Q_n)$ *is a query path;*
- $S = \{var_1 : type_1, \ldots, var_m : type_m\}$ *is a set of variables with the corresponding types,* $var_1, \ldots, var_m$ *are variables in the query* $Q_i (i = 0, 1, \ldots, n)$*,* $type_1, \ldots, type_m$ *are ontology concepts in OKB, where* $type_i$ *describes the scope of variable* $var_i (i = 0, 1, \ldots, m)$*.*
- *L is a linguistic label.*

The detail about the ontology concepts in OKB is discussed in section 3.2.

Definition 5.2.2 (SubPattern) *A **SubPattern** is a binary predicate on query patterns* $QPN_1 = < QP_1, S_1, L_1 >$ *and* $QPN_2 = < QP_2, S_2, L_2 >$,

$$SubPattern(QPN_1, QPN_2) \Longleftrightarrow (S_1 \subset S_2)\ AND\ SubPath(QP_1, QP_2)$$

The following rules hold with SubPattern relations:

SubPattern(X, Y) AND SubPattern(Y, Z) $\Longrightarrow$ SubPattern(X, Z) (transitivity)

SubPattern(X, Y) AND SubPattern(Y, X) $\Longrightarrow$ X $\equiv$ Y (anti-symmetry)

What follows are the examples of query patterns generalized form the benchmark queries described in Appendix A.

Example 5.2.1 *From the benchmark query–"Find a red car followed by a truck within 10 minutes", the following query pattern can be generated :*

Query Patten $QPN = (QP, S, L)$;

- QP *is a query path,* $QP = (Q_1, Q_2)$; $Q_2 =< q_1, \tau_1, Q_1 >$;
  in which,
  Q_1 = <*type1="truck", LR/IR/CCD, AOI, TOI, object.moving = TRUE* >

  $\tau_1(object_j, object_k) = (object_j.t > object_k.t)\ AND$
  $(object_j.time < object_k.time + TimeInterval)$

  q_1 = <*type2="car", LR/IR/CCD, AOI, TOI, (object.color=color1,i.e. "red") AND (object.moving = TRUE)*>
- S *is a set of variables,* $S = \{$ *type1 : Recognizable Object, type2 : Recognizable Object, TimeInterval : Constant, color1 : Color* $\}$;
- L *is a linguistic label,* L = *"Object of type1 follows object of type2 within time t";*

Example 5.2.2 *From the benchmark query–"Find a black car in the area 1 which entered the parking lot through Gate A during 6:00pm – 9:00pm.", the following query pattern can be generated:*

Query Patten $QPN = (QP, S, L)$;

- QP *is a query path,* $QP = (Q_1, Q_2)$; $Q_2 =< q_1, \tau_1, Q_1 >$;
  in which,

Q_1 = <*type1="car", LR/IR/CCD, location1=Area 1, time1=NOW, object.color=color1, i.e., black)*>

$\tau_1(object_j, object_k) = associate(object_j, object_k)$

q_1 = <*type1, LR/IR/CCD, location2 = Gate A, time2 = 6:00pm – 9:00pm, TRUE*>

- *S is a set of variables, S = {type1: Recognizable Object, location : Recognizable Object, location2 : Recognizable Object , time1 : TimeConstant, time2 : TimeConstant, color1 : Color};*
- *L is a linguistic label, L = "Has Object in type1 been spotted before? ";*

Example 5.2.3 *From the benchmark query–"Did the results of some sensors conflict with each other (i.e. different types referring to the same object) while finding a truck passed Gate B at 7:00pm?", the following query pattern can be generated:*

Query Patten $QPN = (QP, S, L)$;

- *QP is a query path,* $QP = (Q_1, Q_2)$; $Q_2 =< q_1, \tau_1, Q_1 >$;
  in which,
  Q_1 = <*type1="truck", LR/IR/CCD, location1= Gate B, time1 = 7:00pm, TRUE*>

$$\tau_1(object_k, object_j) = object_k.time = object_j.time$$
$$AND\ object_k.position = object_j.position$$
$$AND\ object_k.type \neq object_j.type$$

q_1 = <*type1, DependencyGraph, location1, time1, TRUE*>

- *S is a set of variables, S = {type1 : Recognizable Object, location1 : Recognizable Object , time1: TimeConstant};*
- *L is a linguistic label, L = "Check the conflictions among data sources";*

5.3 OBSERVED QUERY PATTERNS

In this section, several important query patterns are described, which can cover most frequently used queries observed in previous research. First I study the typical tasks in fusion

systems (section 5.3.1) – these tasks are proposed by S.K. Chang and E. Jungert in [CJ04], which summarize the common information needs in various fusion systems. Next, I formulate them into the *incremental queries*, and finally abstract them into the *query patterns* (section 5.3.2). In fact, these typical tasks are studied here for the following two major purposes:

1. They provide a complete and accurate description about typical information needs in general fusion systems, which will help readers have a better understanding on the information retrieval problems discussed in this dissertation.
2. They are some excellent examples to demonstrate expressiveness of the proposed query method, because all these tasks can be easily formulated into the incremental queries and query patterns.

5.3.1 Typical Tasks in Information Fusion System

In what follows we describe the typical tasks in information fusion system, which can be grouped into three types. The first type of tasks, is concerned with how to improve the result of the original query by considering other aspects of the data sources. The second type is concerned with how to associate different object instances with each other, and the third group is concerned with queries that need to inspect the dependency tree from the last user query.

- Type I Tasks

  These tasks require the generation of new and elaborate queries that basically depends on certain conditions among the spatial objects normally found in sensor data. This type of tasks requires retrieving similar objects to get more accurate and complete result, for example:

  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; New elaborate queries are created 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?

   New elaborate queries are created 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 new elaborate queries are created from the templates. As a consequence the query processor responds to the question whether it is possible that the primary object can be found in an area with the actual geographic background.

4. Has the object type been previously observed in this background context?

   This query is similar to 3 but requires involvement of earlier output instances, i.e. the IOI must be involved as well.

5. 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.

6. 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.

- 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.

7. Can the retrieved object be associated to an earlier single observation?

   This query type requires the solution of the association problem.

8. 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.

- 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.

  9. Is it possible that the quality of the background information may have influenced the query result?

     For instance, the question could be whether there are any missing data in the AOI. This could be determined from a particular query. For a lot of missing data the risks of not finding a particular object increases.

  10. Did the result of some of the sensors (data sources) contradict each other?

      Does not require any new query; only an inspection of the dependency tree.

  11. 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.

  12. Which sensors (data sources) were used to answer the query?

      Does not require any new 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 On the contrary it could have contributed to another outcome of the query.

  For more detail about these typical tasks, please refer to [CJ04, CJL06].

5.3.2 Query Pattern for the Typical Tasks

In this section, every typical task described in section 5.3.1 is expressed by a (nested) incremental queries where the elementary queries are marked with their processing order[1]. Then the corresponding query pattern is formulated using the definition in section 5.2.

[1]For the details about the query language, please refer to [CJC04].

Query 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?*

Incremental Query

(2) SELECT $object_k.type, object_k.position, object_j.type, object_j.position$

(2) CLUSTER * ALIAS $object_k$

(2) FROM $PerceptionSource$

(2) WHERE $Inside(AOI, object_k)$

(2) AND $object_j.t = object_k.t$

(2) AND $distance(object_k, object_j) < \delta$

(2) AND $similar(object_k, object_j)$

(2) AND $object_j$ in

(1) SELECT $object_i.type, object_i.position$

(1) CLUSTER * ALIAS $object_i$

(1) FROM PerceptionSource

(1) WHERE $Inside(AOI, object_i)$

(1) AND $object_i.t = t_{given}$

(1) AND $object_i.type = type1$

Query Path $QP_1 = (Q_1, Q_2)$; $Q_2 =< q_1, \tau_1, Q_1 >$;

in which,

$Q_1 =$ <type1, PerceptionSource, AOI, t_{given}, TRUE >

$\tau_1(object_j, object_k) = object_j.t = object_k.t$
and $distance(object_k, object_j) < \delta$
and $similar(object_k, object_j)$

$q_1 =$ <OBJECT, PerceptionSource, AOI, t_{given}, TRUE >

Query Patten $QPN_1 = (QP_1, S, L)$;

in which,

$S =$ { type1 : Recognizable Object, δ : Constant}

L = "Are there any other objects in the proximity of the retrieved object that are of similar or of equal type as the retrieved object?"

In the above *similar* function, the similarity is determined by means of the ontology. That is, an object similar to a given object can be defined either as the parent or a sibling of the given object considering the structure of the object part of the ontology. The distance function is simply defined as the Euclidean distance between the objects.

Query 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?*

Incremental Query

(2) SELECT $object_k.type, object_k.position, object_k.t, object_j.type, object_j.position$
(2) CLUSTER * ALIAS $object_k$
(2) FROM $PerceptionSource$
(2) WHERE $Inside(AOI, object_k)$
(2) AND $t_{start} < object_i.t < object_k.t_{given}$
(2) AND $distance(object_k.position, object_j.position) < \delta$
(2) AND $similar(object_k.type, object_j.type)$
(2) AND $object_j$ in
(1) SELECT $objedct_i.type, object_i.position$
(1) CLUSTER * ALIAS $object_i$
(1) FROM $PerceptionSource$
(1) WHERE $Inside(AOI, object_i)$
(1) AND $object_i.t = t_{given}$
(1) AND $object_i.type = type1$

Query Path $QP_2 = (Q_1, Q_2)$; $Q_2 =< q_1, \tau_1, Q_1 >$;
in which,

Q_1 =<type1, PerceptionSource, AOI, t_{given}, TRUE >

$\tau_1(object_j, object_k) = t_{start} < object_i.t < object_k.t$
and $distance(object_k, object_j) < \delta$
and $similar(object_k, object_j)$

q_1 = <OBJECT, PerceptionSource, AOI, t_{given}, TRUE >

Query Patten $QPN_2 = (QP_2, S, L)$;
in which,

S = { type1 : Recognizable Object, δ : Constant}

L = "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?"

Query 3: *Is the present background (context) information of proper type relative to the type of the retrieved object?*

Incremental Query

(2) SELECT $object_j.type, object_j.position, object_p.type$
(2) CLUSTER * ALIAS $object_p$
(2) FROM $ContextSource$
(2) WHERE $Inside(object_p, object_j)$
(2) AND $properbackground(object_p.type, object_j.type)$
(2) AND $object_j$ in
(1) SELECT $objedct_i.type, object_i.position$
(1) CLUSTER * ALIAS $object_i$
(1) FROM $PerceptionSource$
(1) WHERE $Inside(AOI, object_i)$
(1) AND $object_i.t = t_{given}$
(1) AND $object_i.type = type1$

Query Path $QP_3 = (Q_1, Q_2)$; $Q_2 =< q_1, \tau_1, Q_1 >$;
in which,

Q_1 = <type1, PerceptionSource, AOI, t_{given}, TRUE >

$\tau_1(object_p, object_j) = Inside(object_p, object_j)$
and $properbackground(object_p.type, object_j.type)$

q_1 = <OBJECT, ContextSource, AOI, t_{given}, TRUE >

Query Patten $QPN_3 = (QP_3, S, L)$;
in which,

S = {type1 : Recognizable Object }

L = "Is the present background (context) information of proper type relative to the type of the retrieved object?"

Query 4: *Has this object type been previously observed in this background context?*

Incremental Query

(4) SELECT $object_p.type, object_i.type, object_i.position, object_i.t$
(4) CLUSTER * ALIAS $object_p$
(4) FROM $ContextSource$
(4) WHERE $object_j.type = object_p.type$
(4) AND $object_i.type = object_k.type$
(4) AND $t_{start} < object_i.t < object_k.t$
(4) AND $object_i$ in
(3) SELECT $object_m.type, object_m.t, object_m.position$
(3) CLUSTER * ALIAS $object_m$
(3) FROM $PerceptionSource$
(3) WHERE $Inside(AOI, object_m)$
(3) AND $t_{start} < object_m.t < t_{given}$
(3) AND $object_m.type = type1$
(3) AND $object_j$ in
(2) SELECT $object_l.type$
(2) CLUSTER * ALIAS $object_l$

(2) FROM $ContextSource$

(2) WHERE $Overlap(object_l, object_k)$

(2) AND $object_k$ in

(1) SELECT $objedct_n.type, object_n.position$

(1) CLUSTER * ALIAS $object_n$

(1) FROM $PerceptionSource$

(1) WHERE $Overlap(AOI, object_n)$

(1) and $object_n.t = t_{given}$

(1) and $object_n.type = type1$

Query Path $QP_3 = (Q_1, Q_2, Q_3, Q_4)$;

in which,

$Q_2 =< q_1, \tau_1, Q_1 >$;

$Q_3 =< q_2, \tau_2, Q_2 >$;

$Q_4 =< q_3, \tau_3, Q_3 >$;

$Q_0 =$ <type1, PerceptionSource, AOI, t_{given}, TRUE >

$\tau_1(object_l, object_k) = Overlap(object_l, object_k)$

$q_1 =$ <OBJECT alias type2, ContextSource, AOI, t_{given}, TRUE >

$\tau_2(object_l, object_k) = t_{start} < object_m.t < object_k.t$

$Q_2 =$ <type1, PerceptionSource, AOI, t_{given}, TRUE >

$\tau_3(object_i, object_j) = Overlap(object_l, object_k)$

$Q_3 =$ <type2, ContextSource, AOI, t_{given}, TRUE >

Query Patten $QPN_4 = (QP_4, S, L)$;

in which,

$S =$ {type1 : Recognizable Object }

$L =$ "Has this object type been previously observed in this background context?"

