A New Visual Query Language and Query Optimization for Mobile GIS

Haifa Elsidani Elariss

A thesis submitted in partial fulfillment of the requirements of Kingston University for the degree of Doctor of Philosophy

Faculty of Computing, Information Systems, and Mathematics
Kingston University

July 2008

Abstract

In recent years computer applications have been deployed to manage spatial data with Geographic Information Systems (GIS) to store and analyze data related to domains such as transportation and tourism. Recent developments have shown that there is an urgent need to develop systems for mobile devices and particularly for Location Based Services (LBS) such as proximity analysis that helps in finding the nearest neighbors, for example, restaurant, and the facilities that are located within a circle area around the user's location, known as a buffer area, for example, all restaurants within 100 meters. The mobile market potential is across geographical and cultural boundaries. Hence the visualization of queries becomes important especially that the existing visual query languages have a number of limitations. They are not tailored for mobile GIS and they do not support dynamic complex queries (DCQ) and visual query formulation. Thus, the first aim of this research is to develop a new visual query language (IVQL) for Mobile GIS that handles static and DCQ for proximity analysis. IVQL is designed and implemented using smiley icons that visualize operators, values, and objects. The evaluation results reveal that it has an expressive power, easy-to-use user interface, easy query building, and a high user satisfaction.

There is also a need that new optimization strategies consider the scale of mobile user queries. Existing query optimization strategies are based on the sharing and push-down paradigms and they do not cover multiple-DCQ (MDCQ) for proximity analysis. This leads to the second aim of this thesis which is to develop the query melting processor (QMP) that is responsible for processing MDCQs. QMP is based on the new Query Melting paradigm which consists of the sharing paradigm, query optimization, and is implemented by a new strategy "Melting Ruler". Moreover, with the increase in volume of cost sensitive mobile users, the need emerges to develop a time cost optimizer for processing MDCQs. Thus, the third aim of the thesis is to develop a new Decision Making Mechanism for time cost optimization (TCOP) and prove its cost effectiveness. TCOP is based on the new paradigm "Sharing Global Execution Plans by MDCQs with similar scenarios". The experimental evaluation results, using a case study based on the map of Paris, proved that significant saving in time can be achieved by employing the newly developed strategies.

Table of Contents

List of Figures

List of Tables

List of Abbreviations

[DTL 60 10G] Continuously, find the Time Left to reach the nearest gymnasium for the next 60 minutes. Keep on supplying me with an updated result every 10 minutes. The result includes the shortest path

[DTL 60 5 T] & [DNP 3 M] & [DW 200 M] Find the nearest theatre and its shortest path. While I am on my way, keep on continuously supplying me with the 3 nearest motels and their shortest paths that are within 200 meters (or ahead of me, e.g., in half a circle buffer) until I reach the theatre or 60 minutes overlap. Update my map every 5 minutes

[SN 1 M] Find the Nearest 1 Motel

[SN 1 R] Find the Nearest 1 Restaurant

[SN 1 R] Find the Nearest 1 Restaurant

[SN 4 H] Find the Nearest 4 Hotels

[SN 4 H] Find the Nearest 4 Hotels

[SN 6 B] Find the Nearest 6 Bus Stations

[SN 6 B] Find the Nearest 6 Bus Stations

[SNP 5 M] Find the 5 Nearest Motels with their Shortest Paths

[SW 100 U] Find Within 100m Universities

[SW 1000 R] Find all restaurants that are within 1000 meters from my location

[SW 300 U] & [SW 100 R] & [SN 5 H] & [SNP 4 S] & [SN 2 B] & [SDL 7 P] Find the Underground Stations that are within 300 meters from me, the restaurants that are within 100 meters, the 5 nearest hospitals, the 4 nearest schools with their paths, the 2 nearest Bus Stations, and the distance to reach the 7 nearest Police Office with their paths

[SW 50 H] Find Within 50m Hotels

[SW 500 R] Find all restaurants that are within 500 meters from my location

[SW 70 S] Find Within 70m Schools

B Bus Station

CJP Clock-triggered Join Policy

CQ Complex Query

CRM Customer Relationship Management

CSE Common Sub-expression Elimination

DCQ Dynamic Complex Queries

DDL Dynamically Find the distance left to reach the nearest facility

DN Dynamically Find the nearest facilities

DNP Dynamically Find the nearest facilities and their shortest path

DP Dynamically Find the shortest path

DSS Decision Support Systems

DTL Dynamically Find the time left to reach the nearest facility

DW Dynamically Find the facilities that are within a buffer

EDVC Exploratory Database View Constructor

G Gymnasium

GEP Global Execution Plan

GIS Geographic Information Systems

GPS Global Positioning Systems

H Hospital

HCI Human-Computer Interaction

HJP Hot Join Policy

IJP Incremental Join Policy

IVQL Iconic visual query language

*k*NN *k* Nearest Neighbor

LBS Location Based Services

LVIS Un Language Visuel for Information Systems

M Motel

MDCQ Multiple Dynamic Complex Query

MIS Management Information Systems

MMS Multi-Media Messaging System

MSE Minimum Spatial Element

MTE Minimum Temporal Element

P Police Office

QMP Query Melting Processor

R Restaurant

S School

SDL Find the distance left to reach the nearest facility

SMS Short Messaging System

SN Find the nearest facilities

SNP Find the nearest facilities and their shortest path

SP Find the shortest path

STL Find the time left to reach the nearest facility

SW Find the facilities that are within a buffer

T Theatre

TCOP Time Cost Optimization

U Underground Tube / Metro

Acknowledgements

I would like to extend my gratitude to my director of studies, Dr. Souheil Khaddaj, for his continuous support and devoted cooperation throughout the course of this work. His brilliant ideas and objective criticism significantly contributed to this work.

I would like to extend my deep thanks and appreciation to my family who granted me extreme support and exceptional patience especially during the most difficult moments. Their continuous reinforcement and encouragement allowed this work to be completed.

Chapter 1 - Introduction

1.1 Geographic Information Systems (GIS)

Geographic Information Systems (GIS) are computer-based tools mostly used to handle the geo-features of the real world. They provide the capability to input, store, manipulate, analyze, retrieve, transform, and display geographical data related to earth surface and its events as well as to measure aspects of geographic phenomena and processes. They play a major role in many domains such as urban and environmental planning, transportation, utility mapping and tourism, and can be applied in different fields such as Bio-informatics, Air traffic, Location Based Services (LBS), Management Information Systems (MIS), Customer Relationship Management (CRM), Decision Support Systems (DSS) and many others [Cle93, Nur06, Sch06]. There are three types of GIS, the spatial which is mainly concerned with the properties of spatially referenced data such as the location, shape, and size of maps and geographical areas, the temporal which handles the changes of geographical objects over time such as cadastral data (owner, lot number, tax value), and the spatio-temporal which manages the attributes and behavior of geographical objects over time.

With the emergence of mobile technologies, the use of GIS Geo-data by mobile devices becomes very common. Mobile GIS is the combination of systems which include mobile devices, Global Positioning Systems (GPS), wireless communication, and GIS software. Their applications can be used in several fields for several purposes. In Field Worker Services, they are used by fire fighters, emergency workers, inspectors, maintenance teams, and utility crews [Fid07, Nus04]. In tracking systems, they provide the facility to track employees, children in the backyards, and many other types of mobile users [Lad05, Kim07]. In road networks, they provide the ability to organize public transportation [Rep06], query moving objects [Gut06], guide tourists electronically [Bee07], and exploit photos by pedestrians [Bee06]. In data mining, they play a major role in trajectory pattern recognition of moving users [Gia07, And07]. In Location Based Services, they are applied as functional-reminder

applications [Lud06], detector of identifiable areas [Rot06], locator of specific web content [Tez06], and locator of users in Location based games [Gus06, Ras06].

Mobile GIS are typically used in tourist and navigation systems for Proximity Analysis which includes querying the *k* Nearest Neighbor (*k*NN), for example, restaurant, and finding the facilities that are located within a circle area around the user's location, known as a buffer area, for example, all restaurants within 100 meters. The existing Mobile GIS applications have textual or menu-driven environments. In a textual environment, the user formulates his query through typing a text or writing an SQL query. In the menu-driven environment, the user reads and selects items from text menus. Both approaches do not provide a user friendly or easy-to-use environment and in some applications they are aimed at expert users only. Thus, there is an urgent need to develop a visual query language that provides the mobile user with the facility to formulate a visual query using expressive icons. Moreover, the existing mobile GIS applications allow the mobile user to send only one query, called simple query, at a time to the GIS server. But, in many situations the user needs to formulate many simple queries in order to achieve the result that he needs; this might take the form of many maps each of which corresponds to a simple query. Hence, the challenge arises to allow him to send multiple queries, called complex queries, together at the same time in order to produce a single over-layered map that combines the results of all the queries, therefore reducing the overheads significantly.

GIS spatio-temporal queries, both simple and complex, can be classified as either static or dynamic. Static queries concern situations where a user enquires about objects at a single time snapshot which is the current time, for example, find the nearest hotel now. Dynamic queries, also referred to as Continuous queries, concern situations where a user enquires about objects at every time interval during a certain period of time, for example, a user trying to find the nearest hotel every five minutes during the next hour as he/she is moving away from the airport in a taxi. The existing visual query languages are oriented mainly toward desktops and not mobile devices and handle static queries but not dynamic ones. Therefore, addressing these issues requires a new visual language to deal with dynamic complex queries for Mobile GIS which raises a number of challenges particularly with respect to query optimization and processing.

Query Optimization, also known as Query Processing[1], includes a list of tasks that are executed in a particular order for processing a simple query which form an execution plan. The process starts by decomposing the query into many query fragments, then eliminating common sub-expressions (CSE), reordering the tasks, and finally executing the plan. Having a complex query made up of a number of simple queries requires the Query Processor to execute a number of corresponding plans, combine their result in one map, and send the result map to the mobile user. If one or more of the simple queries are dynamic, the processor repeats the same steps for every new time instance that corresponds to the current user location. These repetitions raise the need to develop a Query Processor for Dynamic Complex Queries which aims to eliminate all repetitions, share intermediate results, facilities, spatial areas, time intervals, and generate one Global Execution Plan for each dynamic complex query.

Moreover, when examining dynamic complex queries of multiple users, it is realized that similar complex queries might be formulated by multiple users making the query processor repeat the same tasks again. In order to optimize the processor execution time, the need arises to integrate a new decision making mechanism in the processor that allows sharing previously generated Global Execution Plans.

1.2 Visualization of Query Languages

There are four types of Query Languages for GIS. The first type is the Text query language where the user, typically a domain expert, has to formulate Structured Query Language SQL-like queries. But SQL query languages possess complex textual syntax, textual input and output, and the user might need to know the structure of the database schema before writing a query [Djo96]. Since many GIS users are expected to be non experts in the geographic domain, English language type queries were designed to facilitate queries formulations [Kim05, Yue05].

The second type is the Spatial-Query-by-sketch language [Bla00, Ege97] where the user formulates a query by drawing spatial configurations typically on a touch sensitive screen. In order to enable exact and similar matches to be browsed through,

[1] The term Query Optimization has been internationally defined as Query Processing aiming at producing faster results, as elaborated in [And06, ELM06, Kan94, and Mok05a].

the user can reduce the similarity ranking in order to lower the accuracy threshold for the result.

The third type is Menu-Driven where the user selects one item out of a list of choices that are displayed in a list box and accordingly the query is formulated. A particular case is the Query-by-object language (QBO) [Abd05a] that provides multiple user steps based on menu-driven user interface for banking database queries. The menu-driven query formulation is applied in M-commerce [Abd05b], spatial queries for Relational Data Base Management Systems (RDBMS) and Web GIS [Abd05c], and spatial queries for RDBMS and mobile GIS [Abd05d, Abd05a, and Abd06].

The fourth type is the Visual Query Language (VQL) where the user formulates a query by selecting icons or metaphors. An icon is a pictorial representation of a function or an object. In some visual query languages the user drags and drops the selected icons to a special working area in order to validate the query before its execution [Bon02, Pao04]. In all existing visual query languages, the user can formulate static queries but not dynamic ones [And00, And02, And03a, And03b, And04, And07, And99, Mur00, Mur98a, Mur98b, Mur98c, Smi04, and Smi05].

Different query languages have been used for mobile GIS applications and broadly fall into two of the above categories namely textual and menu-driven environment. However, they do not provide visual query formulation nor do they support dynamic complex queries. Hence, the need arises to develop a visual query language for Mobile GIS that provides the mobile user with the facility to formulate dynamic complex queries. Moreover, there is also a need to deal with the globalization of visual query languages in order to meet global and diversified demands. Globalization refers to the production of products that are used worldwide. Thus, the need to develop international user interfaces becomes a major issue in visual query languages. Marcus, in his paper [Mar99], defines globalization in user-interface design and demonstrates why globalization is vital to the success of computer-based communication products. Hence, in this work the use of smiley icons is proposed in order to build an iconic visual query language.

A smiley is a graphical representation that displays a smiley face to express emotions and convey facial expressions such as, happy, sad or bored. Smiley icons can be easily understood by all peoples in all countries worldwide because they are universal signs that can be well transferred from one culture to another with a high level of expressiveness. Pang, in his thesis [Pan02] uses them to express the operator's feelings in a nonverbal interface for the McDrive system developed for the McDonald chain. Since the elements of a GIS are mostly about functions and objects concerned with real-life actions and facilities, each one of them may be expressed by a smiley icon, and when combined together with operator icons, they can form a visual query language.

1.3 Query Optimization Strategies

The proximity analysis queries are considered as spatio-temporal queries since they involve both space and time and can be categorized into several types. The first type is the static buffer operator queries which are used when a static user enquires about static objects. The second is the static buffer operator queries which are used when a static user enquires about moving objects. The third is the dynamic buffer operator queries which are used when a moving user enquires about static objects. The forth is the dynamic buffer operator queries which are used when a moving user enquires about moving objects. The nearest neighbor operator *k*NN is used similarly to the buffer operator thus the four types of queries consist of the static nearest operator on static objects, static nearest operator on moving objects, dynamic nearest operator on static objects, and dynamic nearest operator on moving objects.

To deal with the different types of queries, different query optimization strategies have been considered in literature. For example, in [Laz02, Son01] the results of previous queries are cached and used to prune the space of subsequent queries. Another example used by [Tao02] is to pre-compute the result of a query that has a fixed trajectory using computational geometry for stationary objects. The last example used in [Mok04b, Xio05] is to incrementally evaluate a query in other words the updates of the queries are evaluated only.