Query 5: *Is the retrieved object partly hidden by some other object that is part of the context?*

Incremental Query

(2) SELECT $object_p.type, object_p.position, object_j.type, object_j.position$

(2) CLUSTER * ALIAS $object_p$

(2) FROM $PerseptionSource$

(2) WHERE $Partlyoverlap(AOI, object_p)$

(2) AND (($object_p.type =$ "$terrain - feature$")

(2) OR ($object_p.type =$ "$building$")

(2) OR ($object_p.type =$ "$natural - object$"))

(2) AND (($Partlyoverlap(object_p, object_j)$)

(2) OR ($Beside(object_p, object_j)$)))

(2) AND $object_p.t = object_j.t$

(2) AND $object_j$ in

(1) SELECT $type$

(1) CLUSTER * ALIAS $object_i$

(1) FROM $PerceptionSource$

(1) WHERE $Inside(AOI, object_i)$

(1) AND $object_i.t = t_{given}$

(1) AND $object_i.type = type1$

Query Path $QP_5 = (Q_1, Q_2)$; $Q_2 =< q_1, \tau_1, Q_1 >$;

in which,

$Q_1 =$ <type1, PerceptionSource, AOI, t_{given}, TRUE >

$\tau_1(object_p, object_j) = (($ $object_p.type =$ "$terrain - feature$")
or ($object_p.type =$ "$building$")
or ($object_p.type =$ "$natural - object$"))
and (($Partlyoverlap(object_p, object_j)$)
or ($Beside(object_p, object_j)$)))

$q_1 =$ <OBJECT, PerceptionSource, AOI, t_{given}, TRUE >

Query Patten $QPN_5 = (QP_5, S, L)$;

in which,

S = {type1 : Recognizable Object }

L = "Is the retrieved object partly hidden by some other object that is part of the context?"

Query 6: *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?*

Incremental Query

(2) SELECT $object_k.type, object_k.position, object_j.type, object_j.position$

(2) CLUSTER * ALIAS $object_k$

(2) FROM $PerceptionSource$

(2) WHERE $Inside(AOI, object_k)$

(2) AND $distance(object_k.position, object_j.position) < \delta$

(2) AND $impact(object_k, object_j)$

(2) AND $object_k.t = object_j.t$

(2) AND $object_j$ in

(1) SELECT $type$

(1) CLUSTER * ALIAS $object_i$

(1) FROM $PerceptionSource$

(1) WHERE $Inside(AOI, object_i)$

(1) AND $object_i.type = type1$

(1) AND $object_i.t = t_{given}$

Query Path $QP_6 = (Q_1, Q_2)$; $Q_2 =< q_1, \tau_1, Q_1 >$;

in which,

Q_1 = <type1, PerceptionSource, AOI, t_{given}, TRUE >

$\tau_1(object_k, object_j) = distance(object_k.position, object_j.position) < \delta$
and $impact(object_k, object_j)$

and $object_k.t = object_j.t$

q_1 = <OBJECT, PerceptionSource, AOI, t_{given}, TRUE >

Query Patten $QPN_6 = (QP_6, S, L)$;
in which,

S = {type1 : Recognizable Object, δ : Constant}

L = "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?"

An example of this query in military application is a bus ($object_j$) with refugees close to a tank ($object_k$). The possible impact could be that it is prohibited to fire at a tank that is close to a bus.

Query 7: *Can the retrieved object be associated to an earlier single observation?*
Incremental Query

(2) SELECT $object_k.type, object_k.t, object_k.position, object_j.type,$
(2) $object_j.t, object_j.position$
(2) CLUSTER * ALIAS $object_k$
(2) FROM $PerceptionSource$
(2) WHERE $Inside(AOI, object_k)$
(2) AND $distance(object_k.position, object_j.position) < \delta$
(2) AND $t_{start} < object_k.t < object_j.t$
(2) AND $associate(object_k, object_j)$
(2) AND $object_j$ in
(1) SELECT $object_i.type, object_i.position$
(1) CLUSTER * ALIAS $object_i$
(1) FROM $PerceptionSource$
(1) WHERE $Inside(AOI, object_i)$
(1) AND $object_i.t = t_{given}$

(1) AND $object_i.type = type1$

Query Path $QP_7 = (Q_1, Q_2)$; $Q_2 =< q_1, \tau_1, Q_1 >$;
in which,

$Q_1 =$ <type1, PerceptionSource, AOI, t_{given}, TRUE >

$\tau_1(object_k, object_j) = distance(object_k.position, object_j.position) < \delta$
and $t_{start} < object_k.t < object_j.t$
and $associate(object_k, object_j)$

$q_1 =$ <OBJECT, PerceptionSource, AOI, t_{given}, TRUE >

Query Patten $QPN_7 = (QP_7, S, L)$;
in which,

$S =$ {type1 : Recognizable Object, δ : Constant}

$L =$ "Can the retrieved object be associated to an earlier single observation?"

In this query it is assumed that the function associate associates two objects to each other, i.e. the two object observations are the same although observed at different positions and at different times.

Query 8:*Can the retrieved object be part of an existing object?*

Since it is a recursive variation of query 7, so it will not be further elaborated.

Query 9: *Is it possible that the quality of the background information may have influenced the query result?*

Incremental Query

(2) SELECT $object_k.sensor, object_k.background, object_j.type, object_j.position$

(2) CLUSTER * ALIAS $object_k$

(2) FROM $DependencyTree$

(2) WHERE $missingdata(object_k.background)$

(2) AND $object_j$ in

(1) SELECT $object_i.type, object_i.position$

(1) CLUSTER * ALIAS $object_i$

(1) FROM $PerceptionSource$

(1) WHERE $Inside(AOI, object_i)$

(1) AND $object_i.t = t_{given}$

(1) AND $object_i.type = type1$

Query Path $QP_9 = (Q_1, Q_2)$; $Q_2 =< q_1, \tau_1, Q_1 >$;
in which,

Q_1 = <type1, PerceptionSource, AOI, t_{given}, TRUE >

$\tau_1(object_k, object_j) = missingdata(object_k.background)$

q_1 = <OBJECT, DependencyTree, AOI, t_{given}, TRUE >

Query Patten $QPN_9 = (QP_9, S, L)$;
in which,

S = {type1 : Recognizable Object }

L = "Is it possible that the quality of the background information may have influenced the query result?"

The measure of data quality here is based on the level of missing data. However, other definitions of quality can be thought of.

Query 10: *Did the final result contradict the result from any of the sensors (data sources)?*
Incremental Query

(2) SELECT $object_k.type, object_k.sensor, object_j.type, object_j.position$

(2) CLUSTER * ALIAS $object_k$

(2) FROM $DependencyTree$

(2) WHERE $object_k.position = object_j.position$

(2) AND $object_k.type \neq object_j.type$

(2) AND $object_k.t = object_j.t$

(2) AND $object_j$ in

(1) SELECT $object_i.type, object_i.position$

(1) CLUSTER * ALIAS $object_i$

(1) FROM $PerceptionSource$

(1) WHERE $Inside(AOI, objecti)$

(1) AND $object_i.t = t_{given}$

(1) AND $object_i.type = type1$

Query Path $QP_{10} = (Q_1, Q_2)$; $Q_2 =< q_1, \tau_1, Q_1 >$;

in which,

Q_1 = <type1, PerceptionSource, AOI, t_{given}, TRUE >

$\tau_1(object_k, object_j) = object_k.position = object_j.position$
and $object_k.type \neq object_j.type$
and $object_k.t = object_j.t$

q_1 = <OBJECT, DependencyTree, AOI, t_{given}, TRUE >

Query Patten $QPN_{10} = (QP_{10}, S, L)$;

in which,

S = {type1 : Recognizable Object }

L = "Did the final result contradict the result from any of the sensors (data sources)?"

Query 11: *Did any sensor (data source) contribute to the result in any extreme way?*

Incremental Query

(2) SELECT $object_k.type, object_k.sensor$

(2) CLUSTER * ALIAS $object_k$

(2) FROM $DependencyTree$

(2) WHERE ($OR(qualitative\_difference(object_k.belief\_value, object_j.belief\_value) = `large'$

(2) AND $object_k.position = object_j.position$

(2) AND $object_k.t = object_j.t$),

(2) ($object_k.t \neq object_j.t$

(2) AND $object_k.position = object_j.position$

(2) AND $object_k.type \neq object_j.type$))

(2) AND $object_j$ in

(1) SELECT $object_i.type, object_i.position$

(1) CLUSTER * ALIAS $object_i$

(1) FROM $PerceptionSource$

(1) WHERE $Inside(AOI, object_i)$

(1) AND $object_i.t = t_{given}$

(1) AND $object_i.type = type1$

Query Path $QP_{11} = (Q_1, Q_2)$; $Q_2 =< q_1, \tau_1, Q_1 >$;

in which,

Q_1 = <type1, PerceptionSource, AOI, t_{given}, TRUE >

τ(objectj, objectk) =

(OR (qualitative-difference($object_k$.belief-value,

$object_j$.belief-value) = "large"

AND $object_k$.position = $object_j$.position

AND $object_k.t = object_j.t$),

($object_k.t = object_j.t$

AND $object_k$.position = $object_j$.position

AND $object_k$.type $\neq$ $object_j$.type))

q_1 = <OBJECT, DependencyTree, AOI, t_{given}, TRUE >

Query Patten $QPN_{11} = (QP_{11}, S, L)$;

in which,

S = {type1 : Recognizable Object }

L = "Did any sensor (data source) contribute to the result in any extreme way?"

Query 12: *Which sensors (data sources) where used to answer the query?*

Incremental Query

(2) SELECT $object_k.sensor$

(2) CLUSTER * ALIAS $object_k$

(2) FROM $DependencyTree$

(2) WHERE $object_k.position = object_j.position$

(2) AND $object_k.type = object_j.type$

(2) AND $object_k.t = object_j.t$

(2) AND $object_j$ in

(1) SELECT $type$

(1) CLUSTER * alias $object_i$

(1) FROM $PerceptionSource$

(1) WHERE $Inside(AOI, object_i)$

(1) AND $object_i.t = t_{given}$

(1) AND $object_i.type = type1$

Query Path $QP_{12} = (Q_1, Q_2)$; $Q_2 =< q_1, \tau_1, Q_1 >$;

in which,

Q_1 = <type1, PerceptionSource, AOI, t_{given}, TRUE >

$\tau_1(object_k, object_j) = object_k.position = object_j.position$
and $object_k.type = object_j.type$
and $object_k.t = object_j.t$

q_1 = <OBJECT.sensor, DependencyTree, AOI, t_{given}, TRUE >

Query Patten $QPN_{12} = (QP_{12}, S, L)$;

in which,

S = {type1 : Recognizable Object }

L = "Which sensors (data sources) where used to answer the query?"

5.4 APPLICATION OF QUERY PATTERN

In order to apply the propose query pattern schema into increment query processing, a post query reasoner is introduced to perform the pattern matching and learning. With the post query reasoner, the pattern can be matched and reused at anytime based upon the query that users currently construct; at the same time, new patterns can be retrieved from user's query history.

This section begins with an overview of the system with the post query reasoner (section 5.4.1). Next, the pattern matching techniques is described which can apply the observed patterns to the query processing and make it simple (section 5.4.2). Then the manually query pattern retrieval (section 5.4.3) and automatically query pattern learning (section 5.4.4) are discussed in detail respectively. Finally, the interoperability between query pattern and ontology knowledge is discussed (section 5.4.5), which provides a novel way to construct ontology from query patterns.

5.4.1 System Overview

The fusion system with the post query reasoner is illustrated in figure 26. At the first level, the system consists of five fundamental components: *incremental query system*, *post query reasoner*, *pattern database*, *query history*, and *ontology knowledge base*. The query history includes all the incremental queries that users have issued. The pattern database stores the query patterns that are either pre-defined or learned from the query history by the post query reasoner. The ontology knowledge base contains the knowledge related to the data sources and retrieved objects, which is described in section 3.2.

The incremental query system has been described in detail in chapter 4. It includes the query interface, source data analysis, and incremental query processor. The query interface

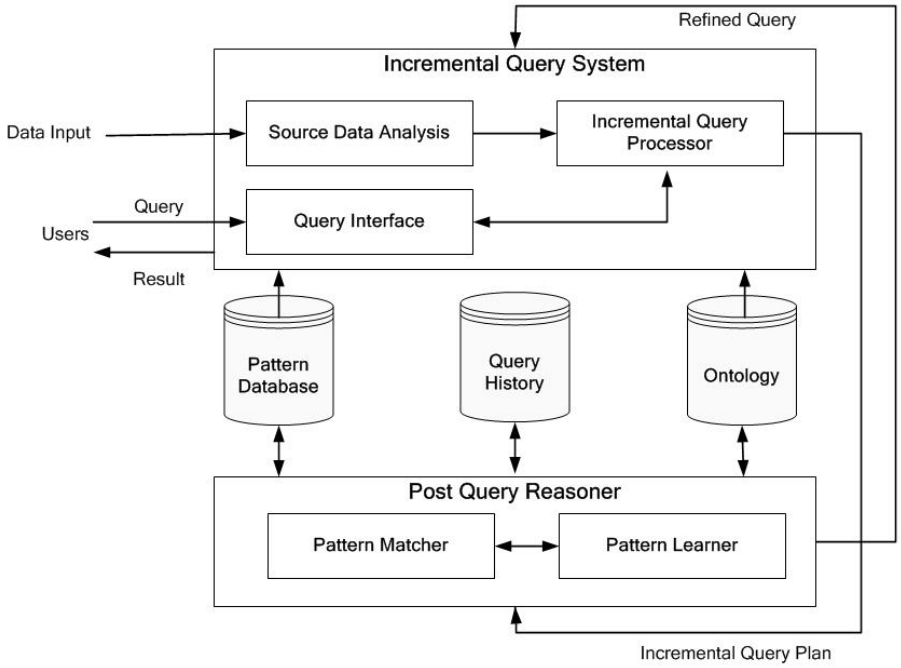

Figure 26: The System Diagram with Post Query Reasoner

interacts with users to retrieve user requests and present the results. The source data analysis performs the preliminary recognition and retrieval operations on the input data. The query execution plan is generated in the query processor which collects user request from the query interface and input data from the source data analysis. The query processor performs the retrieval and fusion operations according to the execution plan. Finally, it returns the query results to users through the query interface.

Initially there are no query patterns in the pattern database. The query patterns are retrieved dynamically by the post query reasoner that performs the rule based reasoning on the query plan feed by the query processor. The post query reasoner consists of two major components: pattern matcher and pattern learner. Because a query is usually processed in serveral iterations, the preliminary query plan and results are sent to the post query reasoner each time an iteration is done. The pattern matcher matches the current query with the query patterns stored in the pattern database, and returns the refined query plans

according to the matched patterns. It is up to users to decide whether to apply the matched query pattern and which one to apply if multiple patterns are applicable. Meanwhile the queries accumulated in the system are the valuable assets to learn new query patterns. The system can either ask users to manually formulate query patterns based on the query paths or automatically learn query patterns from the query history. The query pattern matching and learning algorithms are described in the following sections.

As discussed in previous sections, ontology knowledge base plays an important role in query pattern learning and matching. New query patterns can be generated from the previous patterns simply by adding or modifying ontology concepts. On the other hand, query pattern provides very valuable information describing ontology. It is also possible to construct new ontology from a group of query patterns. The interoperability with ontology is one of the main functions of the post query reasoner.

5.4.2 Pattern Matching

The method to match query pattern is proposed on the basis of *unification*. In fact, unification is an important process widely used in Prolog, resolution, term rewriting, and natural language understanding. In our method, unification is employed to determine whether a query path can match a query pattern.

Definition 5.4.1 *Two queries Q_1 and Q_2 are unifiable on S – a set of variables in Q_2 if only if a substitution on the variables in S can be found to make Q_2 identical to Q_1, that is:*

$$Unifiable(Q_1, Q_2, S) \Longleftrightarrow \exists \text{ substitution } \alpha \text{ on } S,\ Q_1 \equiv Q_2 \text{ with } \alpha$$

What follows is an example of unifiable queries:

Example 5.4.1 *There are two elementary queries Q_1 and Q_2, and S – a set of variables in Q_2:*

- $Q_1 =$ < *"car", LR/IR/CCD, Area 1, NOW, object.moving = TRUE AND object.color = black* >

- Q_2 = <*type1 = "truck", LR/IR/CCD, location1=Gate A, time1 = 7:00 – 9:00pm, object.moving = TRUE AND object.color = color1, i.e. red)*>

- S = { *type1 : Vehicle, location1 : Recognizable Object, time1 : Constant, color1: Color* }

Q_1 *and* Q_2 *are unifiable because there is a substitution* α *on* S,

$$\alpha = \{\ type1/\text{“car”},\ location1/\ \text{“Area 1”},\ time1/\text{“NOW”},\ color1/\text{“black”}\ \}$$

which makes:

$$Q_1 \equiv Q_2 \ with\ \alpha$$

It is worth noting that the substitution in unification process must be performed between the variables with compatible types. For instance, in the example describe above, if the variable "location1" is marked with type "Gate" instead of "Recognizable Object", Q_2 will not be unifiable with Q_1 because the "location1" can not be substituted by "Area 1".

On the basis of the unifiable queries, the *unifiable query paths* are defined as follows:

Definition 5.4.2 *Two query paths* $QP_1 = \{Q_1, Q_2, \cdots, Q_n\}$ *and* $QP_2 = \{Q'_1, Q'_2, \cdots, Q'_n\}$ *are unifiable on* S *– a set of variables in* QP_2 *if only if a substitution on the variables in* S *can be found to make* Q_i *unifiable with* Q'_i $(i = 1, 2, \cdots, n)$*, that is :*

$$Unifiable(QP_1, QP_2, S) \Longleftrightarrow \exists\ substitution\ \alpha\ on\ S, \forall\ i = 1, 2, \cdots, n, Q_i \equiv Q'_i\ with\ \alpha$$

Literally, two query paths are unifiable if and only if every pair of corresponding queries in the both paths can be unified with a single uniform substitution. Based on the unification of query paths, the definition of emphpattern matching is described as follows:

Definition 5.4.3 *A query pattern* $QPN =< QP, S, L >$ *can match a query path* QP_1 *if only if* QP_1 *is unifiable with any subpath of* QP *on* S*, that is:*

$$Match(QPN, QP_1) \Longleftrightarrow \exists QP_2\ SubPath(QP_2, QP)\ AND\ Unifiable(QP_1, QP_2, S)$$

If QP_1 is the current query path and Match(QP_1, QPN) is true, QP_1 could be automatically extended to QP by the query matcher on the basis of unification. For more details

about the techniques of unification, please refer to [Vad88, Kni89]. If multiple query patterns are matched with the current query, all candidates will be listed by the linguistic labels for users to choose. In this way, users could build a complex query quickly by referring to previous experience recorded as a query pattern.

5.4.3 Manually Pattern Construction and Learning

Query patterns are important because they could cover most of the frequently used queries and make them easy to construct and process. In our method, a set of query patterns $S_{pattern}$ is stored in the pattern database and maintained during the query processing. Particularly, the patterns formulating the common tasks in section 5.3 are preinstalled in $S_{pattern}$.

```
    PROCEDURE addPattern(QPN: pattern; S_pattern: set of patterns)
    var QPN_1:pattern;
    begin
1:    for i:=0 to S_pattern.size()-1 do
2:    begin
        /*get the pattern in the set one by one. */
3:      QPN_1:= S_pattern.Item(i);
        /* If QPN is sub-pattern of QPN_1, no need to add QPN, quit. */
4:      if SubPattern(QPN, QPN_1) then return ;
        /* If QPN_1 is sub-pattern of QPN, delete QPN_1 from the set. */
5:      if SubPattern(QPN_1, QPN) then S_pattern.delete(QPN_1);
6:    end;
      /* If QPN is not the sub-pattern of any pattern in the set , add it. */
7:    S_pattern.add(QPN);
    end;
```

Figure 27: Add a Query Pattern QPN into $S_{pattern}$

Every time users finish a query, the complete query path is recorded, and then stored into

the query history. As we discussed above, the primary difference between a query path and a query pattern is that query pattern has a set of variables which can be substituted during pattern matching. In other words, a query pattern can represent a group of similar query paths by introducing a set of variables. Therefore, the *determination of the variable set* is the core problem to generate query patterns from query paths. The simplest solution is to let users decide the variables – a user interface (see section 5.5.1) is designed to ask users to input variables and their scope/type in the OKB for the query path every time they finish a query. As a result, query patterns can be manually defined by users. However, the patterns created by users frequently are repeated or redundant patterns comparing with the ones in pattern database. For this reason, I proposed a learning algorithm to filter the patterns input by users. As shown by the pseudo code in figure 27, when a query pattern QPN input by users is added into the pattern set $S_{pattern}$, we need to check whether or not it is a useful pattern. First QPN is compared with every pattern in $S_{pattern}$, which is implemented by the *for* loop in line 1– 6. If QPN is the sub-pattern of any patterns in $S_{pattern}$, it means that it is a repeated pattern so no need to continue. Meanwhile, if any pattern QPN_1 in $S_{pattern}$ is a sub-pattern of QPN, it means QPN is a better pattern candidate than QPN_1 – so QPN_1 is deleted from $S_{pattern}$. Finally, by the end of the loop QPN is ensured to be a new pattern, it is added to $S_{pattern}$ in line 7. Verified by experiments, the algorithm can effectively maintain a set of high quality patterns.

5.4.4 Automatically Pattern Learning

The method described in section 5.4.3 proves to be a effective method to generate new query patterns. Nevertheless, it still needs users to select the variables in query paths, which puts some extra work for them in addition to query building. It is conceivable that it will be much more convenient if the system can find a way to determine the variables without human guidelines, i.e., the query patterns can be learned automatically. However, for a feasible automatically pattern learning method, it has to solve the following two problems:

1. How to select the variables from query paths to generate new query patterns?
2. How to determine the scope/type of these variables?

The simplest solution for the first problem is to randomly mark the constants in query paths as variables and then add them into pattern database using the same way as we described in section 5.4.3. However, in practice the random selection of variables has been found extremely unpredictable and error-prone. The risk for choosing variables in query paths comes from the following two aspects:

1. If too few or inappropriate constants in a query path are marked as variables, the corresponding pattern could be too specific that it can only match very few query paths. In most cases, this kind of query patterns are useless. Furthermore, it is even worse that they could increase the workload of pattern matching and maintaining.
2. If too many or inappropriate constants are marked as variables, the corresponding pattern could be too general that it can cover a large number of query paths. In this case, users still have to specify many variables even if their queries got matched with this kind of patterns. In other words, it is not obvious to see any advantage of query patterns in this scenario.

Generally speaking, it could be extremely cumbersome to find the right variables in query paths in the pattern learning process. Recall the way how humans recognize query patterns, they identify the repeated structures on the basis of their experience. Following the same way, I proposed a heuristics function based on query history to choose variables from query paths. First, I add constraints on the number of variables in query patterns. A threshold θ is introduced for the number of variables. Only the patterns with less than θ variables are considered to be absorbed into the pattern database. The rationale behind this rule is that we are not interested in the patterns with too many variables because they are usually too general to be useful. Then under this restriction, the quality of a query pattern can be assessed by the number of the query paths it can cover in query history. The more query paths it covers, the more useful it could be in practice. For this reason, the *coverage* of a query path is defined as follows:

Definition 5.4.4 *Given a query pattern QPN and a query history, i.e., a set of query paths H_{path}, the coverage of QPN on H_{path} is the number of the query paths in H_{path} which can be matched with QPN.*

$$coverage(QPN, H_{path}) = \#\{\forall QP \in H_{path}, Match(QPN, QP)\}$$

With the help of the threshold θ and the heuristic function *coverage*, the system can find a appropriate set of variables in query paths, and then generate new query patterns based on them.

For the second problem, we are also facing a similar dilemma as the first one. If the variables are abstracted by some concepts with too narrow ranges in the OKB, the corresponding query patterns could be too specific. On the contrary, if they are abstracted by the ones with too wide ranges, the corresponding query patterns could be too general. Generally speaking, it is an extremely difficult problem to find the right scope/type of the variables in query patterns. In practice, we apply a simple rule to solve this problem: the scope of every variable in query patterns is set as the parent concept of the variable's type in the OKB. For example, given a variable "type1" with the type as "Car" in the OKB, when a query path containing "type1" is learned to generate a new query pattern, the scope of "type1" will be set as the parent concept of "Car" in the OKB –"Vehicle". The rational behind this rule is: it can effectively eliminate some variables with too narrow or too wide ranges in query patterns, although it may hurt some degree of flexibility in query pattern learning.

With these two problems resolved, an automatically pattern learning algorithm is proposed on the basis of the algorithm described in figure 27. When users finish a query, the corresponding query path is added into query history; meanwhile a query pattern is generated and added into the pattern database. The detail is shown by the pseudo code in figure 28: First, QP is directly added into H_{path} in line 1. Next, the subset of variables $S_{maxcoverage}$ which can reach the maximum coverage for QP on H_{path} is identified in line 2 – 3. Then a new query pattern QPN is constructed form the QP with the variable set $S_{maxcoverage}$ in line 4 – 6. Finally, the QPN is added into $S_{pattern}$ using the algorithm described in figure 27.

5.4.5 Dynamic Information Exchange with Ontology Knowledge

Ontology knowledge provides a crucial support for query pattern matching and learning. At the same time, query pattern is among the most valuable assets which describe the ontology knowledge. By adding/modifying the ontology, new query patterns can be generated. On

PROCEDURE addPath (QP: query path; H_{path}: query history;
$S_{pattern}$: set of patterns)

var QPN:pattern;

begin

/* Add the query path into query history. */

1: H_{path}.add (QP);

/* Get the set containing all the variables in QP. */

2: S_{all} = set of all variables in QP;

/* Find the subset with the maximum coverage on query history;
θ is the maximum number of varibles. */

3: $S_{maxcoverage}$ = max $coverage(S, H_{path})$ $\{S \mid S \subset S_{all}$ AND $|S| \leq \theta\}$

/* Create a a new query pattern QPN.*/

4: $QPN.QP = QP$;

5: $QPN.S = S_{maxcoverage}$;

/* Set the label of the pattern as the description of $S_{maxcoverage}$.*/

6: $QPN.L$ = description of $S_{maxcoverage}$;

/* Add the pattern QPN into $S_{pattern}$ by calling the algorithm in figure 27*/

7: addPattern(QPN, $S_{pattern}$);

end;

Figure 28: Add a Query Path QPN into H_{path}, and update $S_{pattern}$

the other hand, the new ontology knowledge can also be constructed from query patterns which are accumulated from user query experiences.