The optimizations of multi-user queries have been focused on a single operator [Li05, Liu03, Mok04b, Mok05b, Mou05, Pap03, Pap04, Pap05, Xio05, Yiu05, and Yu05] but the multi-user complex queries still need to be addressed. Elmongui in [Elm06] raised the challenges in the processing of spatio-temporal queries and showed how important it is to address them. He proposed an optimization strategy for query processing based on the selectivity of each operator and the number of objects that are located a certain area, then, accordingly the order of operators to be executed is swapped. But, the expected increase in the number of users in large-scale mobile systems requires the introduction of a new Decision Making Mechanism that is based on Sharing the GEP (Global Execution Plan) across users since a high rate of similarity between the queries scenarios is expected.

1.4 The Structure of the Thesis

Chapter 2 conducts a thorough investigation into existing visual query languages in order to examine their user interface and query building process. The investigation includes the specification of which aspects of VQL are evaluated, the determination of evaluation method and the discovery of the type of queries that are handled. The chapter concludes with a summary identifying the advantages as well as limitations of the reviewed VQL and it identifies the need for a new visual query language for mobile GIS.

Chapter 3 reviews query optimization strategies in GIS in order to determine their various approaches and examine how they are applied in various domains. It investigates the propagation ruler mechanism and the existing time cost optimization strategies in order to specify how they are implemented with different type of queries. The chapter concludes with a summary identifying the advantages and limitations of the reviewed query optimization strategies and identifies the need for a new query processing paradigm as well as a new decision making mechanism for time cost optimization.

Chapter 4 proposes a new visual query language IVQL that addresses the challenges that are discussed in chapter 2 and describes its design and implementation. It

elaborates the constructs of the language, describes its architecture and explains the formulation of complex queries. It also describes the process of evaluation and presents the evaluation results and analysis of the usability testing as well as the user testing. The chapter summarizes the results that are reported and analyzed as related to the user interface, query building process, and expressive power of the language.

Chapter 5 introduces the Query Melting paradigm which is based on the sharing paradigm, query optimization, and cost optimization. It presents the commonalities that exist between various execution plans of queries, introduces the query melting paradigm, describes its components, introduces the new decision making mechanism which is based on sharing Global Execution Plan approaches. It also explains the templates of the operators and describes the mechanism of the Melting Ruler showing how it results in Query Melting.

Chapter 6 describes the design and implementation of the proposed Query Melting Processor. The architecture of the processor is presented using a variety of diagrams such as the Use case Diagram which is used to visualize the activities and tasks of a system showing at the same time the actor/actors associated with each activity and the Time Sequence diagram that shows the life time of each activity in the system. The implementation of the processor is elaborated by showing and explaining the algorithms that are related to each phase of the processor and evaluates theoretically the cost of Melting. The chapter concludes with a summary of the computational cost of each of the algorithms.

Chapter 7 presents the integration of the proposed IVQL together with the new optimization strategies namely, Query Melting paradigm, Query Optimization, and cost optimization by means of sharing the global execution plans. It starts with a description of the components of the system. It presents an explicit scenario for three users only in order to show how the system works and conducts an experimental study with the aim to evaluate the system using a case study based on a Tourist Mobile GIS application using the map of Paris, and it presents the results and analysis of the evaluation. The chapter concludes with a summary highlighting the results of the experimental study and showing the significant cost effectiveness of the new strategies.

Chapter 8 concludes the thesis by specifying the main contributions of the work in both the visual query languages as well as query and cost optimization research areas. It gives a summary of the work done as well as its evaluation results. It also outlines possible research areas that can be carried out in future work with respect to the type of queries, operators, and optimization strategies.

Chapter 2 - Visual Query Languages for GIS

2.1 Introduction

The emergence of visual query languages plays an important role in human-computer interaction (HCI) that is concerned with the design, evaluation and implementation of interactive computing systems for human use. HCI has moved beyond designing user interfaces to support all human activities and facilitate the user experience through designing systems that would make the user's work easy and efficient, by improving the learning process, providing enjoyable and exciting entertainment, enhancing communication and understanding, and supporting new forms of creativity and expression.

A major aspect of HCI is usability which is an essential key quality factor in software engineering. Khaddaj in [Kha04] defines it as the ability of a product to be used for the purpose chosen. In other words, usability reflects how much the system under development conforms to the objectives that were set for it. According to him, one of the most important features of a useful system is having an easy-to-use user interface. Usability Testing forms the basis for evaluating systems and their user interfaces. Pressman in his book [Pre05] defines it as a measure of how well a computer system facilitates learning, enables users to be efficient and makes them satisfied with the system. According to him, the aspects of the system that are evaluated in order to measure its usability are how easy it is to learn, effective to use, and easy to remember how to use.

The usability aspects that are usually evaluated in any regular query language are the query writing and the ease of use, whereas, the usability aspects of visual query languages depend on the user interface, query building process, expressive power of the visual language, the proper representation of the icons used, the icons recognition, interpretation, comprehension and memorization, and the ease of use. Thus, in order to evaluate those aspects quantitatively, user testing should be conducted where the

user's performance is tested, whereas to evaluate them qualitatively, user satisfaction should be tested.

In order to evaluate the usability aspects of existing VQL a thorough investigation is conducted in this chapter in order to summarize their advantages and limitations, identify the aspects that are used in the usability testing, and determine the method used for their evaluation: user testing or user satisfaction. The existing visual query languages can be categorized as non-spatial such as Kaleidoquery [Mur00, Mur98a, Mur98b, Mur98c] and Link Analysis [Smi05, Smi04], both spatial and non-spatial such as Filter-Flow [Mor04, Mor02] and GeoQA [Sto06, Sto03a, Sto03b, Sto00, Sto98], and spatio-temporal such as LVIS [Bon02, Bon01, Bon00, Bon99], PHENOMENA [Lau03, Lau07, Pao07, Pao04, Pao03], and VISUAL TOOLS [And04, And03a, And03b, And02, And00].

The rest of the chapter explores each category of visual query languages. Section 2.2 is concerned with the non-spatial visual query languages which are related to object databases but do not handle spatial queries such as Kaleidoquery for Querying Object Databases and EDVC Interface for Link Analysis. Section 2.3 is concerned with both spatial and non-spatial visual query languages which provide the capability to query spatial and non-spatial databases such as Filter-Flow Query for Large Databases and GeoQA for Object-relational GIS. Section 2.4 is concerned with spatio-temporal visual query languages which provide the capability to query spatial and temporal databases such as LVIS for Querying Spatio-temporal Databases, PHENOMENA for querying Continuous Fields, and Interactive Visual Tools to explore Spatio-temporal Variations. For each of the listed visual query languages, the icons that are used are described with their meaning, the user interface is presented, and the query formulation process is explained. An investigation is done to determine which aspects are used in the evaluation of the visual language such as expressive power of the icons, the ease-of-use of the user interface, the ease of formulating a visual query, which method of evaluation is conducted, and the evaluation results. Finally, section 2.5 concludes with a summary of the advantages and limitations of the reviewed visual query languages.

2.2 Visual Query Languages for Non-spatial Databases

The existing visual query languages that deal with non-spatial databases are concerned with retrieving objects that satisfy certain conditions and constraints based on their attribute values. The objects could be a person, company, etc. and the constraints could be older than 20 years, greater than 100, etc. An example of these visual query languages is the Kaleidoquery [Mur00] which uses 3D graphical interface to represent and depict a visual query in the form of a filter-flow. Another example is the EDVC [Smi04] which uses objects and links to represent and depict a visual query in the form of a network.

2.2.1 Querying Object Databases

The first example of the visual query languages that deal with non-spatial databases is the Kaleidoquery [Mur98a, Mur00] which was designed for object databases that depicts the query as a filter flow. The query language is graph based and represents the database schema as a starting point for querying the database. The user constructs a query by selecting parts of the schema and combining them with query operators and/or constraints that are placed on attributes of the schema parts therefore the user is actually performing a query starting with a mass of information. Query operators and constraints are used as filters letting through only the information that the user is interested in. The output of the query can flow into other queries for further refinement. This flow and the refinement of information are also depicted while the user is building the query.

The filter flow model takes the input composed of class instances and filters them by the constraints that are placed on the attributes of the classes in order to produce the output. The queries produced by the user are translated into the Object Database Management Group (ODMG) standard Object Query Language (OQL) that can be utilized in any ODMG compliant database as described in [Mur98b, Mur98c]. The class instance (extent) name and the icon associated with the class of the extent are used to visualize the extents and the classes, as shown in Figure 2.1.

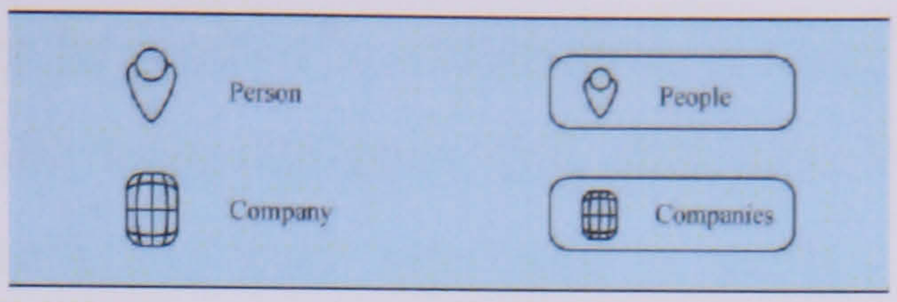


Figure 2.1: Example Schema.

Simple queries can use any of the operators =, <, <=, >, >= and *like*. The simple query shown in Figure 2.2 represents a constraint that results in producing an output that includes only persons who are aged less than 20. The OQL for this query is:

```
select p
from p in People
where p.age < 20
```

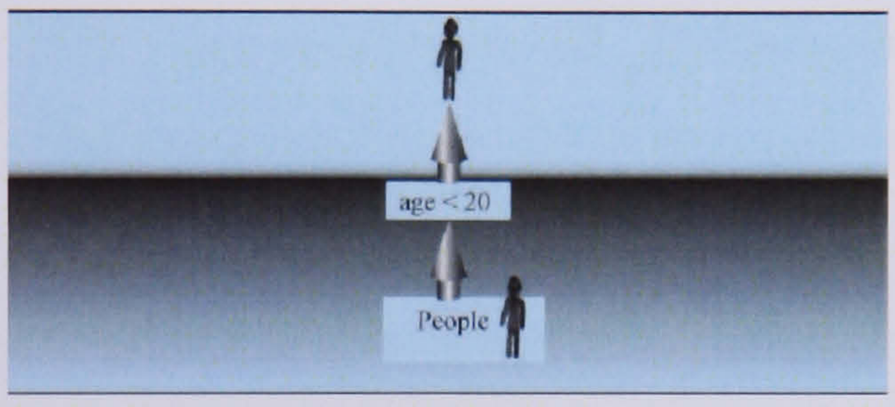


Figure 2.2: A Simple Query showing Attribute Selection.

To visualize the 'and' and 'or' relations of a query, the filter flow model described by Shneiderman [Shn91] was chosen. Figure 2.3 shows how the 'or' is visualized by the extent instances flowing into each part of the 'or' query. It also shows how the 'and' is visualized by the extent flowing through the query and are filtered through each constraint in turn.

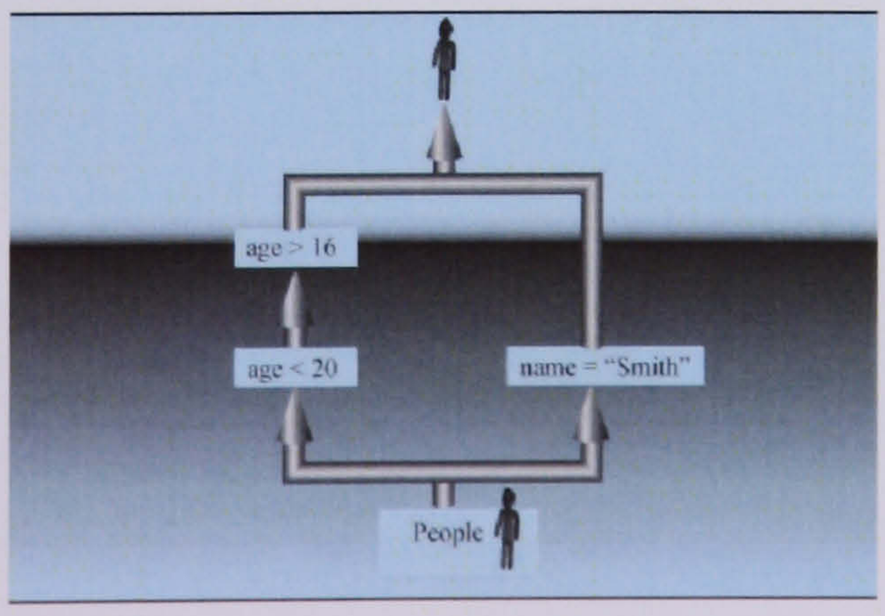


Figure 2.3: Query showing the Use of 'and' and 'or' operators.

Arithmetic operators can be applied to attributes in a class and complex arithmetic expressions could be built using attributes from different classes. The user can also visualize a query that navigates from one class to another related class and apply constraints to the related class's attributes. The queries can use the 'for all' and 'exists' operators of OQL as well as aggregates such as 'count' and 'max' functions. The user builds the query first then selects how the results are to be structured before viewing the output. The results can be sorted using the operator order by that is visualized by an upward pointing arrow '↑' for ascending and a downward pointing arrow '↓' for descending. In order to partition the results of a query, the 'group by' operation can be applied to separate the results into distinct groups depending on the conditions that they satisfy.

The Kaleidoscope, the 3D implementation of Kaleidoquery, was designed and implemented to examine the impact of utilizing different forms of hardware ranging from the standard monitors and desktop mice to head mounted displays, 3D mice and auto-stereoscopic displays. The database schema is displayed as 3D icons composed of the class name and its visualization. A virtual hand allows the user to navigate through the environment and select a class extent or an attribute to apply a constraint, as shown in Figures 2.4 and 2.5.