We believe that the dynamic exchange of information between ontology knowledge and query pattern has significant importance, because for a long time the construction of ontology knowledge remains a difficult problem [McG02]. Particularly, it is extremely cumbersome for end users to construct and maintain ontology knowledge by themselves. However, it is much

easier for them to specify query patterns based on the issued queries. It will be of great value if we can propose a methodology to automatically construct and update ontology knowledge through query pattern retrieving. Here I present some preliminary works in this direction by the following two examples. The further works along this line will be the subject of our future research.

Example 5.4.2 *Suppose several recognition and fusion algorithms are available to detect new object "Motorcycle". In order to generate query patterns about the new object, users can specify it in the OKB by adding new concept "Motorcycle" and the relation IS_A ("Motorcycle", "Vehicle"). As a result, all previous query patterns on Vehicle and its super concepts in the OKB will be applicable on the new object "Motorcycle".*

Alternatively, if users know nothing about the ontology, they still can construct the ontology knowledge by specifying query patterns on the new object. Given a query path containing the new object, users can manually mark "Motorcycle" as variable with the scope "Vehicle" during the query pattern retrieving process. Based on the user's inputs, the system can automatically add the new concept "Motorcycle" and the relation IS_A ("Motorcycle", "Vehicle") into the OKB.

Obviously the two methods in the above example have the same result, however, the second method is much more feasible in practice.

Example 5.4.3 *Suppose there are two concepts C_1 and C_2 in the OKB, but the relation between these two concepts remains unknown. Assuming that plenty of query patterns have been accumulated on these two concepts through user query experience, we can infer their relation from the related query patterns. For instance, we could have the following rule:*

$$(\forall QPN \in S_{pattern}\ QPN \text{ is applicable on } C_1 \Rightarrow QPN \text{ is applicable on } C_2) \Rightarrow IS\_A(C_1, C_2)$$

Literally, it means that if all query patterns on C_1 can also be applied on C_2, C_1 is a subtype of C_2.

More rules similar as the above one can be developed. Obviously, this kind of rules are not always correct, however, with the accumulation of query patterns, the rules based on them will be more and more accurate. Therefore, eventually we can construct a reasonable

ontology knowledge base through the process of query pattern retrieving. In fact, the query pattern represents experience, while the ontology represents knowledge. The rational is that it is always possible to retrieve knowledge from experience.

5.5 EXPERIMENTAL SYSTEM AND EXPERIMENTS

In order to verify the proposed methodology which applies query pattern into the incremental query processing, an experimental system named V3.0 has been designed and implemented on the basis of system V2.0 (see section 4.6.1). The system V3.0 includes a fully functioning post query reasoner that can match and learn query patterns. I begin this section by the introduction of the experimental system (section 5.5.1). Then the experiments on it are described in detail (section 5.5.2–5.5.3).

5.5.1 Experimental IVET System V3.0

Table 6: Experimental System V3.0

Item	**Detail**
Title	Experimental IVET System V3.0
Goal	An incremental query system with a post query reasoner that can match and learn query patterns, on the basis of system V2.0 (table 4 in section 4.6.1)
Challenges	Query pattern matching and learning
Designed by	Xin Li

The experimental IVET system V3.0 has been implemented followed by the system diagram shown in figure 26. Some important attributes of the system are listed in table 6.

The user input goes through the query interface to the query processor. After the query is executed, the query results go through a fusion stage, and the final results together with

the query are sent to the post query reasoner. The reasoner will match the query patterns in $S_{pattern}$ with the current query. The matched query patterns are sent to the query processor as well as the query result. At the end of the query, the query pattern annotated by user is sent back to the reasoner. New query patterns are synthesized by pattern learner and saved in the pattern database using the algorithm described in figure 27.

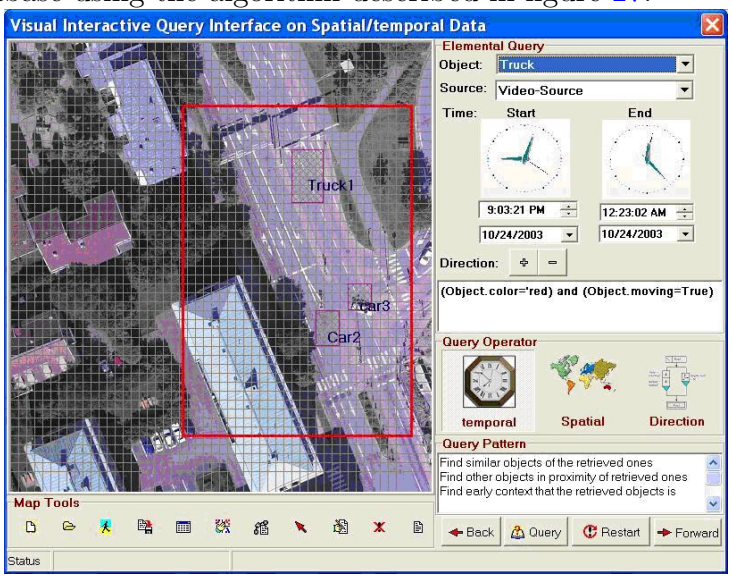

Figure 29: The Query Interface of the System V3.0

The main user interface is shown in figure 29. On the left-hand side of the main window there are some objects shown on an image that is the background. The red rectangle on the image shows the area of interest (AOI) of the current query. As we discussed in the previous sections, an elementary query has five variables to be specified: Object, Source, Time (TOI), Space (AOI) and Direction. The AOI is shown on the image. The other four items, together with other information, are shown on the right hand side: (from top to bottom) elementary query, query operator, query patterns and four functional buttons. Query patterns are shown by their linguistic labels.

Every time an elementary query is completed, query pattern matching is performed. For example, the current query path is $QP_1 = (Q_1, Q_2, \cdots, Q_i)$. When the query Q_{i+1} is completed, the query path becomes $QP_2 = (Q_1, Q_2, \cdots, Q_i, Q_{i+1})$. All the query patterns $QPN =< QP, S, L >$ in $S_{pattern}$ with $Match(QPN, QP_1)$ are shown to the user to choose

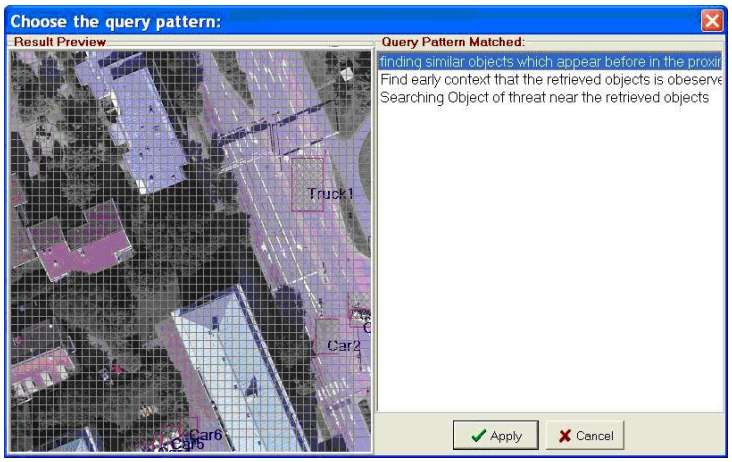

Figure 30: The User Interface to Select Matched Patterns

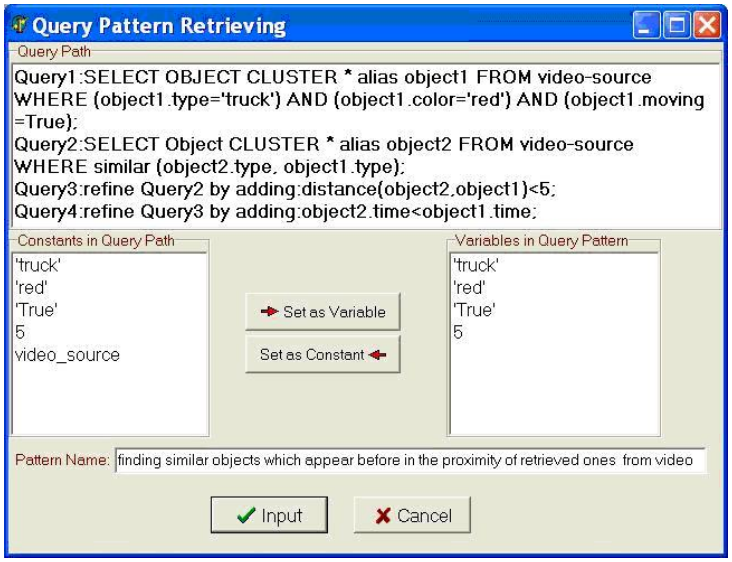

Figure 31: The User Interface to Specify Query Patterns

from. They are listed by their linguistic labels as shown in figure 30. A result preview is also shown to let the user know what the result will look like if this query pattern is chosen.

As we discussed before, the query patterns are retrieved dynamically. Initially there are

only predefined query patterns in the query pattern set $S_{pattern}$. When a query is completed, the query path will be shown and the user is asked to choose the variables and input a linguistic tag for each variable. The interface for users to specify query patterns is shown in figure 31. The query path is shown on the top, and the constans in the path are listed blow it. Using the buttons "–>Set as Variable" and "Set as Constant<–" in the middle, users can set any constant as a variable with any type/scope or vice versa. Once users finish and click the "input" button, a new query pattern is generated and added to the query pattern set $S_{pattern}$ using the algorithm described in figure 27.

5.5.2 Query Processing Experiments on System V3.0

Table 7: Experiment 4

Item	**Detail**
Title	Experiments on System V3.0
Goal	Verify the usability of the incremental query method with the post query reasoner (i.e. utilizing predefined query patterns and learning new patterns from user inputs.)
Description	Participators are asked to perform queries expressed in natural language using the experimental system V3.0. Users input queries by either building a new query or using query patterns. When done, users are asked to manually input the query patterns, and the result is visualized on the map which is easy to be verified.
Designed by	Xin Li
Participators	8 graduate students, University of Pittsburgh

In order to analyze performance of the proposed incremental query with the post query reasoner, I designed an query processing experiment on system V3.0. Table 7 lists some important attributes of this experiment. Eight end users have kindly participated in this study. Similar as the experiments on system V1.0 (see section 3.6.2) and system V2.0 (see section 4.6.2), the experiment includes the following three steps:

- **Step1**: assuming users have no pre-knowledge on incremental query and query pattern, we give them a very quick tutorial (30 min) by showing several examples on our system.
- **Step2**: we carefully choose three most frequently issued queries as benchmark queries (see Appendix A), and tell users the queries in English. Then ask them to perform these queries one by one using our system. Note that the query patterns that can match these queries are stored in the pattern database beforehand; however, users can choose whether or not to use query patterns by themselves.
- **Step3**: we monitor the time that users spend on each query and the system resource usage during this period.

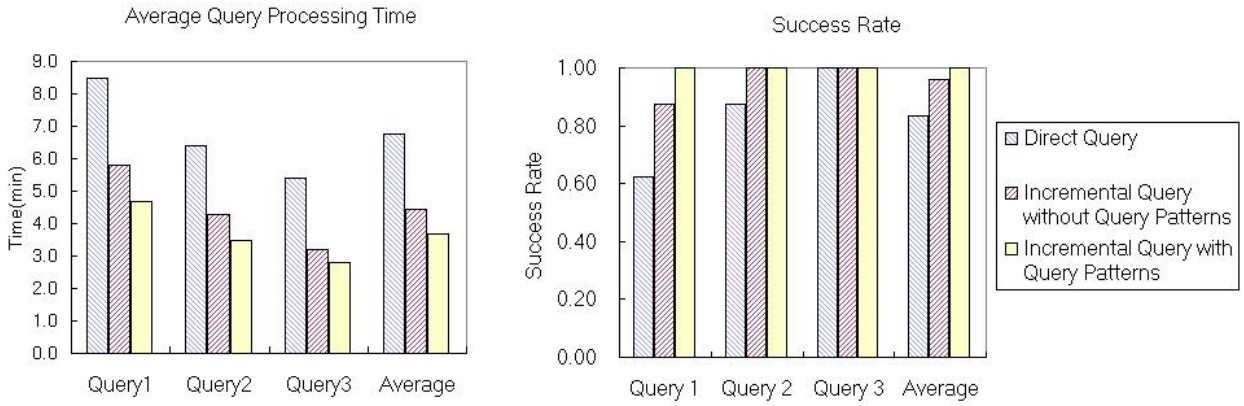

Figure 32: Result Analysis of Experiment 4

A threshold δ of response time (i.e. 15 min) is defined: if users can finish a query (from formulating a query until getting a satisfied result) within δ time, it is defined as a successful case, otherwise it is a failure. To compare with the performances of the direct query and the incremental query without query patterns, I plot the time and failure rate of this experiment together with the experiment 1 (see table 2) and experiment 3 (see table 5) in figure 32.

As shown in figure 32, the incremental query can save 35.3% of processing time compared to the direct query. Furthermore, with the post query reasoner that can utilize the query patterns, it can save 45.6% of processing time compared to the direct query. At the same time, the success rate also has a significant improvement while applying query patterns into the incremental queries.

It also proves to be an efficient way for users to input new query patterns using the interface shown in figure 31. It is convenient that users can define some customized query patterns for their personal needs.

5.5.3 Pattern Learning Experiments on System V3.0

Table 8: Experiment 5

Item	**Detail**
Title	Automatically Pattern Learning
Goal	Verify the usability of the query pattern auto-learning algorithm (i.e. learn query patterns without user efforts)
Description	Participators are asked to perform queries whatever they are interested in using the experimental system V3.0. Users input queries by either building a new query from the scratch or using pre-defined query patterns. Query history is recorded for learning new query patterns using the proposed algorithm.
Designed by	Xin Li
Participators	8 graduate students, University of Pittsburgh

An experiment has been designed to verify the proposed the pattern auto-learning algorithm described in section 5.4.4. Some important attributes of this experiments are listed in table 8.