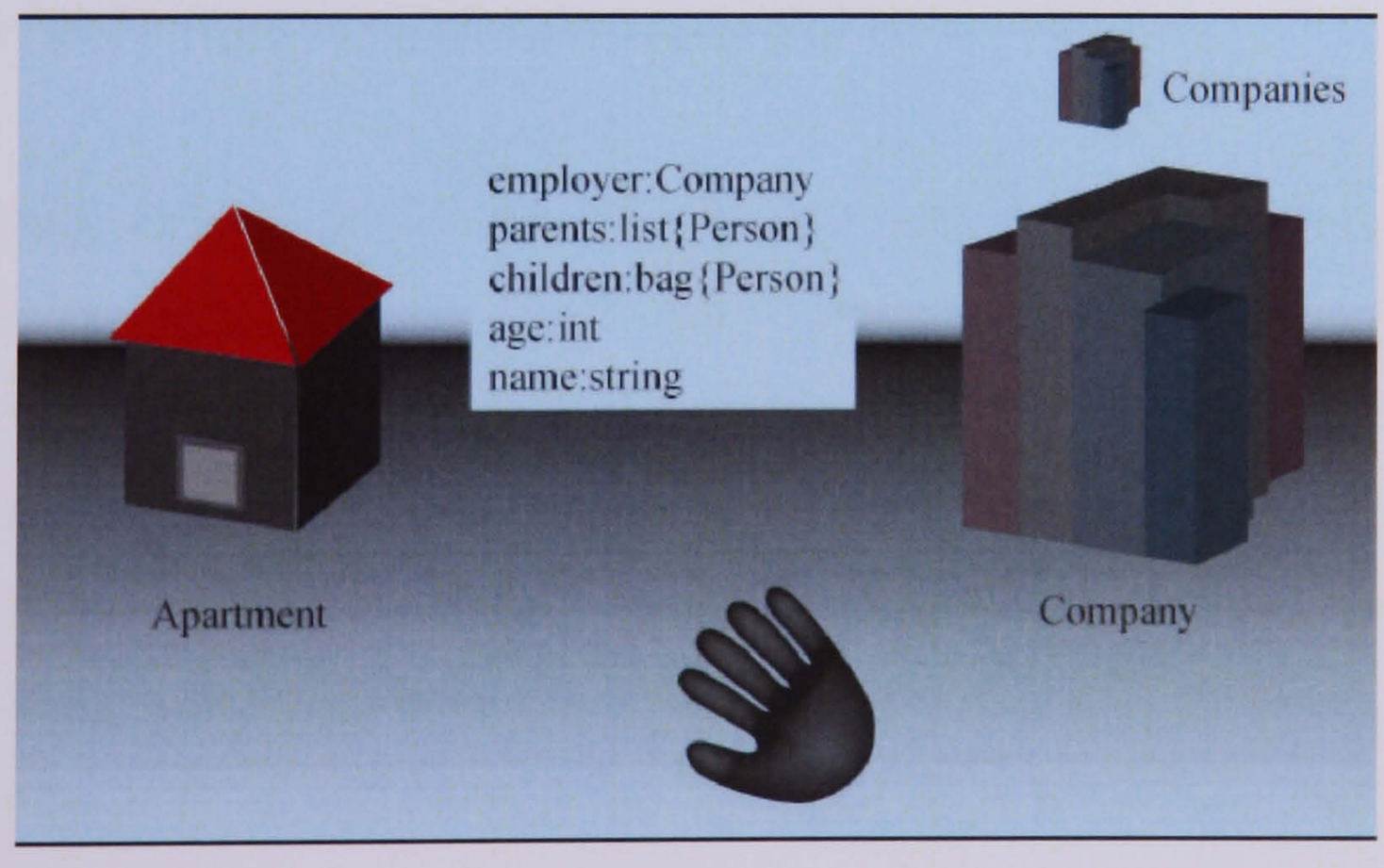


Figure 2.4: A Person Attributes and the Virtual Hand.

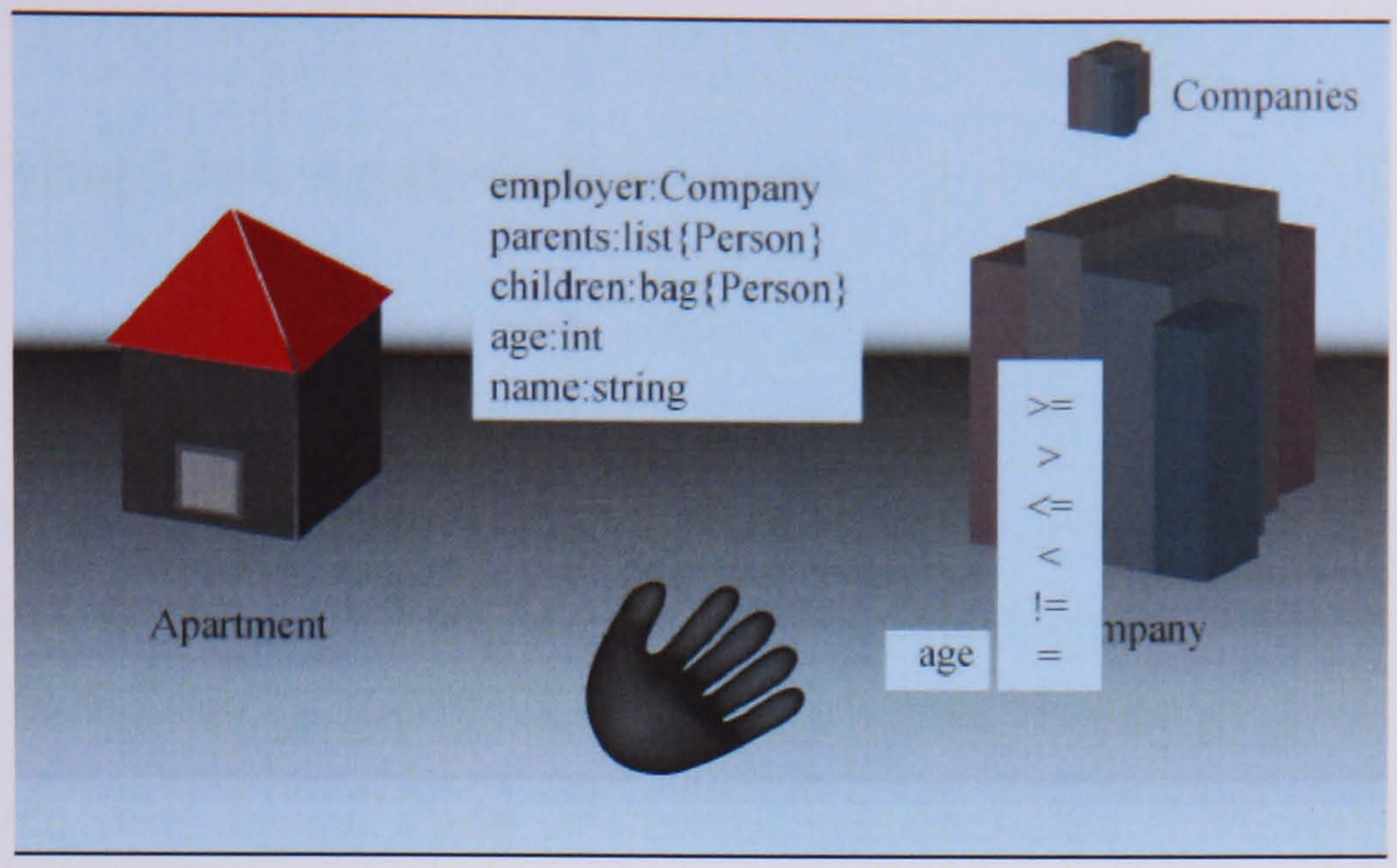


Figure 2.5: Age Operations.

To evaluate Kaleidoscope [Mur98b], OQL was used to compare the 'select' statements with the visualized interface for building queries in Kaleidoquery. Two subject groups were selected from students who hold university degrees and were classified as programmers and non-programmers. Programmers correctly defined significantly more relationship queries in Kaleidoquery. They showed higher preference for attribute comparison in OQL and for data flows over 'and' and 'or' in OQL. Non-programmers answered significantly more correct queries in Kaleidoquery than OQL. They defined more correct queries with results selection and showed a higher performance for the structuring aspects of Kaleidoquery. In general, programmers performed better than non-programmers.

As a summary, Kaleidoquery, the visual query language that was developed in [Mur00], has a powerful visual query builder but it is mainly aimed at object oriented databases. However, some of the ideas could be modified to be used on relational models; also perhaps additional visual operators to deal particularly with spatio-temporal queries can be added as to extend the Kaleidoquery in the object oriented mode. Therefore, any new extension should provide a visual depiction of the spatio-temporal schema and its attributes using the same 3D environment that was used in Kaleidoquery as it has proven to be easy to understand, more user-friendly, and more efficient.

2.2.2 Querying Link Analysis Databases

The second example of the visual query languages that deal with non-spatial databases is the Exploratory Database View Constructor (EDVC) [Smi04, Smi05] which is a visual query interface especially designed to support Link Analysis that is used to understand data collected in criminal or terrorist threat analysis and helps in guiding the future course of an investigation. EDVC is based on the method of visualizing the data in the form of a network that shows the connections and relationships between the objects of interest such as people, locations and the information about them, thus enabling the user to establish if objects are connected. It uses rectangular icons to visualize objects such as people and locations, directional lines to represent the connections and links between the objects, and icons and lines to construct a query which is then either executed or saved for further refinement where objects and links can be altered to supply more meaningful information.

When a visual query is ready for execution, it is converted from the User Interface Classes to the Query Operation Classes that is independent from any data source. The current implementation of EDVC is for the DBMS Sentences [Laz04]. The results of the queries are produced in the form of a link chart to increase the comprehensibility of the user. It produces also text notes that display information that are not stored in the database but might be of significance in future investigations. They suggest to the user some key points about the link chart results, such as 'The evidence suggests that Henry and Jack is in fact the same person'. The investigator can visually detect if people or objects are connected when investigating a particular serious crime, atrocities or terrorist work. The objects could represent people, companies, hotels, restaurants, banks, etc. and the lines relationships such as friend, sister, married to, works at, frequents, invests in, manages, advises, employs, owned by, date of birth, calls, owns, etc. Thus, the network of connections can be visualized by using a Link Chart that is made up of icons representing nodes and arrows representing edges as shown in Figure 2.6.

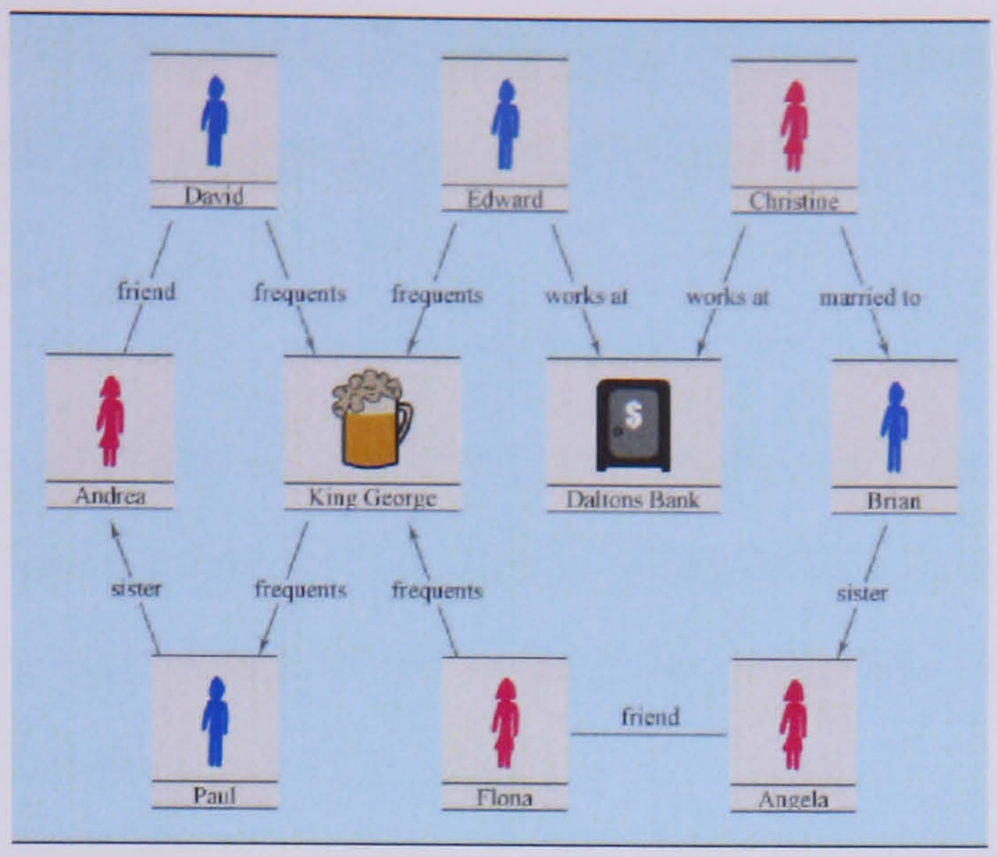


Figure 2.6: The Connections between a Set of People displayed as a Network.

To represent a network of connections within a database, the Binary-relational models [Abr74, Mcg80] are used based on the concepts of entity type and relationship type. Queries are constructed visually through building link charts, by adding to the working area the objects to be queried along with the required links between them. Objects displayed with a question mark, as shown in Figure 2.7, are used to inform the user that these objects have one or more links in the database but these links are not currently displayed in the chart, thus suggesting that further exploration from this object might be useful. The user could carry out the exploration in a step-by-step manner or by using the special facilities that the explorer provides.

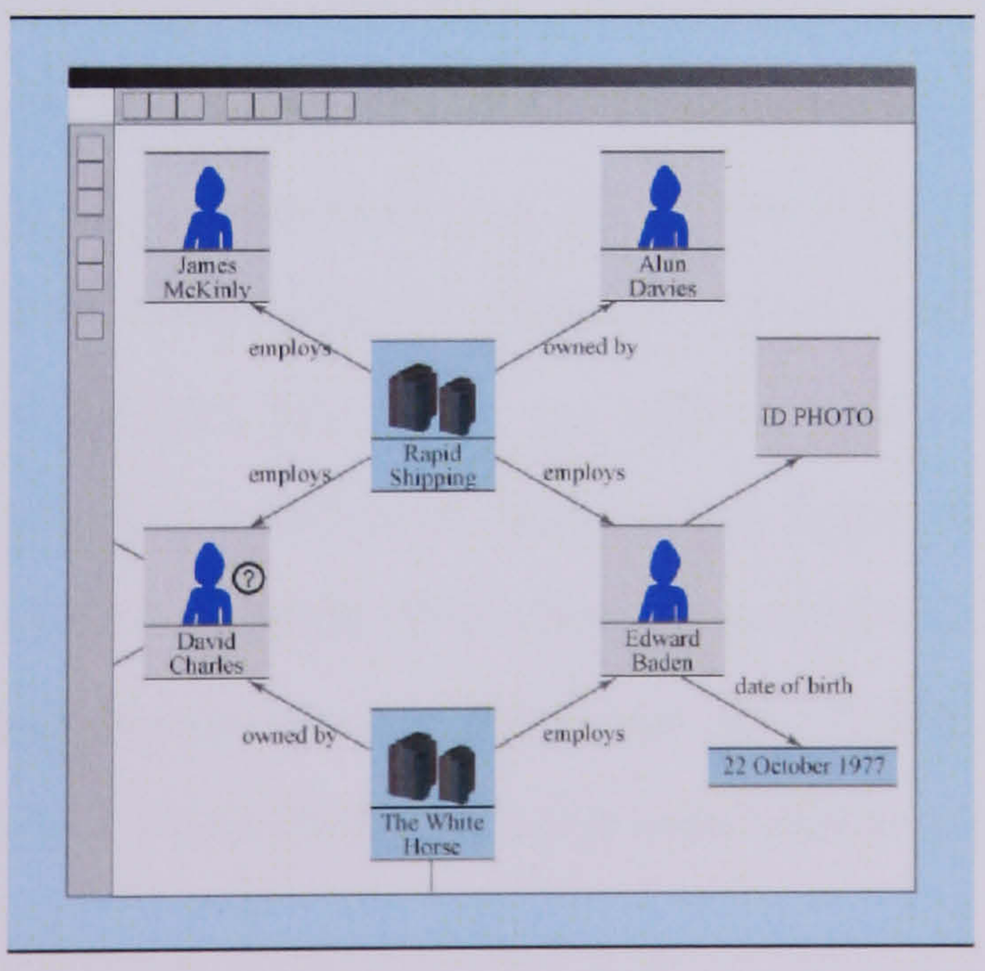


Figure 2.7: An Explorer Interface Chart in Course of Development.

The explorer provides six facilities, dealing with objects which can be added to the chart, that the user can use to carry out an exploration of the data stored as listed below:

1. Display information stored about an object: the explorer displays in a separate window the values and objects linked to a specific object.
2. Display objects connected to an object: the explorer displays all the objects that are connected to a specific object.
3. Display paths connecting two objects: the explorer displays all the paths connecting two specific objects.
4. Display objects connected to a group of objects: the explorer displays all the objects connected to two or more specific objects.
5. Display objects similar to an object: the explorer displays all the objects that are similar to a specific object according to a given set of similarity conditions.
6. Display the common properties of two selected objects: the explorer displays all the objects and values linked to two selected objects based on the fact that their links have the same type and direction of link.

The explorer provides also a programming level interface that allows the user to build special query operations and displays their results as a link chart. The user may remove elements from the chart or reposition its elements for better clarity which can be improved by rearranging elements in a circular or hierarchical fashion. The queries are referred to as filter patterns which are mainly divided into two categories. The first category consists of queries that filter the database objects and links and the results of which are made up of objects and links are added to the chart displayed in the explorer. The second category consists of queries that filter the objects and links that are displayed in a specified link chart. This category is used to construct a sequence of queries when further modifications and operations are needed to increase refinement and comprehensibility. The Filter Pattern Constraints consist of five types.

1. Object Constraints: an object constraint matches an object or all the objects that are instances of an object type.
2. Link Constraints: a link constraint matches two objects that are linked by instances of a given link type. Figure 2.8 shows a link constraint that matches

instances of a specified link type to display all people who telephoned David Jones.

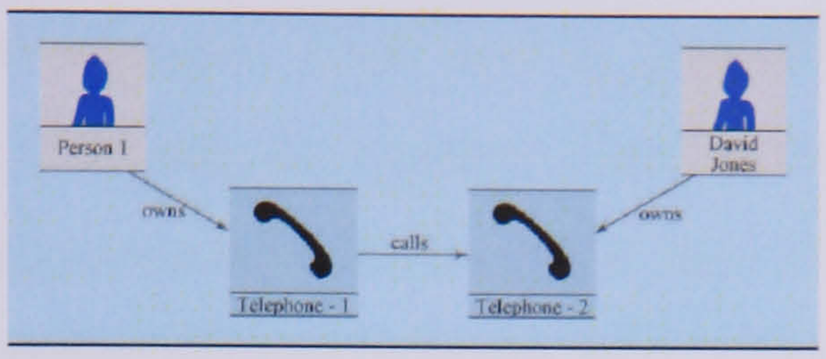


Figure 2.8: A Filter Pattern to display all People who telephoned David Jones.

3. Path Constraints: a path constraint matches connections between objects. It produces all the paths connecting two objects. The produced paths may include the shortest path.
4. Object Comparison Constraints: object comparison constraints connect two object constraints. It specifies whether the objects matched by the constraints should be identical or different as shown in Figure 2.9.

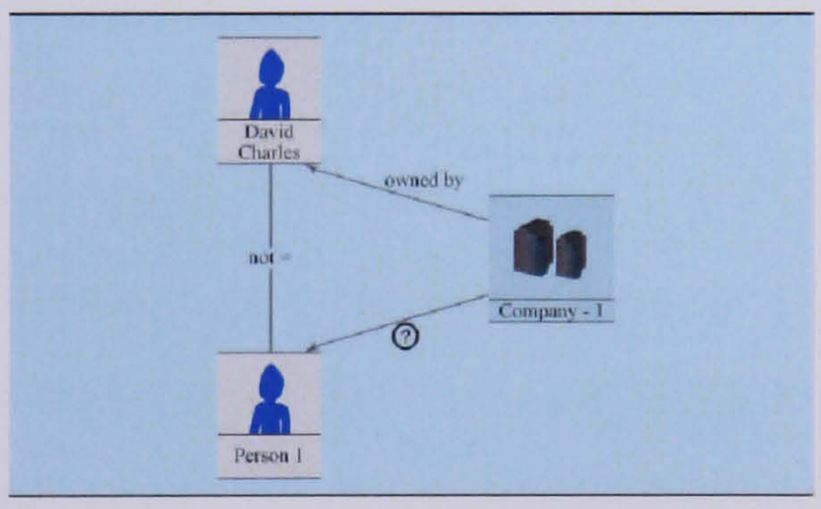


Figure 2.9: A Filter Pattern matching People involved in a Company owned by David Charles.

5. Excluded Constraints: a constraint may be excluded from the chart. When it is excluded, all the objects, links or paths that were matched by the constraint will be automatically removed from the chart.

EDVC not only focuses upon visualizing the query specifications but also on visualization of the database query results. It shows the results of a query in the form of a link chart which may also be modified to play the role of a new query. Elements from the link chart may be removed or modified to increase comprehensibility and to get a better clarity of information. EDVC is implemented using two groups of classes. The first group is used to provide the User Interface and the second one to perform the

Query Operations. The data exchanged between the two groups is independent from the data in the database. EDVC is developed using the DBMS Sentences where each database is referred to as a profile that consists of many chapters to store metadata and information about profiles changes. Each user is provided with a separate profile hence, if many users are accessing the system, none of them can actually view nor affect the other users changes made to the common data. Thus, EDVC provides each investigator with the possibility of independently testing hypotheses.

The evaluation that was conducted in [Smi05] aimed at evaluating the ease of use of EDVC visual user interface. Twenty subjects were chosen at random from different educational background, ranging from early school leavers to postgraduates students. All the subjects had experience of interacting with graphical user interfaces but had no previous knowledge on using a database query language, a programming language, or Link Analysis. Each subject filled in his personal information in a pre-evaluation questionnaire. The query system was explained to each subject in two separate tutorial sessions, one for the introduction of the explorer interface and the other for the introduction of the filter pattern editor. Three aspects of usability were tested: query writing, query reading and query comprehension, with the aim to measure how proficient a subject is at forming queries. The results of the user evaluation indicate that the visual interface provided by EDVC is well suited to supporting the users with little or no experience with a conventional DBMS interface, the subjects were able to form a useful class of queries and this was facilitated by the intuitive style of the interaction provided in the explorer, the user interface for constructing path filters needs to be improved, and a greater level of training needs to be provided.

2.3 Visual Query Languages for both Spatial and Non-spatial Databases

The existing visual query languages that deal with both spatial and non-spatial databases are concerned with retrieving objects that satisfy spatial conditions about points, lines, and regions and have attribute values that satisfy conditions and constraints. The spatial objects could be a building, road, or city and the non-spatial could be any attribute with a certain value. An example of these visual query languages is the Filter-Flow [Mor04] which uses a filter-flow to depict a query about

both spatial and non-spatial objects. Another example is the GeoQA [Sto00] which uses an interactive map and a query composer to formulate joins about both spatial and non-spatial data.

2.3.1 Querying Spatial and Non-spatial Objects and Relations

The first example of the visual query languages that deal with both spatial and non-spatial databases is the Filter-flow [Mor04, Mor02] which has an interface for large spatial databases. The various types of queries are expressed by Basic Filters that are formed of icons each of which is used for a different type of spatial relations. The different types of spatial and non-spatial constraints are expressed using a diagrammatic technique utilizing a data flow metaphor, based on the filter flow metaphor proposed by Young and Shneiderman [Shn93], that uses the metaphor of the water flowing through a series of pipes and uses the layout of the pipes to indicate the binary logic operators 'and' and 'or'.

The different relations that exist between spatial constraints are presented in a hierarchical fashion. Filters are used to construct query diagrams between data input and data output elements and the queries are visualized by a flow of information that may be filtered or refined. The visual queries produced by the user are parsed and translated to the extended SQL query language. Rectangular boxes are used to depict and visualize the object classes. Each rectangle contains the name of the class and an icon representing its spatial data type, whether point, line, polygon or any other composite data type previously defined in the database. Figure 2.10 shows an example schema and the icons used to depict the basic spatial representations.

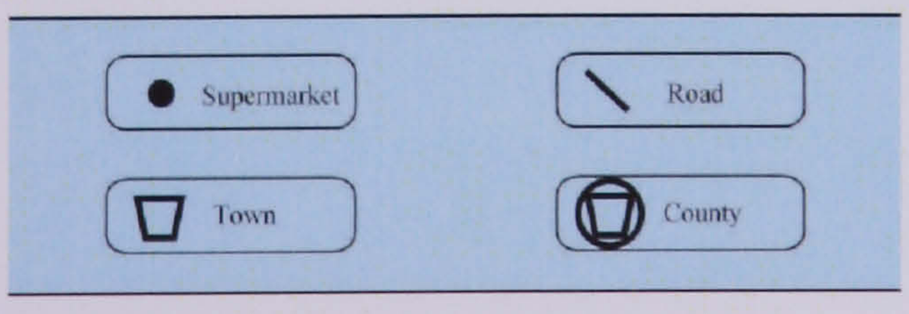


Figure 2.10: Example Schema, the Basic Spatial Representation of the Objects.

Simple queries are represented by simple filters that produce results based on non-spatial conditions and they can use any of the operators =, <, <=, >, >= and 'like'. Spatial (unary) operators can be included in any query to produce results related to

area, volume, perimeter/boundary, etc. Non-spatial (aspatial) filters are used to represent constraints over the stored attributes and the properties of the features. Figure 2.11 shows an aspatial filter that depicts the query that results in producing an output that includes all the roads with road type 'motorway'. The SQL for this query is: 'Select All From Road Where Road.rtype='motorway' '.

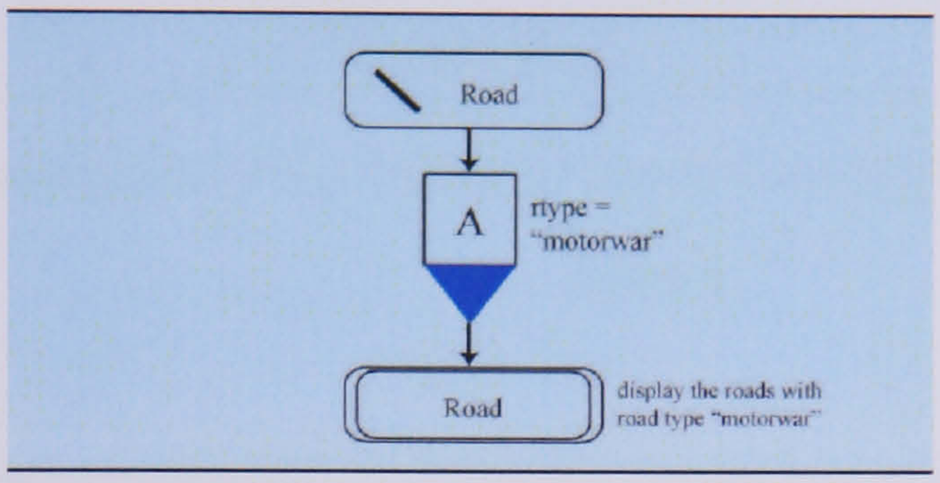


Figure 2.11: An Aspatial Filter in a Simple Query Construct.

The constraints are joined in series. The flow will pass through only when all the multiple constraints are satisfied. The negation of constraints is depicted by a line crossing any of aspatial or spatial filters. They may be applied on any or all of the constraints, but they do not affect the order in which constraints appear. Join operations are used to express a relationship between objects and spatial join operations are used to express a spatial relationship between spatial objects in the database. They are visualized by a rectangular box placed in spatial and aspatial filters. Each join filter may be linked to many result boxes associated with every joined object class. Figure 2.12 shows the join filters and an example of a spatial join query which is to find all the motorway roads that cross counties with population more than 50000. There are three types of spatial relationships: 1) Directional: 'North', 'South', 'East' and 'West', 2) Proximal: related to distances and the measure of proximity, 'within a distance of x meters', and 3) Topological: 'intersection', 'inclusion', 'adjacency', etc.