Because the primary goal of this experiment is to testify whether or not the reasoner can identify new patterns form user query history, the participators are not requested to issue any predefined queries. Instead, they are asked to query anything that they feel interested. A variety of queries are input into the reasoner, and then query patterns are generated using the auto-learning algorithm described in figure 28. To summarize, the experiment includes the following steps:

- **Step1**: assuming users have no pre-knowledge on incremental query, we give them a very

quick tutorial (30 min) by showing several examples using our system.

- **Step2**: asking users to query whatever they are interested in based on the given scenario, i.e., a secure parking lot with a variety of vehicles. Users are also encouraged to compose complex queries, e.g. multiple objects with many constraints.
- **Step3**: sending the queries into the post query reasoner that can generate new query patterns. Manually check the new patterns in the pattern database to evaluate the learning algorithm.

Because our query learning algorithm (figure 28) depends on the coverage function on query history H_{path} to determine the variables, when H_{path} is empty the algorithm chooses the variables almost as same as random selection. As a result, the pattern learning process is extremely unpredictable and error-prone at the very beginning of the experiments. In fact, this problem can be categorized as a *bootstrap problem* which has also been identified in many other learning processes [Mit97]. To resolve this problem, I simply defer the learning process until a minimum number of query paths get accumulated in H_{path}. The results turn out to be very promising. Many reasonable patterns are successfully learned and added into the pattern database in our experiment. What follows are examples of the patterns learned in the experiment.

Example 5.5.1 *The following pattern is learned from the query history in the experiment.*
Query Patten $QPN = (QP, S, L)$*;*

- QP *is a query path,* $QP = (Q_1, Q_2)$*;* $Q_2 =< q_1, \tau_1, Q_1 >$*;*
  in which,
  Q_1 = <*type1=“car”, LR/IR/CCD, AOI, NOW, object.color=color1, i.e. “black”* >

  $\tau_1(object_j, object_k) = Beside(object_j, object_k)$

  q_1 = <*type2=“truck”, LR/IR/CCD, AOI, NOW, object.color=color2, i.e. “red”* >
- S *is a set of variables,* $S =$ { *type1 : Vehicle , type2: Vehicle, color1 : Color, color2 : Color*}*;*
- L *is a linguistic label,* $L =$ *“ type1 : Vehicle , type2: Vehicle, color1 : Color, color2 : Color”;*

Literally the query in this example is "find a red truck beside a black car in given area AOI now". The corresponding pattern learned from it is "find an vehicle with a color beside another vehicle with another color in given area AOI now". Obviously, it is a meaningful pattern, which could be frequently repeated in the described scenario.

Example 5.5.2 *The following pattern is learned from the query history in the experiment.*
Query Patten $QPN = (QP, S, L)$;

- QP *is a query path,* $QP = (Q_1, Q_2)$; $Q_2 =< q_1, \tau_1, Q_1 >$;
  in which,
  Q_1 = <*type1="car", LR/IR/CCD, location1 = "Area 1", NOW, object.color =color1, i.e. "red"* >

  $\tau_1(object_j, object_k) = InFrontOf(object_j, object_k)$

  q_1 = <*type2="truck", LR/IR/CCD, AOI, NOW, object.moving = TRUE* >
- S *is a set of variables,* $S =$ { *type1 : Vehicle, type2 : Vehicle , color1 : Vehicle , location1 : Area*};
- L *is a linguistic label,* $L =$ " *type1 : Vehicle, type2 : Vehicle , color1 : Vehicle , location1 : Area";*

Literally the query in this example is "find a moving truck in front of a red car in area 1 now". The corresponding pattern learned from it is "find a moving vehicle in front of another vehicle with a color in any given area now". Obviously, it is a meaningful pattern, which could be frequently repeated in the described scenario.

Example 5.5.3 *The following pattern is learned from the query history in the experiment.*
Query Patten $QPN = (QP, S, L)$;

- QP *is a query path,* $QP = (Q_1, Q_2)$; $Q_2 =< q_1, \tau_1, Q_1 >$;
  in which,
  Q_1 = <*type1="car", LR/IR/CCD, location1 = "Gate A", time1 = 9:00am – 10:00am, object.moving = TRUE* >

  $\tau_1(object_j, object_k) = associate(object_j, object_k)$

q_1 = <*type1="car", LR/IR/CCD, location2 = "Area 1", NOW, object.moving = FALSE* >

- *S is a set of variables, S = { type1 : Vehicle, time1 : TimeConstant, location1 : Gate , location2 : Area};*
- *L is a linguistic label, L = " type1 : Vehicle, time1 : TimeConstant, location1 : Gate , location2 : Area";*

Literally the query in this example is "find a car currently parked in area 1 which passed through Gate A during 9:00am – 10:00am". The corresponding pattern learned from it is "find a static vehicle in any given area which was spot to move through a given gate during a given time". Obviously, it is also a meaningful pattern, which could be frequently repeated in the described scenario.

Although the proposed algorithm proves to be able to learn reasonable query patterns automatically, it is still hindered by the bootstrap problem discussed above. For this reason, in practice I use a hybrid method which combines the manually pattern input and automatically pattern retrieving. A coverage threshold β is defined for any query path QP that users construct: if any query pattern generated from QP can cover more than β query paths in the query history, it is qualified to be learned as a query pattern automatically; otherwise, the users will be asked to select variables and their types, i.e. the query pattern is generated manually.

5.6 SUMMARY

In this chapter, several novel techniques are proposed to identify and utilize the query patterns, which are reusable templates in the incremental queries. Particularly, query pattern is defined as a domain independent concept so that it can formulate the general fusion tasks. A post query reasoner equipped with pattern matching and learning capability is designed to apply query patterns into the processing of incremental queries. The knowledge based pattern matching and learning techniques are described in detail. Demonstrated by the

experiments, the query pattern and related techniques can effectively elevate the system performance and lower the user mental load.

6.0 RELATED WORK

In this chapter, I review the works that relate to our query-based information retrieval approach in fusion systems, and discuss in detail how our solution advances the state of the art. Because our works mainly span the research areas of both information fusion and query system, I review the related work from fusion research community (section 6.1) and database research community (section 6.2) respectively, and compare them with our solution.

6.1 INFORMATION FUSION SYSTEMS

There are numerous works in the literature that address information retrieval problem in fusion systems from various perspectives e.g. fusion model, fusion system, and fusion techniques, here I only focus on the system level methodology and techniques, and describe several important approaches on fusion system that closely relate to our works.

Back to the early stage of information fusion research, fusion systems were all designed as *specialized systems* where a variety of fusion-related functions were tightly coupled together. Frequently this kind of systems dedicatedly aims at problems in a specific domain, and the fusion processes are only conceptually classified into different levels, but implemented together in a single procedure. M. Nichols has briefly surveyed 76 fusion systems for DoD applications such as multisensor battlefield surveillance system and missile tracking system in [Nic01]. The overall architectures and the types of fusion processing of these systems are assessed. From the descriptions, we can see that they are all designed as specialized systems. One of the most important advantages of specialized fusion systems is: they can achieve the maximum efficiency for some specific fusion problems if they are well designed. For this

reason, nowadays there are still some fusion applications implemented as specialized systems. However, frequently it is a hard problem to design a specialized fusion system, because the methodology and techniques used in one system usually can not be simply reused into others. With the increasing needs for various fusion systems, a more flexible and reusable framework becomes extremely important.

Due to the increasing complexity of fusion applications in recent years, many researchers apply the software engineering techniques into the design and implementation of fusion system. C. Bowman et al. proposed a systematic process to develop a fusion system using engineering methodology in [BS01]. A standard model is described for representing the requirement, design, and performance of a fusion system. Particularly, a fusion process is interpreted as a network of processing nodes, which is called *fusion tree*. Similar to the source dependency graph (SDG) in our approach, the fusion tree serves as a basic tool for the fusion process refinement and optimization, and resource management. However, unlike the source dependency graph (SDG), the fusion tree is not dynamically constructed based upon user information requests, but predefined either manually or automatically through some engineering techniques according to the system requirements. As a result, the fusion systems designed through this way can only handle one kind of information retrieval requests with only minor variations – it is enough for some fusion applications e.g. in a multisensor missile tracking system, the only target is *missile* and the only interested attributes are *speed* and *position*. However, in general it is still not a broadly applicable approach yet.

As the fruit of rapid developments on the techniques of both information fusion and software engineering, the component-based approach on the design of fusion systems has been widely investigated. In a component-based fusion system, fusion-related functions are encapsulated into different components, and the interfaces between components are carefully designed to achieve the maximum flexibility and reliability. In [WSX01], J. Wang et al. described a component framework for constructing flexible, robust and efficient software applications for multisensor fusion system. The whole fusion system is divided into three parts: man/machine interface, logic part and database. These parts are specified using component object model (COM), and further broken down into four layers: sensor driver level (SDL), logical sensor level (LSL), fusion unit level (FUL), and task unit level (TUL).

At each level, there are some components with different functions. Similar approaches that apply the concept of software components into the design of fusion systems can be found in [LOdlCD+98, ZS04, Ant97]. These systems prove to be able to achieve a certain degree of flexibility. However, in most cases the fusion component itself is still highly domain specific – it has to be modified or replaced while applying the system into different applications.

In recent years, the agent-based technology has been intensively studied in a broad range of application areas. Mobile agent technology is beginning to be applied to information retrieval from multimedia databases [KDB01]. Meanwhile more and more fusion systems employ agents to handle heterogeneous data sources [PHG+99, AKSss, DLSssa, DLSssb, RJX03]. Compared to component, agent is much more autonomous which lends itself to being more flexible and intelligent. Multiple agents can exchange information and cooperate with each other to achieve fusion. For example, in [AKSss], H. Alex et al. proposed a Hierarchical Collective Agent Network (HCAN) in the fusion system with a set of networked distributive data sources (i.e. sensors), where an anatomy of the agent modules for enabling intelligent data fusion and management is presented in detail. There is no doubt that agent techniques can contribute the fusion processes with a certain degree of intelligence and flexibility. However, the mobile agents in these systems are still highly domain-specific and usually depend on ad-hoc, 'hardwired' programs to implement them.

Compared to all the approaches described above, **our method** on the design of fusion systems is based upon the commonalities underlying various fusion applications (i.e., the general fusion system model described in chapter 2), and hence it can be applied into a much broader range of applications. No matter what kind of data sources (e.g. sensors or non-sensors), and what kind of user information requests (e.g. any query with any contraints), they can be handled uniformly by the proposed techniques e.g. source dependence graph, query refinement and optimization, incremental query, and query pattern. A good example is that our method, which is basically developed from the IVET system, can be easily employed into different application areas, such as the local search system (example 1.1.2). From this point of view, it is indeed a broadly applicable approach.

6.2 QUERY SYSTEMS

Query processing is one of the most important topics in database research. In this section, I review some important related works from database research community – query systems (section 6.2.1) and query language (section 6.2.2) on heterogeneous information sources, and query pattern (section 6.2.3). Our approach is also compared to them respectively.

6.2.1 Query Approaches on Heterogeneous Information Sources

The needs for information retrieval on the basis of multiple information sources have been dramatically increasing in recent years, especially with the explosive growth of the online information. Several important query approaches from the perspective of database research have been proposed to resolve the heterogeneity of data sources, which lends themselves to a uniform information access for end users. What follows is a brief review of some typical systems.

In [SSKT95, SSKS95], L. Shklar et al. proposed an information retrieval system named *InfoHarness*, which constructs a uniform access to a large amount of heterogeneous online data. The different data schemas are reconciled by the *metadata* automatically extracted from the original data sources. The metadata is defined by a stable hierarchy of abstract classes and an extensible hierarchy of terminal classes, and hence they can be applied to the new data sources without major modifications. For the end users, they can simply query cross the data sources using general web browsers.

A. Levy et al. described a query system named *Information Manifold* in [LRO96], which retrieves the information from over a hundred of WWW sources into databases, and provides a uniform query interface to answer user complex queries which could be far beyond the keyword-based search. The query execution plan is generated based on the descriptions of the contents and capability of sources (which is similar as the ontology knowledge about data sources in our approach). However, the descriptions about sources are mainly obtained manually. It means that users have to create the description when they extend the query system to cover any new data sources.

M. Genesereth et al. presented *Infomaster* in [GKD97, DG97], which is a information retrieval system over multiple distributed heterogeneous information sources on the Internet. For end users, it works as exactly same as a centralized, homogeneous information retrieval system. The data heterogeneity is harmonized by a global data schema created through the rule-based reasoning. However, all the rules are created and maintained manually. The performance of the system largely depends on the quality of the rules defined.

The *LED* system, which is described by A. Doan et al. in [DDH01, Doa03], constructs a uniform access to a multitude of data sources through a single mediated schema. Multiple learners on the basis of different machine learning techniques are employed to match the data schema from the heterogeneous data sources to a global mediated schema. Because basically it is a supervised learning approach, a reasonable amount of training data is needed. Some parts of the schema (e.g. tags) could not be able to get matched simply because none of them appeared in the training data set.