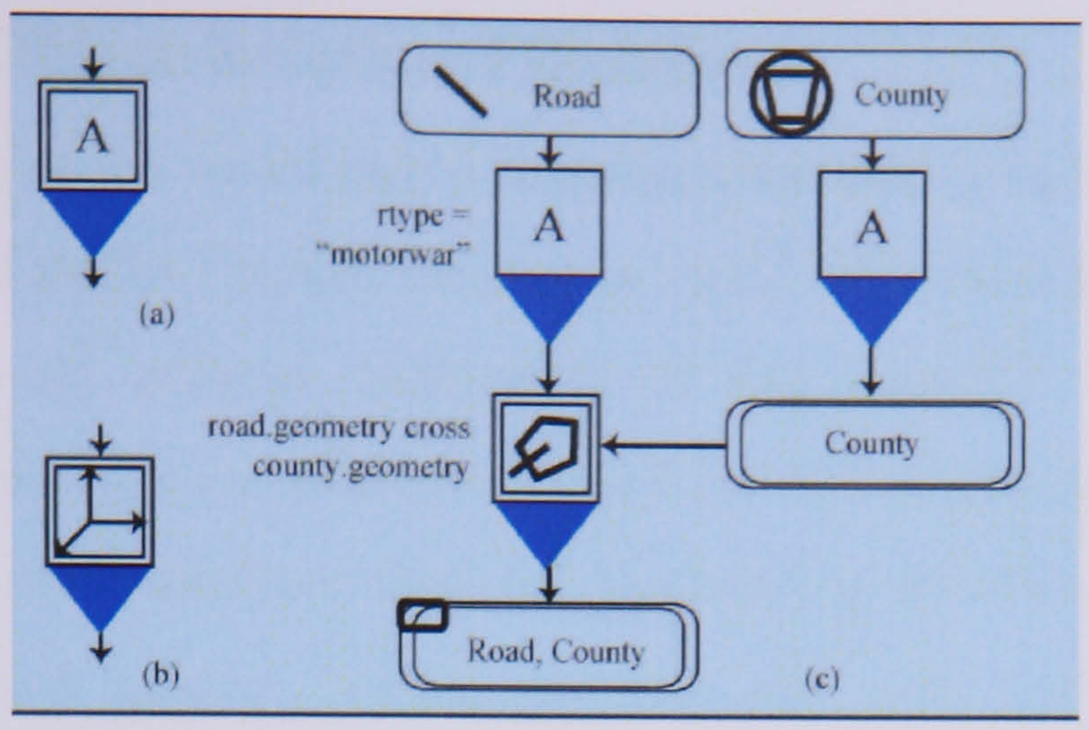


Figure 2.12: a) Non-Spatial Join Filter b) Spatial Join Filter c) Spatial Join Query.

The evaluation that was conducted in [Mor04] aimed at evaluating the filter-flow visual query language. The ArcView GIS interface was used to compare its query building process with the filter-flow visual query formulation. Evaluation tests for both the language and the interface have been designed to reflect and evaluate the expressiveness of each. Two subject groups of users were selected and classified as users with some experience of using GIS systems and users with no prior knowledge of GIS. The results showed that the Filter-flow visual query language simplified the learning process of the user, made the query expression process easier and proved to make the query expression more readable.

As a summary, the Filter-flow visual query language that is described in [Mor02, Mor04] may be used effectively to visualize queries associated with both spatial and non-spatial databases through a powerful visual query builder. However, it is not suitable for supporting spatio-temporal databases. Additional visual operators such as start, finish, during, etc. to deal particularly with spatio-temporal queries can be visually represented and added so as to extend the query power of the Filter-flow visual query language. Therefore, any new extension should provide a visual representation of the spatio-temporal relations and constraints using icons. It should also provide a visual depiction of the query filtering process the way it was provided in the Filter-flow visual language.

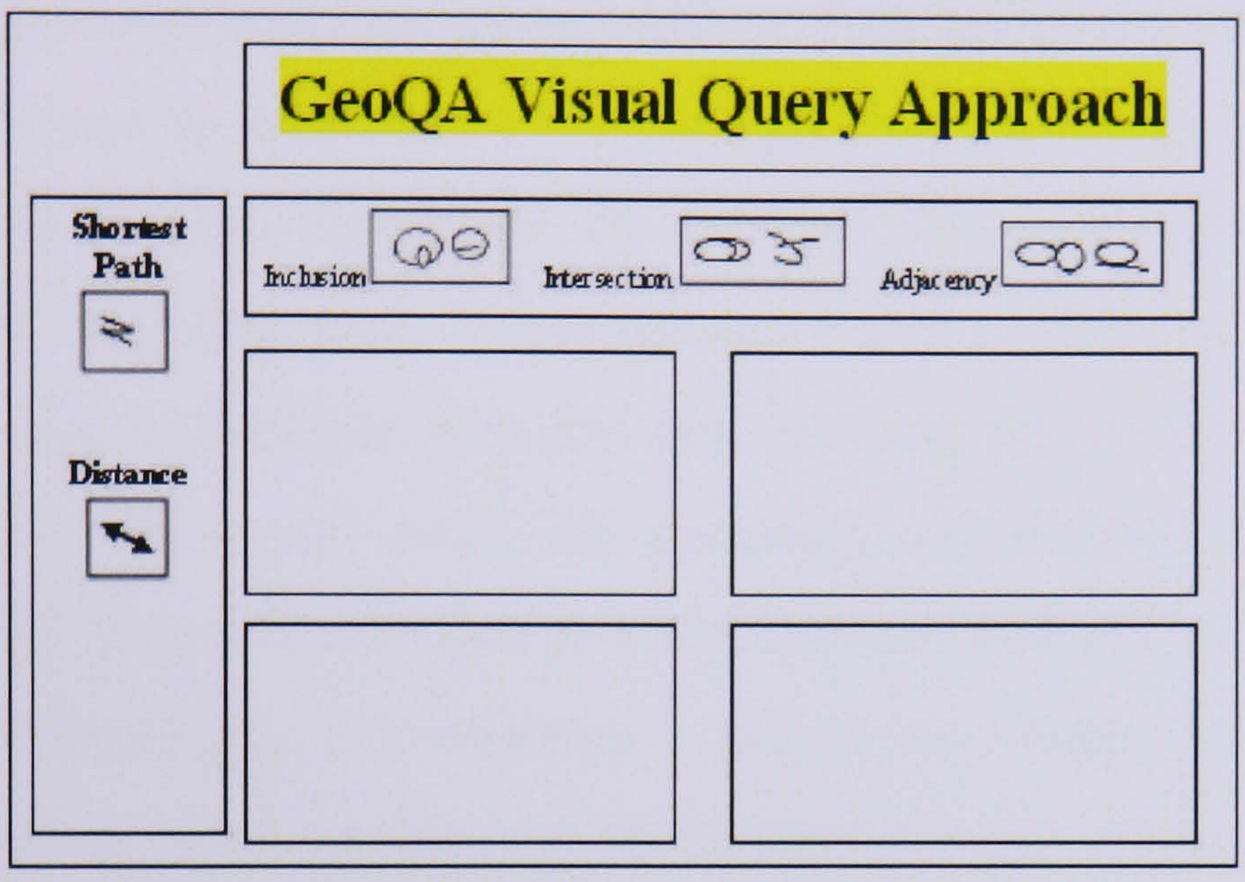
2.3.2 Querying Object-Relational Databases

The second example of the visual query languages that deal with both spatial and non-spatial databases is GeoQA [Sto00] which is a visual query and analysis tool that has been developed as an integral component of the GinisNT, an Object-Oriented geographic information system framework used for GIS applications development. It has an interactive visual interface used for formulating and processing spatial, non-spatial and combined queries, which provides the user with the capability of performing spatial, thematic and statistic analysis, with graphical presentation of query results for an easier spatial analysis of specific geographic situations. This may be applied in GIS applications that are used for urban planning, natural resources management, weather modelling, vehicle navigation, prevention of pollution and natural disasters, etc. It is deployed on the GIS application GeoTT which is developed for monitoring, maintaining, inventory managing, and analyzing the Telegraph-Telephone cable network in the Republic of Serbia, Yugoslavia as described in [Djo96, Sto98].

GeoQA graphical user interface is a WYSIWYG interface made up of icons, toolbars and a panel in which maps are displayed each in a separate window and may be moved, panned, and zoomed. It is supported with dialog boxes, forms, pop-up menus and direct object manipulation. Two sets of Icons are displayed in toolbars based on their function and representation. The first set contains icons representing all spatial entities. The second set contains icons representing spatial relations (topological, geometric and direction) and spatial operators (point, region, window). The user uses the mouse and the icons to manipulate directly and interactively the maps and objects that are visible on the screen. Dialog boxes and text input are used in formulating spatial queries that include spatial relations.

The user is provided with a variety of query facilities such as point-query, region-query, query composer, and SQL query builder, where a query can be saved for later loading and refinement. The output result is displayed in a window in the panel with or without the background map based on the user's choice. The GeoQA processor, the Mediator [Sto99], is for the OO front-end to RDBMSs, processes the non-spatial queries, and uses geometric filters, spatial indexing (grid structure) and computational

geometry algorithms to process spatial queries. The GeoQA interface shown in Figure 2.13 is based on the use of geo-referenced continuous raster maps, displayed as a background on the screen, that enable the user to move, pan and zoom in, as well as easily get all information related to a referenced spatial as well as non-spatial object in a map.



GeoQA, *Stojanovic 2000*

Figure 2.13: The Visual Query Interface of GeoQA.

The spatial operations that can be performed through queries are of two types namely the point-query and the region-query. The point-query is used to display a dialog box that contains all the thematic properties of a certain spatial entity. The region-query is used to mark by a specific color all the entities that are located in, adjacent to or overlap a certain region. The spatial querying and analysis is used to perform complex spatial queries that include spatial relations. Complex queries can also be formulated by two ways. The first is by using a specific GUI component called Query Composer and the second one a combination of icons called Iconic Query Composition. In the Query Composer, the query is formulated by using icons, dialog boxes and text interactively whereas in the Iconic Query Composition, the query is formulated by using icons only. The user selects from the toolbars all the icons that represent the elements needed to be defined and builds any query. The primary objects and the reference objects can be selected from the spatial entity toolbar whereas the spatial relations icons can be chosen from the spatial relations toolbar. After combining the

selected icons, the spatial query is shown on the screen in the form of a simple text. The user can review the query for possible mistakes before executing it.

The thematic and combined querying is used to perform alphanumeric queries through the SQL Query Builder which formulates the SQL 'WHERE' clause of a query. Its dialog form contains list boxes and icons. The user can build a query by choosing attributes (Fields) of spatial entities from list boxes, clicking on icons that represent relational operators (<, >, =), then choosing values from the value list boxes. To combine two or more conditions, the user can click on icons that represent logical operators 'and', 'or', and 'not'. While the user builds a query, the corresponding SQL 'WHERE' clause is displayed in a text box allowing him to check for any errors or mistakes, interactively, and make any necessary amendments before executing the query. The SQL Query Builder also provides the user with a set of icons that represent aggregate functions such as 'minimum', 'maximum', 'sum' and 'average' that are used for statistical analysis and can be applied on any set of attribute values for thematic querying and analysis.

As a summary and discussion, the GeoQA that is developed by [Sto00] is a visual query and analysis tool developed as an integral component of the GinisNT object-oriented GIS applications development framework. It is used for performing both spatial and non-spatial queries as well as analyzing spatial, thematic and statistical data. It provides the user with a user-friendly and ease-of-use visual interface and enables fast query processing. The query results are graphically represented for easier spatial analysis. GeoQA should be extended in the aim to cover the exploration of spatio-temporal queries by including new temporal concepts and relations hence making it a spatio-temporal query and analysis tool. A limitation is that no tests were conducted to evaluate the visual query interface. Usability testing should be conducted to determine whether the icons are understandable and convenient for users who perform data analyses in various spatial and non-spatial domains.

2.4 Visual Query Languages for Spatio-Temporal Databases

The existing visual query languages that deal with spatio-temporal databases are concerned with retrieving objects that satisfy spatial conditions about points, lines, and regions and at the same time vary with time or satisfy a temporal constraint. The spatial objects could be a building, road, or city and the non-spatial could be any attribute with a certain value. An example of these visual query languages is the LVIS [Bon02] which uses icons to represent objects and relations. Another example is the PHENOMENA [Lau03] which uses icons to represent spatial objects and temporal phenomena. The last example is the VISUAL TOOLS [And04] which use time-control panels to specify the beginning and the end of a time interval and the query is executed on an interactive map.

2.4.1 Querying Spatial and Temporal Objects

The first example of the visual query languages that deal with spatio-temporal databases is LVIS [Bon02, Bon00, Bon99] which uses Geographic pictures to represent spatial objects (object types) and Geometric shapes to represent relations among them (operators). It uses Balloons and Anchors placed on a temporal axis to describe spatio-temporal criteria. The user builds a visual query by selecting a combination of the icons that represent the pictures, shapes, and temporal axis. The temporal queries are built inside balloons which may be minimized in form of anchors for better visualization of the query under development.

The interface of LVIS contains a working area where the user builds the visual queries that can be validated then transferred to the query area either for execution or further refined. The query is then translated to a pivot language that is independent from the host query language of the target GIS platform and is compatible with the standardized Spatial-SQL [ISOI96] for spatial operators and with TSQL [Sno95] for temporal operators. Then, the visual query is translated from the pivot language into the GIS target language and the result of the query is displayed on a map. LVIS is an extension of the CIGALES [Auf95, Lba97] which is a visual query language for spatial databases where temporal aspects were not considered and the set of operators is very limited. Some of the icons that are used to visualize object types are shown in Figure 2.14.

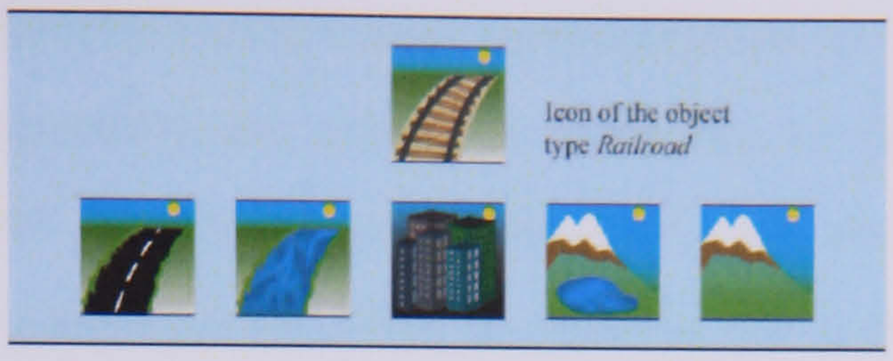


Figure 2.14: Sample of the Object Types of the Queried Database.

The three spatial types that can be handled by LVIS are represented by geometric shapes associated with each object of a query as shown in Figure 2.15.

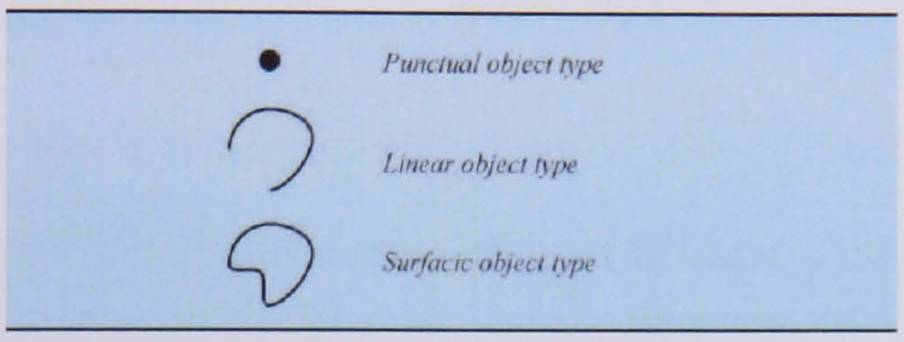


Figure 2.15: The Basic Elements of the Geometrical Shapes.

The thematic criterion of an object is visualized using a text located under its icon. During query building, a question mark is located above an object to show that this object is the target of the query, as shown in Figure 2.16.

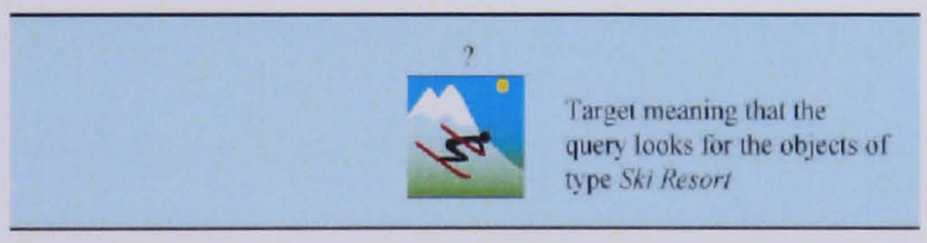


Figure 2.16: The Target Basic Visual Element.

A balloon visual metaphor is used to represent temporal criteria. It contains the specification related to a given object and can be viewed using a reduced representation called anchors. Figure 2.17 shows the anchor visual metaphor.

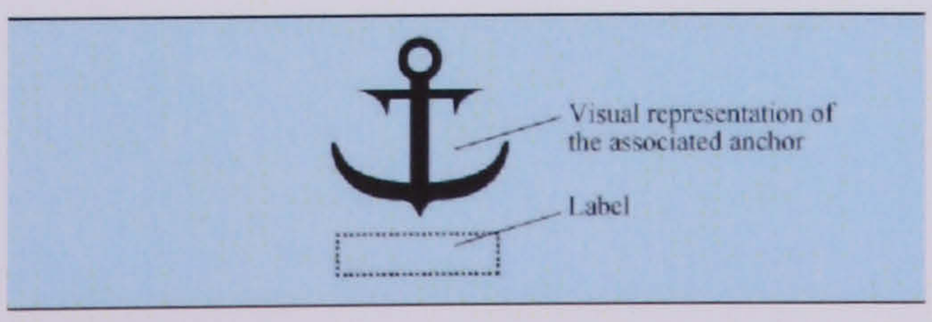
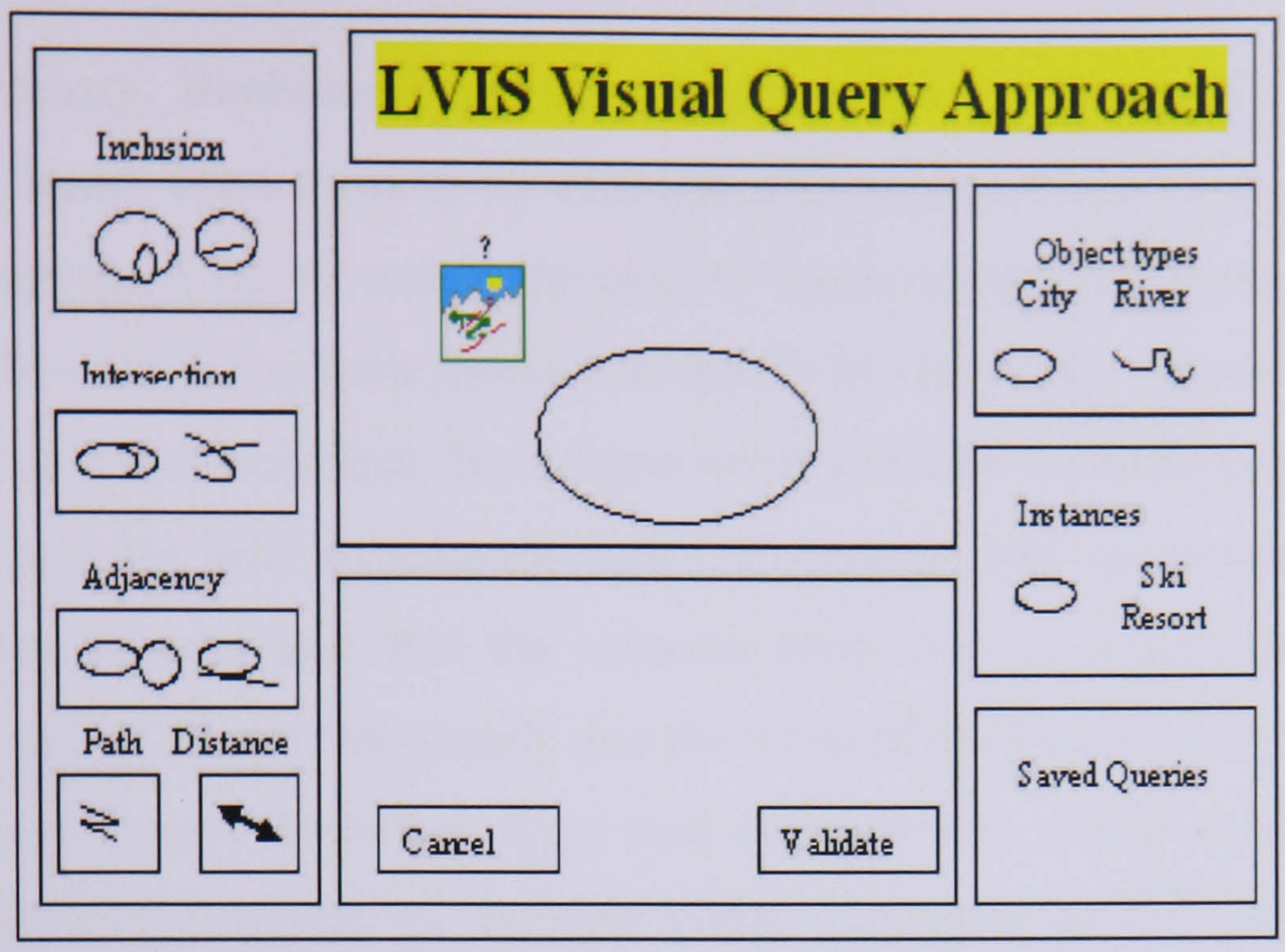


Figure 2.17: The Anchor Visual Metaphor.

The temporal criteria of a query are represented by interval values located onto a linear axis which is the main temporal object type that is handled by the model. If the period of an

interval is the null duration, another specific visual representation is used to express it and it is called the Instant. LVIS interface uses visual representations to express the operators that make up a query. The operators are classified according to their type: logical. spatial, temporal or spatio-temporal. The logical operators 'and', 'or' and 'not' are represented by icons. Other visual representations are used to express the spatial topological operators: 'intersection', 'inclusion', 'adjacency', 'disjunction' and 'equality'. The temporal operators of LVIS are either binary or unary and based on the definition given by Allen in [All83]: 'before', 'equals', 'meets', 'overlaps', 'during', 'starts' and 'finishes'. The Binary operators are used to compare two objects during time and the Unary operators are used to compare one object vis-à-vis a reference period.

Spatio-temporal queries involve at the same time spatial and temporal queries. They can also specify criteria such as the evolution of spatial properties and changes over time. The visual representations of the six evolution operators are 'creation', 'split', 'merge', 'growth', 'diminution' and 'destruction'. LVIS converts the visual queries to a textual language called a pivot language which is then translated into the host language of any GIS platform. There are three levels through which the queries have to pass in order to be processed: the visual, textual and target languages. The graphical interface of LVIS includes a set of icons representing the objects types as well as the icons representing the operators. In order to build a query in the working area, the user selects the object type icon which is then added to the working area, then clicks on the operator icon which applies the operation on the selected object types. For example, the effect of clicking on the intersection icon is shown in Figure 2.18. The user validates the query by clicking the Validate Icon. Then LVIS transfers the query to the validation area in order to be either executed or modified by adding further refinements such as applying new operators to the whole query or a sub-part of the current query.



LVIS, *Bonhomme 2002*

Figure 2.18: Selection of the Operator Intersection.

The evaluation that was conducted in [Bon02] aimed at evaluating the expressive power of LVIS. In order to perform the evaluation tests, the marketed GIS MapInfo has been chosen as the target platform as in [Bon01]. A set of queries has been defined for the testing process and they are classified as spatial, temporal and spatio-temporal queries. An example of a spatio-temporal query used is shown in Figure 2.19: *Which trucks did come into Paris before 6 p.m. and go out after 8 p.m.?* Two subject groups were chosen: persons working with GIS and persons who are non-experts in geography. The results have shown that the icons of the language are well accepted by the two populations.

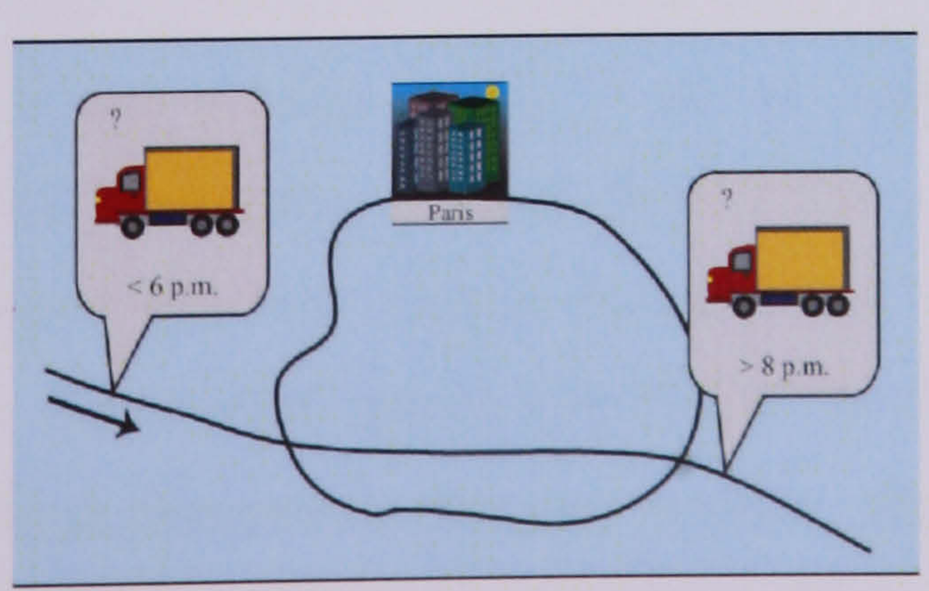


Figure 2.19: The Visual Representation of a Spatio-Temporal Query.

As a summary, Bonhomme in [Bon02] developed LVIS which is a visual query language based upon a query-by-example philosophy. LVIS is an extension of the visual language Cigales and is devoted to spatio-temporal information systems. In [Bon00, Bon02], it has been shown that the LVIS visual query interface has proven to be effective in the sense that the recognition of its icons is effective to users especially the ones without prior experience with GIS systems and spatio-temporal databases. The results showed also that the subjects were able to recognize the relationship between spatial objects very easily and the icons of the language are well accepted by the two populations. However, these tests covered only spatial and logical operators. They should be extended to the new visual metaphors of balloons and anchors in order to test the recognition of the visual representation of spatio-temporal queries. A limitation is that the tests were done on paper instead interactively where a subject is working on the interface of the visual query language, trying, testing and evaluating in a real-life experience, hence, making his sense of judgment better.

2.4.2 Querying Continuous Fields

The second example of the visual query languages that deal with spatio-temporal databases is PHENOMENA as described in [Lau03, Pao03, and Pao04] which is used for Continuous Fields. It manages both continuous fields and discrete objects in a uniform manner. Continuous fields represent real-world events and phenomena that are related to the environment and its resources, such as temperature, pressure and electromagnetism. Phenomena are measured by distinguishing what varies and how smooth their variation is. Discrete objects represent points, lines or areas. In the visual user interface of PHENOMENA, the user is provided with the capability to capture some features of a scenario by selecting an area of interest and handling the events involved. The user can select portions of continuous fields and then formulate queries based on spatial conditions visually using a set of suitable metaphors called geometaphors. When building a query, the user chooses first a continuous field that is represented as geographic data and then a function that describes its behavior. Combining continuous fields with spatial conditions provides the user with the capability of querying those phenomena that vary in space, such as the temperature on a region, the density of population in a city, the pollution in a river, a surface

elevation or the point where the wind has the maximum power. Thus, PHENOMENA is intended to be used by domain experts such as geologists, meteorologists, archaeologists and sociologists.

The graphical user interface of PHENOMENA is made up of two areas. The area on the left contains the icons that represent continuous fields such as temperature and the icons that represent spatial objects such as river. The area on the right contains the icons that represent the aggregate functions such as 'minimum'. The user drags and drops the icons into the central tabbed panel in order to formulate the query that follows the SQL like Select-From-Where scheme. Once a query is formulated, it can be either executed or saved for later loading and refinement. PHENOMENA is used to define an extension of the standard OpenGIS SQL that is meant to manage continuous fields and to integrate them with discrete data. The icons that visualize the geographical region Rhone River, Temperature, and the function 'minimum', are shown in Figures 2.20, 2.21, and 2.22.

Figure 2.20: The Geometaphor Icon Used to Represent the Rhone River.