Compared to the systems described above, **our solution** possesses some unique features with significant value, although we are all addressing the information retrieval problem from multiple heterogeneous data sources. As mentioned by P. Anokhin [Ano01], one of most important issues for information retrieval from heterogeneous sources is to *resolve the potential information conflicts among different data sources*. These conflicts primarily come from the following three levels from high to low:

1. **Schema Level**: the conflicts are caused by the different schemas of data sources.
2. **Representation Level**: the conflicts are caused by the different natural language and measurement system used by data sources.
3. **Data Level**: the conflicts are caused by the different data values about the same objects from data sources.

A crucial point is that the lower level conflicts can be only observed when all higher level conflicts are resolved. For example, the data level conflicts will not appear until the data sources get matched on schema and representation levels. In other words, we can not determine whether or not data values conflict each other until they are identified to be about same objects and under comparable measurement systems.

The resolutions of the conflicts at these three levels are of potentially equal importance because system can not work without any of them. However, various works have different emphases: most current works from database research community are focusing on the first two levels of conflicts. As the systems described above, numerous approaches is proposed to resolve the conflicts at the schema level by various means [SSKT95, SSKS95, LRO96, GKD97, DG97, DDH01]. A good number of approaches are described to resolve the conflicts at the representation level [LRO96, Doa03]. However, as far as we studied, very few approaches from database research community explicitly address the possible conflicts at the data level. **Our solution** is just the work that aims to fill this blank. Unique from the previous approaches, our system can generate the practical query execution plan to resolve the data level conflicts based on the description of data sources and fusion algorithms (i.e. ontology knowledge base).

As a matter of fact, the works proposed to resolve the conflicts at schema and representation levels are widely recognized as the research about *information integration*. At the same time, the resolution of the data level conflicts belongs to the techniques of *information fusion*. As briefly discussed in section 1.6.2, a major difference between information integration and information fusion is that the data sources are much more controllable in fusion systems. More precisely, although they are both addressing the problem of information retrieval from heterogeneous data sources, fusion systems usually emphasize on the data level conflicts while assuming that it is a relatively trivial problem to resolve the conflicts at schema and representation levels or these kinds of conflicts are already resolved; on the contrary integration systems mainly focus on the conflicts at the schema and representation levels and frequently ignore the data level inconsistences.

From this point of view, **our solution** serves as a novel approach with significant value, which uniquely addresses the fusion problem in query system over multiple heterogeneous data sources.

6.2.2 Query Language

Heterogeneous data sources pose some unique challenges for query systems – query language is one of them. Back to 1989, W. Litwin proposed MSQL [LAZ+89] – an extension of standard query language SQL, which is designed for a nonprocedural manipulation of data in different and nonintegrated relational databases. Besides the concept of table, sets of tables called *multitables* and *multirelations* are introduced to manipulate the tables referring same entities from different databases. Using MSQL, user query over multiple autonomous databases is expressed in a single statement. Inspired by MSQL, in [GLRS93] J. Grant et al. proposed a theoretical foundation for such languages by presenting a multirelational algebra and caculus based upon the relational algebra and calculus.

On the basis of these works, in recent years many query languages have been proposed aiming at effective and flexible interoperability in multi-database systems. For example, SchemaSQL proposed by L. Lakshmanan et al. in [LSS96, LSS99, LSS01], offers the capability of uniform manipulation of data and meta-data in relational multi-database systems. In [GL98], F. Gingras proposed a multi-dimensional language called nD-SQL, which supports queries that interoperate amongst multiple heterogeneous relational sources.

Compare the query languages mentioned above to **the query language used in our system** – ΣQL which is proposed by S. K. Chang et al in [CJC04], there are several commonalities: all languages are designed for information retrieval on multiple heterogeneous data sources, and they all retain the flavor of SQL; all of them emphasize the interoperability among multiple data sources. However, ΣQL possesses some unique features which other query languages are not equipped with.

1. Unlike the other languages, ΣQL explicitly supports fusion operations.
2. ΣQL are not restricted for only relational data sources, it can be applied to the non-database sources.
3. Using the clause "CLUSTER", ΣQL offers a natural way to express spatial/temporal queries.

For the detail about ΣQL, readers can refer to [CJC04].

Furthermore, **the incremental query proposed in this dissertation** can also be envisioned as a new query language on the basis of ΣQL. It retains the advantages of ΣQL because every incremental query will be eventually interpreted into ΣQL. Meanwhile it can break a complex query into several uniform, simple iterations, which can greatly lower the workload of query building as well as query execution.

6.2.3 Query Pattern

The patterns of user information needs are widely investigated in many information retrieval systems. Particularly in recent years, with the blooming of web search engines such as Google, Yahoo and so on, numerous approaches have been proposed to identify the query pattern in web search process.

In [JSBS98], M. Jason et al. presented a in-depth analysis of user queries based upon the transaction logs created during online keywords-based search. The user information needs are analyzed at three levels: 1) **Query Level**: every keywords-based query is recorded in a simple and uniform format, which is similar as the *elementary query* in our approach; 2) **Session Level**: it includes multiple queries in one session that shows the process of query refinement by users, which is similar as the *query path* in our approach; 3) **Term Level**: it computes the rank and distribution of frequently searched terms. The report described a real slice of life on the web, and presented a large-scale, quantitative study on user information needs. More similar research based on the analysis of query logs can be found in [SMHM99, SWJS01]. By different means, these approaches can identify various patterns in user information needs. However, another important problem – how to utilize these patterns to improve the query perfermance, is not addressed with enough efforts yet.

In [LL03], R.L. Liu et al. described implicit query pattern approach in a query system on a hierarchical information space, which can identify user information needs progressively through the interactions with users. The user query patterns are retrieved by text mining on the documents that user searched. The system does not record these patterns directly, but updates the user's category profile based on them. Through the category profile, system can map the user information needs to suitable nodes in the hierarchical information space

without requiring users to enter long queries, conduct many interactions, and suffer heavy cognitive load – which is similar as the goal of the query pattern in our approach.

Compared to all the approaches described above, **our query pattern schema** is proposed in a much simpler way. The simplicity is its major strength – it is a natural extension of the incremental query, which is defined as an abstract query from a group of similar query paths. Even end users can directly construct and modify query patterns by themselves. With the support of ontology knowledge base, query patterns in our approach can cover the most frequently used queries and make them easy to construct and process. From this point of view, it describes not only the pattern of user information needs, but also the pattern of the entire process of query handling.

7.0 CONCLUSION

Given the rapid proliferation and the growing size of applications today, the information retrieval on the basis of fusion operations becomes a crucial problem with dramatically increasing interests from both practical and theoretical perspectives. Our method provides a concrete, broadly applicable solution for the problem, which has been fully verified in the experimental systems. In this chapter, first I briefly recap the contributions of this dissertation (section 7.1), and then discuss the future directions (section 7.2).

7.1 KEY CONTRIBUTIONS

This dissertation makes three major contributions:

- First of all, this dissertation contributes *a broadly applicable approach* for information retrieval in fusion systems. As the related works discussed in chapter 6, up to the time of this dissertation, most previous fusion systems are highly domain-specific, and most existed query systems do not address the fusion problem with enough efforts. Our solution describes a domain-independent approach on general query systems supporting information fusion. As shown in figure 26 in chapter 5, the overall system architecture consists of query processor, query interface, post query reasoner, ontology knowledge base and so on. The techniques for these components are described in detail and verified by the experiments, which can be applied in a wide range of information fusion applications.

- The second major contribution of this dissertation is *a novel incremental query method*

(chapter 4), which successfully resolves the accumulated complexity of information retrieval in multi-object and multi-source scenarios. The incremental query is originally motivated by resolving the query building bottleneck; however, it is not just a new query interface, but indeed a novel query processing method including query interface, query language, and processing techniques. Demonstrated by the experiments, it can significantly improve the system usability and lower the user mental load.

- The third major contribution of this dissertation is *the techniques to retrieve and utilize query pattern* (chapter 5). With these techniques, the fusion systems can achieve a significant degree of intelligence. The system can keep improving by referring the previous experiences formulated as query patterns using the post query reasoner that can learn and match query patterns. Although in our method the query pattern is envisioned as a natural extension of the incremental query, it is indeed a concept which does not depend on any query schemas. Moreover, query pattern can also serve as an informative data source to retrieve the ontology knowledge. There is no doubt that the query pattern and related techniques lead a promising direction for further research towards intelligent fusion systems.

7.2 FUTURE DIRECTIONS

We have made some significant progress in developing a solid frame work and effective techniques for information retrieval on multiple heterogeneous data sources, but still substantial works remains towards a broadly applicable intelligent fusion system. In what follows I briefly describe some interesting directions of our future work.

7.2.1 Solution Across Application Areas

The best way to demonstrate that our approach is a broadly applicable solution is to employ it into different application areas. In fact, we have successfully applied the incremental query approach into several non-sensor-based fusion systems. For example, we applied fusion

query into user profiling in e-learning system and local search system. The works in these application areas will eventually benefit the current solution which is originally motivated by the IVET system.

7.2.2 Statistical Query Pattern Learning

Query pattern learning is a crucial process by which system can keep improving over the previous experience. Currently we employ a heuristic function (see *coverage* function defined in section 5.4.4) as the guideline for the automatic pattern learning. It works fine on a moderate-size set of query paths. However, it is conceivable that for the accurate assessments of patterns on large set of query experiments, some advanced statistical learning techniques such as Bayesian network and Dempster-Shafer's method will be more suitable. More research will be done along this line.

7.2.3 Dynamic Exchange of Information from Query Patten and Ontology

As we discussed in section 5.4.5, it is a promising direction to construct ontology knowledge from query patterns. Generally speaking, it could be extremely cumbersome for end users to construct and maintain ontology knowledge. However, users can easily create query patterns with our incremental query schema. Definitely it is worth the further investigation to build a feasible connection from query patterns to ontology knowledge, so that users can eventually construct and update ontology knowledge by manipulating query patterns.

7.2.4 Efficient User Interaction

Our incremental query approach mainly depends on the interactions with end users to retrieval right information from right data sources. Even if the system can automatically generate the query based on the matched query patterns, it still needs users to verify it. Therefore, an efficient user interaction solution is of great importance to our approach. As described in the experimental systems, much effort has been done towards an efficient visual user interface, e.g. visual query interface (see figure 12, 22, and 29), the visualization of

query processing (see figure 14), the selection of query patterns (see figure 30), and so on. However, there are still a lot of works to do. The key is to discover how to minimize the user mental load but maximize the impact of the user feedback at the same time.

7.2.5 Combination with Information Integration

As we discussed in section 6.2.1, the research on information fusion and information integration are both mainly addressing information retrieval problem on heterogeneous data sources, but focus on different perspectives. However, within applications in the real world, frequently both fusion and integration problems are raised at the same time. It will be of great value if we can make them consolidate with each other.

BIBLIOGRAPHY

[AEG03] A. Abdelmoty and B. El-Geresy. Qualitative tools to support visual querying in large spatial databases. In *Proceeding of the Ninth International Conference on Distributed Multimedia Systems (DMS)*, pages 300–305, Miami,FL, Sep. 2003.

[AHK98] Yigal Arens, Chun-Nan Hsu, and Craig A. Knoblock. Query processing in the sims information mediator. In *Readings in agents*, pages 82–90. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1998.

[AKSss] Hitha Alex, Mohan Kumar, and Behrooz Shirazi. Midfusion: An adaptive middleware for information fusion in sensor network applications. *Information Fusion*, In Press.

[Ano01] Philipp Anokhin. *Data Inconsistency Detection and Resolution in the Integration of Heterogeneous Information Sources*. PhD thesis, George Mason University, 2001.

[Ant97] R.T. Antony. Database support to data fusion automation. *Proceedings of the IEEE*, 85:39–53, Jan. 1997.

[BS01] Christopher L. Bowman and Alan N. Stainberg. A systems engineering approach for implementing data fusion systems. In Hall and Llinas [HL01], pages 16.1–16.38.

[CCC+04] G. Casella, S.K. Chang, G. Costagliola, E. Jungert, Xin Li, and T. Horney. An architecture for interactive query refinement in sensor-based information fusion systems. In *Proceeding of the Tenth International Conference on Distributed Multimedia Systems (DMS)*, pages 315–321, San Francisco Bay,CA, Sep. 2004.

[CDH+02] S.K. Chang, W.Y. Dai, S. Hughes, P. Lakkavaram, and Xin Li. Evolutionary query processing, fusion and visualization. In *Proceeding of the Eighth International Conference on Distributed Multimedia Systems (DMS)*, San Francisco Bay, CA, Sep. 2002.

[CDOP02] G. Costagliola, A. Delucia, S. Orefice, and G. Polese. A classification framework to support the design of visual languages. *Journal of Visual Languages and Computing*, 13(6):573–600, Dec. 2002.

[CJ04] S.K. Chang and E. Jungert. Iterative information fusion using a reasoner for objects with uninformative belief values. In *Proceeding of the International Conference on Information Fusion (Fusion)*, Stockholm, Sweden, Jun. 2004.

[CJC04] S.K. Chang, E. Jungert, and G. Costagliola. Querying distributed multimedia databases and data sources for sensor data fusion. *IEEE Transactions on Multimedia*, 6(5):672–687, Oct. 2004.

[CJL06] S.K. Chang, Erland Jungert, and Xin Li. A progressive query language and an interactive reasoner for information fusion support. *Journal of Information Fusion*, 2006.

[CLRS01] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein, editors. *Introduction to Algorithms (Second Edition)*. The MIT Press, Boston, NY, 2001.

[DDH01] A. Doan, Pedro Domingos, and Alon Y. Halevy. Reconciling schemas of disparate data sources: a machine-learning approach. In *Proceedings of the 2001 ACM SIGMOD*, pages 509–520, 2001.