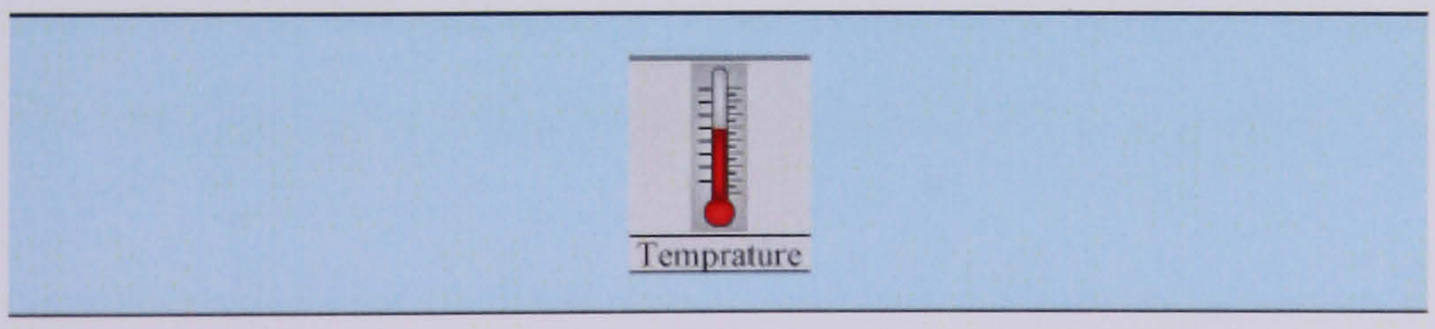


Figure 2.21: The Geometaphor Icon Used to Represent Temperature.

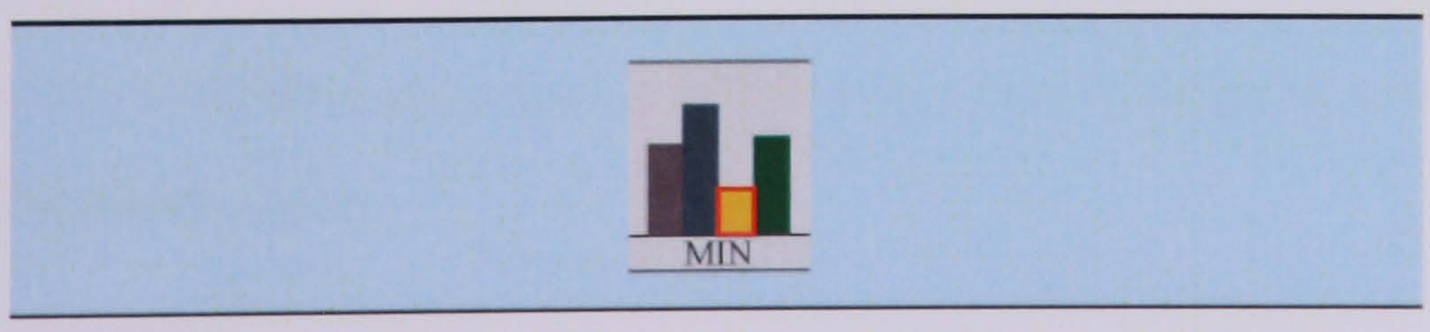


Figure 2.22: The Icon Used to Represent the Mathematical Function Minimum.

The features that are available have been grouped based on the following four categories: continuous operators, spatio-continuous operators, aggregate functions and

derived attributes. Continuous operators are used to select subparts of continuous fields, such as 'minimum point', 'maximum point', 'concave', and 'convex'. Spatio-continuous operators are used to select subparts of continuous fields based on domain conditions, such as 'interior', 'exterior', and 'boundary'. Aggregate functions are used to derive a collection of values from a dataset, such as 'minimum', 'maximum', 'mean', 'sum', 'count', 'intersection', 'union', and 'difference'. Derived attributes are used to extract important information using basic functions, such as 'surface', 'integral', 'area' and 'density'. Other spatial operators may also be included, such as 'touches', 'crosses', 'within', 'contains', 'overlaps', 'disjoint', 'distance', and 'equals'.

The visual user interface of PHENOMENA is made up of two areas. The area on the left which is called the Geodictionary contains all the geometaphors that are used to visualize spatial fields and continuous fields on which queries may be formulated. The area on the right which is called the working area is the interactive area where the queries are formulated. It is made up of two tabs, the Visualize tab is used for the aggregate, spatial and intensity icons and the Condition tab is used for constructing the WHERE clause of a query and constructing complex queries by applying logical operators and combining atomic conditions. PHENOMENA is based on the tree metaphor to allow users to construct combined queries by using the 'and' operator which is represented by a line connecting two or more conditions, the 'or' operator which is represented by branching the conditions and linking them by the LINK button, and the 'not' operator which can be applied on a condition by selecting the 'not' button.

Figure 2.23 shows the 'Intersection' function to be applied on all the geometries that resulted from the previously constructed conditions. The RUN button generates the SELECT-FROM-WHERE clause and displays it in a box in order to allow the user to verify the textual SQL query.

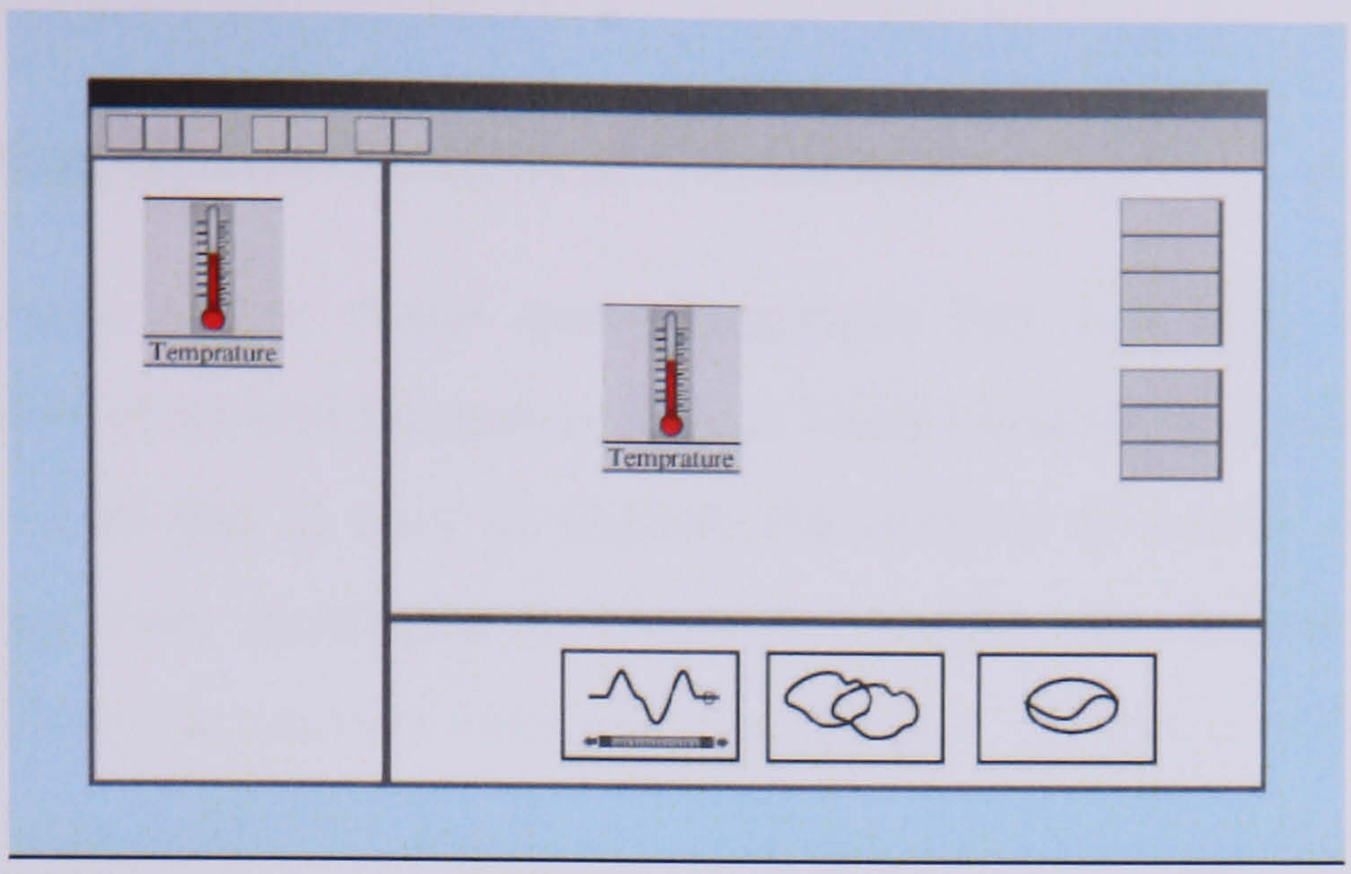


Figure 2.23: The Application of the Intersection Aggregate Function.

As a summary and discussion, PHENOMENA that was developed by [Lau03, Pao04], is a visual query language based upon a drag-and-drop philosophy, an extension of the OpenGIS SQL, and devoted to spatio-continuous information systems. Its major power underlies in the fact that it can manage continuous fields as actual spatial data and provides the user with the capability to be able to manipulate both continuous fields and discrete objects in a uniform manner. The same aggregate functions such 'minimum' can be applied on geographic data such as The Dry Status of Vegetation and on continuous fields such as Temperature. The results of both fields are visualized as spatial fields.

The evaluation of PHENOMENA that was conducted in [Pao03] aimed at evaluating the visual query interface which has proven to be effective in the sense that the recognition of its icons is effective to users especially the ones without prior experience. This task was made easy due to the usage of intuitive visual representation of both the continuous fields as well as the conditions that are involved. A major advantage is the use of geometries to select portions of continuous fields based on some spatial conditions applied. However, nothing was mentioned about any tests conducted in order to evaluate the assessment of subjects and the efficiency of the visual query building. PHENOMENA can be enhanced to include temporal fields operations. The time parameter should be considered in order to include real life phenomena, such as the study of the temperature on Earth, the analysis of the pollution, following guided tours, querying about a certain geographic area of interest and exploiting activities.

2.4.3 Exploring Spatio-Temporal Variations

The third example of the visual query languages that deal with spatio-temporal databases consists of several Interactive Visual Tools [And04] which are an extension of the CommonGIS that is used to explore the analysis of spatial data. The early visual tools that were developed in [And03b, And02, and And99] extended the CommonGIS power to include only the visualization of spatial data analysis and comparison. Those tools were limited in the sense that they tackled spatial data not spatio-temporal data. The latest interactive visual tools support the exploration of spatio-temporal data, i.e., the data that vary temporally while referring to spatial locations as elaborated in [And00, And03a]. They provide the user with the possibility to manage spatio-temporal data such as crime analysis in a certain geographic area over a certain period of time. Different types of crimes can be analyzed such as, burglary, vehicle burglary, violent crime, murder, property crime, etc. The tools provide the user with the capability to explore the variations in the dataset and the changes of thematic properties, i.e. values of attributes.

The tools include a variety of interfaces for output analysis such as animated thematic maps (time-controlled maps), map series, dynamic transformations of the data, value flow maps and time graphs. The interface environment provides the user with time-control icons located in a time-control panel to specify the beginning and the end of a certain period of time during which the changes have occurred on a spatial location. The user can analyze the temporal changes that occurred on a specific spatial location based on the type of analysis needed, such as the output that is produced as a set of maps called map series used to compare many maps that are displayed serially in one panel. These new tools have been added to the previously developed ones to extend the power of CommonGIS in the aim to make it include the comparison and analysis of spatio-temporal data. They can be used by exploratory analysts in the analysis of thematic spatio-temporal data because they use time controls to allow the user to set the starting and ending of a period of time as well as the time-moment at which any analysis or map animation starts.

The first Interactive Visual Tool's interface is based on a time-control panel that includes time controls to set the starting and ending of a period of time. The user can set a moment in time that belongs to this period of time. Then, the tool will display the data values and attributes that are related to the spatial location at this pre-set time moment. This is called time-controlled thematic maps or time maps. The user can display the data values of the previous and next years by clicking the back and forth icons and he can also set the value of the step to the number of years he wants to skip if he wants the output to display the data values at different time intervals. The same time panel is also used for running map animation. The play icon displays automatically the changes of the data values on the map taking into consideration the step value. The delay value, which is like a waiting time between redrawn map outputs, is used to control the speed of the animation and allows the user to have enough time to look at the maps and their changing data values.

The second Interactive Visual Tool was developed to handle map series. The map series are used to compare multiple data values distributions of the same attributes that are related to the same spatial map but at different moments in time. The maps are displayed serially one below the other in the same panel to provide the user with a clearer view of the maps and allow him to evaluate local changes, i.e., how the attribute values have changed from one moment to another at a given location. If the user points the mouse cursor to an area on any of the maps displayed in the panel, a popup menu displays the attribute values of all the time moments that are represented in the whole panel. The third Interactive Visual Tool was developed to handle dynamic data transformations (called transformers) used to replace the current attribute values that are displayed on a map with transformed values, i.e. those that reflect the differences between current attribute values and those of another time moment, without the need to redraw the map displayed again. Different colors are used to represent positive and negative differences for clarity purposes.

The fourth Interactive Visual Tool was developed to handle value flow maps which can be used to explore the temporal variation of attribute values (called behavior) at a particular spatial location. It displays on each area of the map a small diagram called value flow symbol which is made up of a horizontal axis that corresponds to the time dimension and a vertical axis that corresponds to the values of the attributes at a

particular location and can be compared to the mean or median at each time moment. The tool reflects on the maps the differences between the attribute values and mean of their locations. The positive behaviors are displayed in red above the horizontal axis and the negative ones in blue below the axis. The user can easily understand the results and compare the areas results using colors.

The fifth Interactive Visual Tool was developed to handle time graphs that display all the behaviors in one common coordinate system. The user can analyze and evaluate more precisely the differences between two or more behaviors at any time moment on a single time graph by selecting their locations. Thus, only the behaviors of the user-selected locations are shown for an easier comparison. If the user wants to analyze the general development trends of a certain area, he can also select to compare the mean averages, i.e. the line that connects the years averages. The tool allows the user to displays the average behaviors of many locations together on the same time graph for comparison purposes.