[DG97] Oliver M. Duschka and Michael R. Genesereth. Query planning in infomaster. In *Proceedings of the 1997 ACM symposium on Applied computing (SAC)*, pages 109–111, 1997.

[DLSssa] Patrick Dohertya, Witold Lukaszewicza, and Andrzej Szalasa. Communication between agents with heterogeneous perceptual capabilities. *Information Fusion*, In Press.

[DLSssb] Patrick Dohertya, Witold Lukaszewicza, and Andrzej Szalasa. Hierarchical Collective Agent Network (HCAN) for efficient fusion and management of multiple networked sensors. *Information Fusion*, In Press.

[DM03] A. Doan and R. McCann. Building data integration systems: A mass collaboration approach. In *Proceeding of the Sixth International Workshop on the Web and Databases*, pages 25–30, San Diego, CA, Jun. 2003.

[DMD+03] A. Doan, J. Madhavan, R. Dhamankar, P. Domingos, and A. Halevy. Learning to match ontologies on the semantic web. *VLDB Journal, Special Issue on the Semantic Web*, 12(4):303–319, 2003.

[Doa03] A. Doan. *Learning to Map between Structured Representations of Data*. PhD thesis, University of Washington, 2003.

[GKD97] Michael R. Genesereth, Arthur M. Keller, and Oliver M. Duschka. Infomaster: an information integration system. In *Proceedings of ACM SIGMOD*, pages 539–542, 1997.

[GL98] Frederic Gingras and Laks V. S. Lakshmanan. nD-SQL: A Multi-Dimensional Language for Interoperability and OLAP. In *Proceedings of the 24th International Conference on Very Large Databases (VLDB)*, pages 134–145, San Francisco, CA, USA, 1998.

[GLRS93] John Grant, Witold Litwin, Nick Roussopoulos, and Timos Sellis. Query languages for relational multidatabases. *The VLDB Journal*, 2(2):153–172, 1993.

[GMPDQ+97] H. Garcia-Molina, Y. Papakonstantinou, A. Rajaraman D. Quass, J Ullman Y. Sagiv, and J. Widom. The TSIMMIS project: Integration of heterogeneous Information sources. *Journal of Intelligent Information System*, 8(2), 1997.

[Goo] Google Local. http://local.google.com/, accessed on Nov. 11, 2005.

[GSGN+01] C. A. Goble, R. Stevens, S. Bechhofer G. Ng, N. W. Paton, P. G. Baker, M. Peim, and A. Brass. Transparent access to multiple bioinformatics information sources. *IBM Systems Journal*, 40(2), 2001.

[HHT01] Mary Jane Hall, Capt. Sonya A. Hall, and Timothy Tate. Removing the hci bottle neck: How the human-computer interface affects the performance of data fusion systems. In Hall and Llinas [HL01], pages 19.1–19.12.

[Him05] M. Himmelstein. Local search: the internet is the yellow pages. *Computer*, 38:26–34, Feb. 2005.

[HJF03] T. Horney, E. Jungert, and M. Folkesson. An ontology controlled data fusion process for query language. In *Proceeding of the International Conference on Information Fusion*, Cairns, Australia, Jul. 2003.

[HL01] D. L. Hall and J. Llinas, editors. *Handbook of Multisensor Data Fusion*. CRC Press, New York, 2001.

[HM04] D. L. Hall and S. A. H. McMullen, editors. *Mathematical Techniques in Multisensor Data Fusion*. Artech House Publishers, 2004.

[JC84] M. Jarke and J. Cohen. Query optimization in database systems. *ACM Computing Surveys*, 16(2):111–152, Jun. 1984.

[JSBS98] Bernard J. Jansen, Amanda Spink, Judy Bateman, and Tefko Saracevic. Real life information retrieval: a study of user queries on the web. *SIGIR Forum*, 32(1):5–17, 1998.

[KDB01] H. Kosch, M. Doller, and L. Boszormenyi. Content-based indexing and retrieval supported by mobile agent technology. In M. Tucci, editor, *Multimedia Databases and Image Communication*, pages 152–166. Springer-Verlag, Berlin, 2001.

[Kni89] K. Knight. Unification: A multidisciplinary survey. *ACM Computer Surveys*, 21:93–121, 1989.

[LAZ+89] W. Litwin, V. Abdellatif, A. Zeroual, B. Nicolas, and Ph. Vigier. Msql. a multidatabase language. *Information sciences*, 49:59–101, 1989.

[LB85] Hector J. Levesque and Ronald J. Brachman. A fundamental tradeoff in knowledge representation and reasoning (revised version). In Hector J. Levesque and Ronald J. Brachman, editors, *Readings in Knowledge Representation*, pages 45–70. Morgan Kaufman Pub, 1985.

[LC95] Y. Lee and F. Chin. An iconic query language for topological relationships in gis. *International Journal of Geographical information Systems*, 9(1):24–46, 1995.

[LC04] Xin Li and S.K. Chang. An interactive visual query interface on spatial/temporal data. In *Proceeding of the Tenth International Conference on Distributed Multimedia Systems (DMS)*, pages 257–262, San Francisco Bay, CA, Sep. 2004.

[LC05] Xin Li and S.K. Chang. A personalized e-learning system based on user profile constructed using information fusion. In *Proceeding of the Eleventh International Conference on Distributed Multimedia Systems (DMS)*, pages 109–114, Banff, Canada, Sep. 2005.

[LL03] Rey-Long Liu and Wan-Jung Lin. Mining for interactive identification of users' information needs. *Inf. Syst.*, 28(7):815–833, 2003.

[LOdlCD+98] J.A. Lopez-Orozco, J.M. de la Cruz, E. Doninguez, E. Besada, and O.R. Polo. An open sensing architecture to autonomous mobile robots. In *Intelligent Control (ISIC)*, pages 610–615, San Francisco Bay, CA, Sep. 1998.

[LRO96] Alon Y. Levy, Anand Rajaraman, and Joann J. Ordille. Querying heterogeneous information sources using source descriptions. In *Proceedings of the 22nd International Conference on Very Large Databases (VLDB)*, pages 251–262, Bombay, India, 1996.

[LSS96] Laks V. S. Lakshmanan, Fereidoon Sadri, and Subbu N. Subramanian. Schemasql - a language for interoperability in relational multi-database systems. In *Proceedings of the 22nd International Conference on Very Large Databases (VLDB)*, pages 239–251, Bombay, India, 1996.

[LSS99] Laks V. S. Lakshmanan, Fereidoon Sadri, and Subbu N. Subramanian. On efficiently implementing schemasql on an sql database system. In *Proceedings of the 25th International Conference on Very Large Databases (VLDB)*, pages 471–483, Edinburgh, Scotland, 1999.

[LSS01] Laks V. S. Lakshmanan, Fereidoon Sadri, and Subbu N. Subramanian. Schemasql: An extension to sql for multidatabase interoperability. *ACM Trans. Database Syst.*, 26(4):476–519, 2001.

[McG02] Deborah L. McGuinness. Ontologies come of age. In Dieter Fensel, Jim Hendler, Henry Lieberman, and Wolfgang Wahlster, editors, *Spinning the Semantic Web: Bringing the World Wide Web to Its Full Potential*. MIT Press, Boston, NY, 2002.

[MIKS00] E. Mena, A. Illarramendi, V. Kashyap, and A. Sheth. OBSERVER : An approach for query processing in global information systems based on operation across pre-existing ontologies. *International Journal on Distributed and Parallel Databases*, 8(2):223–271, 2000.

[Mit97] Tom M. Mitchell, editor. *Machine Learning*. McGraw-Hill Science/Engineering/Math, New York, 1997.

[Nic01] Mary L. Nichols. A survey of multisensor data fusion systems. In Hall and Llinas [HL01], pages 22.1–22.7.

[NS02] A. Narayanan and T. Shaman. Iconic SQL: Practical Issues in the Querying of Database through Structured Iconic Expressions. *Journal of Visual Languages and Computing*, 13(6):623–647, Dec. 2002.

[PHG+99] A. Preece, K. Hui, P. Gray, P. Marti, T. Bench-Capon, D. Jones, and Z. Cui. The kraft architecture for knowledge fusion and transformation. In *Proceeding of the Conference of Expert System*, pages 294–299, Miami,FL, Sep. 1999.

[PK00] G. J. Pottie and W. J. Kaiser. Wireless integrated network sensors. *Communication of the ACM*, 43(5):51–58, 2000.

[RJX03] L. Ronnie, M. Johansson, and Ning Xiong. Perception management: an emerging concept for information fusion. *Information Fusion*, 4:231–234, Sep. 2003.

[Rus03] Stuart J. Russell, editor. *Artificial intelligence : a modern approach*. Prentice Hall/Pearson Education, Upper Saddle River, N.J., 2003.

[SBW98] Alan N. Steinberg, Christopher L. Bowman, and Franklin E. White. Revisions to the JDL data fusion model. *Proceedings of SPIE*, 3719, 1998.

[SBW04] A.N. Steinberg, C.L. Bowman, and F.E. White. Rethinking the JDL data fusion model. In *Proceeding of the NSSDF Conference*, JHAPL, Jun. 2004.

[Sch98] Alexander Schrijver, editor. *Theory of Linear and Integer Programming*. John Wiley and Sons, 1998.

[SMHM99] Craig Silverstein, Hannes Marais, Monika Henzinger, and Michael Moricz. Analysis of a very large web search engine query log. *SIGIR Forum*, 33(1):6–12, 1999.

[Sor70] H. W. Sorenson. Least-squares estimation: from gauss to kalman. *IEEE SPECTRUM*, 7:63–68, Jul. 1970.

[SSKS95] L. Shklar, A. Sheth, V. Kashyap, and K. Shah. InfoHarness: Use of Automatically Generated Metadata for Search and Retrieval of Heterogenous Information. In J. Iivari, K. Lyytinen, and M. Rossi, editors, *Proceedings of the Seventh International Conference CAiSE'95 on Advanced Information Systems Engineering*, volume 932, pages 217–230, Jyväskylä, Finland, 1995. Springer-Verlag.

[SSKT95] Leon Shklar, Amit Sheth, Vipul Kashyap, and Satish Thatte. InfoHarness: a system for search and retrieval of heterogeneous information. In *Proceedings of ACM SIGMOD*, pages 478–478, 1995.

[SVGM00] Tajana Simunic, Haris Vikalo, Peter Glynn, and Giovanni De Micheli. Energy efficient design of portable wireless systems. In *Proceeding of the international symposium on Low power electronics and design*, pages 49–54, Rapallo,Italy, Aug. 2000.

[SWJS01] Amanda Spink, Dietmar Wolfram, Major B. J. Jansen, and Tefko Saracevic. Searching the web: The public and their queries. *Journal of the American Society for Information Science and Technology*, 52(3):226–234, 2001.

[Tor03] Vicenc Torra, editor. *Information Fusion in Data Mining*. Springer, New York, 2003.

[Vad88] S. Vadera. A theory of unification. *Software Engineering Journal*, 3:149–160, Sep. 1988.

[Whi88] F.E. White. A model for data fusion. In *Proceeding of the First National Symposium on Sensor Fusion*, 1988.

[WL90] E. Waltz and J. Llinas, editors. *Multisensor data fusion*. Artect House, Boston, NY, 1990.

[WSX01] Jun Wang, Jianbo Su, and Yugeng Xi. COM-based software architecture for multisensor fusion system. *Information Fusion*, 2:261–270, Dec. 2001.

[WVV+01] H. Wache, T. Voele, U. Visser, H. Stuckenschmidt, G. Schuster, H. Neumann, and S. Huner. Ontology-based integration of information-a survey of existing

approaches. In *Proceeding of the Seventeenth International Joint Conference on Artificial Intelligence*, Seattle,WA, Aug. 2001.

[Yah] Yahoo Local. http://local.yahoo.com/, accessed on Nov. 11, 2005.

[YG03] Yong Yao and Johannes Gehrke. Query processing for sensor networks. In *Proceeding of the Conference on Innovative Data Systems Research (CIDR)*, Asilomar,CA, Jan. 2003.

[ZS04] Kejun Zhang and Jianbo Su. RTOS-based software architecture for multisensor fusion system. In *The 5th Asian Control Conference*, Melbourne, Australia, Jul. 2004.

APPENDIX A

BENCHMARK QUERIES IN THE IVET SYSTEM

I choose the following three queries as the *benchmark queries* to evaluate the system performance because they represent three typical kinds of queries in information fusion systems[1].

Query1: Find a red car followed by a truck within 10 minutes.

Direct Query:

```
SELECT object
CLUSTER ALIAS OBJ1, OBJ2
FROM
    SELECT t
    CLUSTER *
    FROM LR, IR, CCD
WHERE OBJ1.type="car" AND OBJ1.color="red" AND OBJ2.type="truck" AND
        OBJ1.time<OBJ2.time AND OBJ1.Time>OBJ2.time-10min
```

Incremental Query: $Q =< q_1, \tau_1, Q_1 >$;

in which,

$Q_1 =$ <"truck", LR/IR/CCD, AOI, TOI, object.moving = TRUE >

$\tau_1(object_j, object_k) = (object_j.t > object_k.t)$

[1]See section 5.3.1 for detail about the common tasks in information fusion systems.

AND $(object_j.time < object_k.time + 10min)$

q_1 = <"car", LR/IR/CCD, AOI, TOI, (object.color="red") AND (object.moving = TRUE)>

Query2: Find a black car in the area 1 which entered the parking lot through Gate A during 6:00pm – 9:00pm.

Direct Query:

SELECT $object_k$,
CLUSTER * alias $object_k$
FROM LR, IR, CCD
WHERE $Inside(GateA, object_k)$
AND $6:00pm < object_k.time < 9:00pm$
AND $associate(object_j, object_k)$
AND $object_j$ in
SELECT $object_i$
CLUSTER * alias $object_i$
FROM LR, IR, CCD
WHERE $Inside(Area1, object_i)$
AND $object_i.time = now$
AND $object_i.type =$ "*car*"
AND $object_i.color =$ "*black*"

Incremental Query $Q =< q_1, \tau_1, Q_1 >$;

in which,

Q_1 = <"car", LR/IR/CCD, Area 1, NOW, object.color="black">

$\tau_1(object_j, object_k) = associate(object_j, object_k)$

q_1 = <"car", LR/IR/CCD, Gate A, 6:00pm – 9:00pm, TRUE>

Query3: Did the results of some sensors conflict with each other (i.e. different types referring to the same object) while finding a truck passed Gate B at 7:00pm?

Direct Query

SELECT $object_k.type, object_k.sensor$

CLUSTER * alias $object_k$

FROM $DependencyTree$

WHERE $object_k.time = object_j.time$

AND $object_k.position = object_j.position$

AND $object_k.type \neq object_j.type$

AND $object_j$ in

SELECT $object_i.type, object_i.position$

CLUSTER * alias $object_i$

FROM LR, IR, CCD

WHERE $Inside(GateB, object_i)$

AND $object_i.time = 7:00pm$

AND $object_i.type =$ "$truck$"

Incremental Query $Q =< q_1, \tau_1, Q_1 >$;

in which,

$Q_1 =$ <"truck", LR/IR/CCD, Gate B, 7:00pm, TRUE>

$\tau_1(object_k, object_j) = object_k.time = object_j.time$

AND $object_k.position = object_j.position$

AND $object_k.type \neq object_j.type$

$q_1 =$ <"truck", DependencyGraph, Gate B, 7:00pm, TRUE>

APPENDIX B

SURVEY ON USER QUERY MANNER

Name: Date:

B.1 BACKGROUND SURVEY

Please rate your knowledge on the following topics from 1 to 5:

1 don't know
2 know a little
3 know some
4 experienced
5 proficient

Database Design	◯ 1 ◯ 2 ◯ 3 ◯ 4 ◯ 5
SQL Query Language	◯ 1 ◯ 2 ◯ 3 ◯ 4 ◯ 5
Entity Relation Diagram	◯ 1 ◯ 2 ◯ 3 ◯ 4 ◯ 5
Temporal/Spatial Query	◯ 1 ◯ 2 ◯ 3 ◯ 4 ◯ 5
Information Fusion	◯ 1 ◯ 2 ◯ 3 ◯ 4 ◯ 5

B.2 QUERY BUILDING EXPERIMENTS

B.2.1 Scenario

In our experiments, three kinds of sensors are used to keep track of the vehicles on the ground area (a parking lot with high security need), including laser radar (LR), infrared video (IR, similar to video but generated at 60 frames/sec) and CCD digital camera. Figure 33 shows the sample images taken by LR and CCD camera.

B.2.2 Information Retrieval

In our experiments, users information needs are interpreted as temporal/spatial queries. The following two methods are available to build this kind of queries:

1. Using a SQL-like temporal/spatial query language (e.g. ΣQL)

2. Incrementally construct query by elementary queries and query operators.

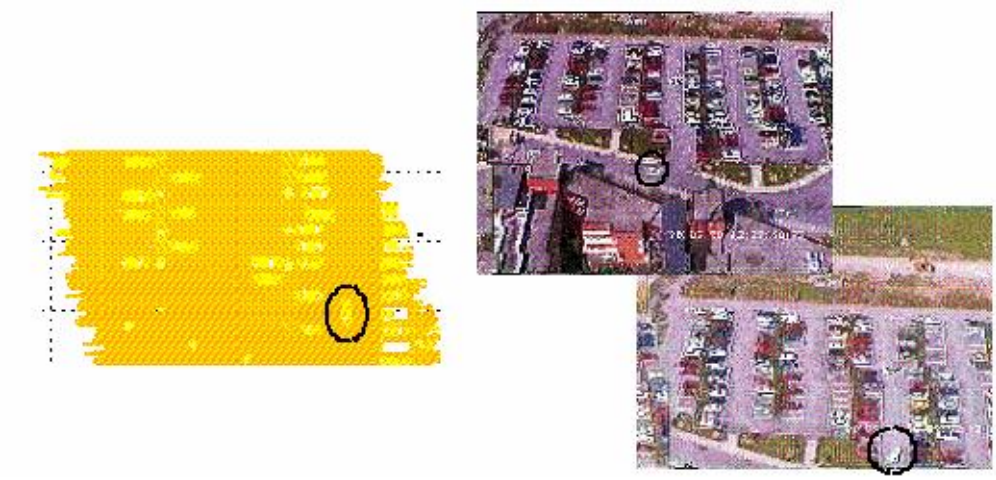

Figure 33: Sample Images in Our Experimental System

B.2.3 Query Building Experiments

For the following user study, you will be asked to specify the give query in natural language using the two methods mentioned above. Please mark your start time and finish time for EACH of them. Please indicate your preference.

Example B.2.1 *Query: Find a red car which passed toll gate A.*

Solution 1: SQL-Like Query

Start Time: 04:05:23; Finish Time: 04:25:45

```
SELECT object
CLUSTER object ALIAS OBJ1
FROM
    SELECT t
    CLUSTER *
    FROM LR, IR, CCD
WHERE OBJ1.type = 'car' AND OBJ1.color = 'red' AND
          Inside(OBJ1,'gate A')
```

Solution 2: Incremental Query

Start Time: 04:35:27; Finish Time: 04:40:45

```
Object1=<OBJ='car', SOURCE=LR/IR/CCD,AOI='gate A',
          TOI=all, PREDICATE = (object.color='red')>
```

Example B.2.2 *Query: Find a red car which passed toll gate A between 6:00pm–9:00pm.*

Solution 1: SQL-Like Query

Start Time: 04:05:23; Finish Time: 04:25:45

```
SELECT object
CLUSTER object ALIAS OBJ1
```

FROM

SELECT t

CLUSTER *

FROM LR, IR, CCD

WHERE OBJ1.type = ‘car’ AND OBJ1.color = ‘red’ AND

AND Inside(OBJ1,‘gate A’) AND OBJ1.t > 6:00pm

AND OBJ1.t < 9:00pm

Solution 2: Incremental Query

Start Time: 04:35:27; Finish Time: 04:40:45

Object1=<OBJ=‘car’, SOURCE=LR/IR/CCD,AOI=‘gate A’,

TOI=‘6:00pm–9:00pm’, PREDICATE = (object.color=‘red’)>

Example B.2.3 *Query: Find a truck which passed toll gate A between 6:00pm–9:00pm*

Solution 1: SQL-Like Query

Start Time: 04:05:23; Finish Time: 04:25:45

SELECT object

CLUSTER object ALIAS OBJ1

FROM

SELECT t

CLUSTER *

FROM LR, IR, CCD

WHERE OBJ1.type = ‘truck’ AND

AND Inside(OBJ1,‘gate A’) AND OBJ1.t > 6:00pm

AND OBJ1.t < 9:00pm

Solution 2: Incremental Query

Start Time: 04:35:27; Finish Time: 04:40:45

Object1=<OBJ='car', SOURCE=LR/IR/CCD,AOI='gate A',
TOI='6:00pm–9:00pm', PREDICATE = TRUE >

Example B.2.4 *Query: Find a red car followed by a truck within 10 minutes which both passed toll gate A between 6:00pm–9:00pm.*

Solution 1: SQL-Like Query

Start Time: 04:05:23; Finish Time: 04:25:45

```
SELECT object
CLUSTER * ALIAS OBJ1 OBJ2
FROM
    SELECT t
    CLUSTER *
    FROM LR, IR, CCD
WHERE OBJ1.type = 'car' AND OBJ1.color = 'red' AND
        OBJ2.type = 'truck' AND OBJ1.t < OBJ2.t AND
        OBJ1.t > OBJ2.t-10 AND Inside(OBJ1,'gate A') AND
        Inside(OBJ2,'gate A') AND OBJ1.t > 6:00pm
        AND OBJ1.t < 9:00pm AND OBJ2.t > 6:00pm
        AND OBJ2.t < 9:00pm
```

Solution 2: Incremental Query

Start Time: 04:35:27; Finish Time: 04:40:45

Object1=<OBJ='car', SOURCE=LR/IR/CCD,AOI='gate A',
TOI='6:00pm–9:00pm', PREDICATE = (object.color='red')>
Object2=<OBJ='truck', SOURCE=LR/IR/CCD, AOI='gate A',
TOI='6:00pm–9:00pm', PREDICATE = TRUE>
Query Operator τ={object1.t $\leq$ object2.t AND object1.t $\geq$ object2.t-10}

Please follow the example to construct the following queries:

Query 1 *Is there any simlar car beside a white car in the parking lot?.*

Solution 1: SQL-Like Query

`Start Time:___:___:___ Finish Time:___:___:___  (hh:mm:ss)`

Solution 2: Incremental Query

`Start Time:___:___:___ Finish Time:___:___:___  (hh:mm:ss)`

Which way do you prefer to specify this query?

◯ SQL-like Query; ◯ Incremental Query;

Query 2 *Find a blue truck which has been previously observed in the toll gate A.*

Solution 1: SQL-Like Query

`Start Time:___:___:___ Finish Time:___:___:___  (hh:mm:ss)`

Solution 2: Incremental Query

`Start Time:___:___:___ Finish Time:___:___:___  (hh:mm:ss)`

Which way do you prefer to specify this query?

◯ SQL-like Query; ◯ Incremental Query;

Query 3 *Is there any area (100 feet × 100 feet) in the parking lot where the number of cars is increasing between 8:00pm–9:00pm?*

Solution 1: SQL-Like Query

`Start Time:___:___:___ Finish Time:___:___:___  (hh:mm:ss)`

Solution 2: Incremental Query

`Start Time:___:___:___ Finish Time:___:___:___  (hh:mm:ss)`

Which way do you prefer to specify this query?

◯ SQL-like Query; ◯ Incremental Query;

B.3 FEEDBACKS

Do you have any query which you think is interesting in this scenario? Any comments about our research topics and experiments? Thank you!

APPENDIX C

VITA

Xin Li received his B.E. and M.E. degrees from Tsinghua University, China in 1999 and 2001 respectively. He became a PhD student in Department of Computer Science, University of Pittsburgh in 2001, and has been working with Prof. Shi-Kuo Chang since then. His research interests include information retrieval, visual language, and software engineering. He has been a summer research intern in ITRI, Taiwan (2004) and Google Inc., USA (2005).

APPENDIX D

PUBLICATIONS OF XIN LI

(10 papers in total, up to April 2006)

Journal Papers:

- [**J2**] **Xin Li**, Chieh-Chih Chang, and S. K. Chang. *Face Alive Icons*. Journal of Visual Language and Computing.
- [**J1**] S. K. Chang, Erland Jungert and **X. Li**. *A Progressive Query Language and an Interactive Reasoner for Information Fusion Support*. Journal of Information Fusion.

Conference Papers:

- [**C7**] S. K. Chang, **Xin Li**, Ricardo Villamarin, Dan Lyker, Chris Bryant. *The Design and Implementation of the Chronobot/Virtual Classroom (CVC) System*. The Twelfth International Conference on Distributed Multimedia Systems (DMS'06), Grand Canyon, USA.
- [**C6**] **Xin Li** and S. K. Chang. *User Profiling in the Chronobot/Virtual Classroom System*. The Eighteenth International Conference on Software Engineering and Knowledge Engineering (SEKE'06), San Francisco, USA.
- [**C5**] **Xin Li** and S. K. Chang. *A Personalized E-Learning System Based on User Profile Constructed Using Information Fusion*. The Eleventh International Conference on Distributed Multimedia Systems (DMS'05), Banff, Canada. Sep. 5 7, 2005, pp. 109-114.

- **[C4] Xin Li**, Chieh-Chih Chang, and S. K. Chang. *Face Alive Icons*. The Seventeenth International Conference on Software Engineering and Knowledge Engineering (SEKE'05), Taipei, Taiwan, July 14-16, 2005, pp. 29-36.
- **[C3] Xin Li** and S. K. Chang. *An Interactive Visual Query Interface on Spatial/temporal Data*. Proc. of Int'l Conf. on Distributed Multimedia Systems(DMS'04), San Francisco Bay, CA, Sep. 8-10, 2004, pp. 257-262.
- **[C2]** Giovanni Casella, S. K. Chang, Gennaro Costagliola, Erland Jungert, **Xin Li** and Tobias Horney. *An Architecture for Interactive Query Refinement in Sensor-based Information Fusion Systems*. Proc. of Int'l Conf. on Distributed Multimedia Systems(DMS'04), San Francisco Bay, CA, Sep. 8-10, 2004, pp. 315-321.
- **[C1]** S. K. Chang, W. Dai, S. Hughes, P. Lakkavaram and **Xin Li**. *Evolutionary Query Processing, Fusion and Visualization*. Proc. of Int'l Conf. on Distributed Multimedia Systems(DMS'02), San Francisco Bay, CA, Sep. 2002.

Workshop Papers:

- **[W1] Xin Li** and Weimin Yan. *An Internet Based Examination Platform (IBEP)—TH-IBEP*. Proceeding of the International Workshop on Distance Learning and Virtual Campus, Beijing, China, Nov. 2000.