As a summary and discussion, New Interactive Visual Tools have been designed and implemented in [And00, And03a, and And04] using Java applets. Their main advantage is that they are interactive which provides the user with an immediate feedback and an online interaction while analyzing spatio-temporal data. When the user clicks an icon or selects an option, the tools automatically update the data representation on the maps that are displayed in the panel without the need to redraw any of those maps. Another advantage is that they produce many maps serially one under the other in the same panel for a better comparison and use different colors to show positive and negative values. Hence, the tools provide the user with quick reading, understanding, exploration and analyzing of spatio-temporal data. One limitation is that no tests were conducted to evaluate them. Usability studies should be conducted in order to determine whether these tools effectively support the tasks they are designed for and whether they are understandable and convenient for users who perform data analyses in various spatio-temporal domains.

2.5 Summary

This chapter has presented a thorough investigation into existing visual query languages and examined their user interfaces and query building processes. The investigation also covered the specification of which aspects of visual query languages are evaluated, the determination of evaluation method and the discovery of the type of queries that are handled. The reviewed visual query languages have proved to have significant contribution in the query language area. Although, they provide very different user interfaces they all give the user the ability to formulate a query by selecting icons or metaphors. In fact, a number of these languages allow the user to drag and drop the selected icons to a special working area in order to validate the query before its execution.

Moreover, the languages provides interfaces to a variety of systems and GIS applications, some are based on object oriented concepts, others are purely spatial but not temporal etc. which have impacted the design and the development of the user interfaces. Some of the visual query languages have been tested in order to evaluate the user interface and the user satisfaction while others have not. Finally, some of the conducted tests were not adequate either because they have not covered different usability aspects or the tests were not of interactive nature.

A common limitation to all the languages is that they are not tailored for Mobile GIS, they do not provide complex query formulation, and they do not include dynamic queries. Thus, there is a need for a new visual query language aimed at Mobile GIS, and that can handle dynamic complex queries. This should take into account the evaluation of the expressive power of the icons, the user interface and the query building process. Moreover, a thorough usability testing should be conducted through user testing and user satisfaction in order to measure its ease of use.

Chapter 3 - Query Optimization and GIS

3.1 Introduction

In order to access the data of any type of databases, a query is formulated, executed, and its results are reported to the user. The software component that is responsible for the execution of a query is the Query Processor which is also commonly known as the Query Optimizer. Its main purpose is to convert the query into low-level language and execute it. Therefore, query optimization plays a major role in querying databases and in fact it has been the point of interest of a lot of research [Liu03, Mou05, Pap03, Son01, Xio05, Yiu05, and Yu05]. With the emergence of Geographic Information Systems query optimization undertook a major change because each GIS query should be converted into an Execution Plan due to the nature of the data which is spatially referenced and includes maps. Hence, each operator in GIS such as 'Find the Nearest Facility' has a corresponding execution plan made up of a list of steps that should be executed in a certain order to produce the necessary output.

GIS Simple query optimization which is responsible for processing simple queries that are formulated with one operator only consists of converting the simple query into its corresponding execution plan, and executing the plan. Whereas GIS complex query optimization which is responsible for processing complex queries that are formulated with many operators, consists of converting each operator into its corresponding plan, and requires managing the multiple execution plans before executing them. This is due to the fact that commonalities might exist between the steps, which requires merging them and the sequence of steps might need to be re-arranged in order to construct one global execution plan for the whole complex query. Only then, the complex query optimizer can execute the global execution plan as elaborated in [And06, Elm05, Elm06, Mok05a, and Mok05b].

In order to evaluate the query optimization strategies of existing query processors a thorough investigation is conducted in this chapter which summarizes their advantages and limitations, identifies the approaches that are used, and determines the methods used for their evaluation. The existing query optimization strategies can be

categorized based on using the Sharing Paradigm [And01, And02a, And02d, And02e, And03, And06, Afe98], push-down Strategy [Elm05, Elm06], or both [Mok03, Mok04a, Mok04c, Mok05a].

The rest of the chapter explores each category of query optimization strategies. Section 3.2 explores the process undertaken in Query Optimization for the various types of databases and queries. Section 3.3 introduces the sharing paradigm as well as the push-down strategy. Section 3.3.1 is concerned with the multiple-client simple query processors which are based on the sharing paradigm by: 1) sharing memory for cashing input data, intermediate results, and query results, however they do not handle multiple dynamic complex queries such as the GIS Runtime System, and 2) sharing of the underlying space and object of interest for the type of queries that deal with finding if an object is located inside an area, however they do not consider the type of queries that deal with proximity analysis. Section 3.3.2 is concerned with the proposed multiple-client multiple-predicate query processors which are based on the push-down approach to provide the capability to swap or reshuffle the execution order of operators in an execution plan such as the Data-to-Plan and Histograms for Data Stream Management Systems. The proposed processors shows that building a uniform adaptive query optimization framework is the key in addressing building a query evaluation plan and determining the optimal plan by including: selectivity estimation, cost estimation, adaptive query optimization model, and an extension of query optimization to cover multiple multi-predicate spatio-temporal queries. They are concerned with sharing sub-plans of multiple queries but do not suggest sharing totally global execution plans by multiple similar dynamic complex queries. Section 3.3.3 is concerned with the continuous simple query processors which are based on both the sharing paradigm as well as the push-down strategy by introducing a new Shared Global Plan particularly implemented against the initial query execution plan, but do not perform sharing by merging the operators and objects, and do not handle sharing global execution plans by multiple complex queries with similar scenarios, such as the Pervasive Location-Aware Computing Environments (PLACE) for Spatio-temporal Streams. Section 3.4 is concerned with a ruler mechanism which provides the capability to navigate through hierarchical change and its propagation such as Navigating through Hierarchical Change Propagation in spatio-temporal queries but is not applied in the query optimization of dynamic complex queries. Finally, section 3.5

concludes with a summary of the advantages and limitations of the reviewed query optimization strategies.

3.2 Query Optimization

The main goal of query optimization is to minimize the cost of processing a group of queries while constructing one global execution plan for all the queries. Multiple-query optimization has been investigated by many researchers [Kan94, Par88, and Sel88] at the algorithm low-level representation of the queries instead of analyzing input at the query language level (command) to produce a global access plan. In order to convert the queries into executable code, the queries should undertake the following steps:

1. Decompose queries into high level primitive operations (e.g. Selection, Join, and Projection).

2. Perform common sub-expression elimination (CSE).

3. Reorder the execution of queries (Push-Down) in order to minimize the execution time.

4. Construct a global access plan.

5. Convert the queries into a low-level program (query translation).

The primary contribution of [Kan94] is that the query optimizer can recognize when temporary relations that are used to hold intermediate results can be reused, induce common sub-expressions which can be removed, perform common sub-expression elimination (CSE) that was previously introduced and suggested by [Par88, Sel88], and reordering the execution of queries in order to minimize the expected execution time.

The steps that are listed earlier apply for relational databases. However, after a thorough examination of GIS queries in general and Location-Based Services (LBS)

queries in particular, it is obvious that the same steps can be followed, particularly since most GIS are based on relational databases. Moreover, processing LBS queries has become of great importance due to the nature of LBS applications which include requesting the nearest facility, navigating a street network path turn by turn, locating people, receiving alerts such as a notification of sale on fuel or a warning about a certain traffic jam, finding stolen phones, calling emergency services, advertising, common profile matching (dating), automatic airport check-in, and paying to toll collection watch.

Two types of queries exist in location-based applications [Elm06, Mok05a]. First, the stationary queries also known as static queries, where the user issues a query about objects around him based on his current XY location. The result consists of one map that shows the objects with respect to his position at the current time instance. Second, the continuous queries also known as dynamic queries, where the user issues a query that should run for a certain period of time. The GIS server receives at every time interval the new location of the user from a GPS system. The result consists of multiple maps each representing the answer based on the new location of the user. The dynamic query stops when the period of time expires, the user issues a cancellation order, he reaches his destination, or gets disconnected. When a user is disconnected, his unit automatically issues a message to the server informing it about the interruption.

In order to build a global execution plan for GIS queries, the query optimization steps identified earlier can be extended and applied on "multiple spatio-temporal continuous multi-predicate queries" also known as "dynamic complex queries". The queries are decomposed, common sub-expressions are eliminated, the execution of queries is reordered, a global access plan is constructed, and the queries are translated into low-level programming. Moreover, the execution of a set of queries requires several "global execution plans" to be built. Only then, the query execution cost in terms of resources can be assessed. This is normally based on execution cost [And06, Elm06, Mok05a] which is the time it takes to be executed, the space it occupies in the RAM cache memory, and the number of I/O operations it executes.

3.3 Query Optimization Strategies

A lot of research has been made at different levels of query processing and optimization with the aim to produce cost effective results. At the level of common sub-expression elimination (CSE), the sharing paradigm has been used as a means to achieve scalability which is defined as the ability to handle faster a large number of queries. Sharing can be applied on the input data in order to eliminate the I/O operations [And01, And02a, And03], on intermediate query results in order to be re-used by final results, on query results in order to be re-used by other similar queries, on the underlying space where intersections of areas can be queried once, on query operator in order to execute once multiple similar queries, and on the object of interest [Mok03, Mok04a, Mok05a]. Whereas, at the level of re-ordering the execution of operations, the push-down strategy has been applied as a means to optimize the execution plan through swapping, reshuffling operations, or pushing them down in the plan in the aim to achieve a faster retrieval of the results [Elm05, Elm06].

3.3.1 Sharing Paradigm in Query Optimization

The existing query optimization strategies that use the sharing paradigm are concerned with processing multiple queries in data analysis applications. An example is the Virtual Microscope Processor [And02c, And02d, And06, Afe98] which is based on sharing data input to eliminate I/O operations, intermediate results in RAM, and query results to be used by other queries. It provides efficient processing of multiple-client queries in data analysis applications such as data analysis, data exploration, and visualization of large multi-dimensional and multi-resolution scientific datasets. Most data analysis applications access a subset of the data stored in order to analyze it and produce needed results. In a multi-user environment, many clients access the same dataset and perform similar analysis on the data. Thus, the server needs to execute simultaneously multiple queries in an efficient way in the aim to minimize latencies to the clients.

The work involved the design and experimentation of a framework that examines efficient strategies and runtime support for the execution of the queries [And06]. The runtime system was designed for shared-memory multi-processors. It tackles queries

that are user-defined, allows the input data to be shared, and the query results to be reused by other queries. It aimed at optimizing query processing by (1) maintaining intermediate data structures generated by queries for intermediate results, (2) caching input data in memory, and (3) providing support for multi-threaded execution where each query is executed as a thread. Caching methods are used to store query results in the memory in the aim to avoid the duplication of expensive methods computation, hence, speed up the execution of queries [And02b, And03]. These results along with input data can be used by other queries to produce new results. When caching is implemented at the server side, multiple clients can share the query results.

The running system was implemented as a middleware on the Virtual Microscope [Afe98] which is an application used for browsing digitized microscopy images. The architecture of the framework consists of four major components namely Query Server, Data Sources, Page Space Manager, and Data Store Manager. The Query Server is responsible for planning and executing the queries that are received from clients. It inquires about the available memory space from the Page Space Manager and the Data Store Manager in order to better schedule the queries. The Data Sources provides a flexible mechanism for storing datasets. Input Datasets are stored in the memory in a page-based form. Retrieving data stored on a local disk in the form of pages (chunks) is faster than retrieving single data items individually. Thus, the I/O overhead is reduced significantly as elaborated in [And02a].

The Page Space Manager as described in [And02b] controls the allocations and manages the buffer space available for input data. It implements the replacement policies that are specified by the server. Its main function is to keep track of I/O requests received from the multiple queries. It orders and merges them, thus eliminates duplicate requests and minimizes the I/O overhead. The Data Store Manager is responsible for caching input data and query results in order to be used later by other queries. First, it stores for each user-defined query its semantic information about intermediate data structures. Then, it provides dynamic storage space for intermediate data structures that are generated as partial or final results of a query. Its main function is to allocate the buffer space, assign a pointer to it, and return the allocated buffer.

The experiment that was described in [And02c] was conducted on an 8-processor Symmetric Multiprocessing (SMP) machine, running Linux Kernel version 2.4.3. Each processor was a Pentium III 550 MHz and the main memory had a 4 GB capacity. The emulator software, called Driver Program, was used to emulate the behavior of multiple simultaneous clients. Two scenarios were considered in order to examine the performance of the runtime system and for each scenario two executions were done; each considered as a case making a total of 4 cases. In the first scenario, 16 clients were emulated each generating 16 queries about the same dataset. The calculated value of the overlap index was large 70% reflecting a high overlap among queries. In the second scenario, 8 clients were emulated each generating 16 queries about disjoint datasets making a relatively small overlap index of 59%. Two executions were done for each scenario. In the first one, the Data Store Manager was ON, hence maintained intermediate results. In the second one, it was OFF, hence did not maintain intermediate results. The execution time in seconds of each case was recorded.

An evaluation of the performance was conducted in [And06] where a comparison of the 4 cases results showed that a better performance was obtained when the Data Store Manager maintained intermediate results. In the high overlap index case, the execution time decreased by about 30% to 40%. In the low one, it decreased by about 18%. Results were also reported for varying the number of threads, e-g, for the cases where the maximum number of queries allowed to execute concurrently varies from 2 to 16. The query execution time decreased as the number of threads increased. The same was done but varying the size of the data storage in the RAM from 64 MB to 512 MB. The results showed that query evaluation time decreased as the size of the data store manager cache increased. There was less I/O overhead due to the fact that the number of page requests decreased. The same was done by increasing the number of queries to 1600. The execution time decreased by 38%. In summary, Andrade [And03] developed a GIS runtime system for executing multiple query workloads through using Shared Memory. The strategy was implemented by using intermediate data structures and caching data in memory, with each query being executed as a thread.

3.3.2 Push-Down Strategy in Query Optimization

The existing query optimization strategies that use the push-down approach are concerned with processing execution plans where the order of execution of the operators affects the execution time without affecting the output result of the query such as having both the Selection and the Join operators in the same execution plan. In some cases and depending on the data records, the execution of one the Selection operator before the Join operator produces faster results than the execution of the Join operator before the Selection operator, and vice versa [Mok04b, Mok04c, Mok05b]. The same applies to the 'SELECT' statement that includes the 'INSIDE' operator and the 'WHERE' clause for spatio-temporal databases [Elm05]. The push-down strategy is based on swapping operators in an execution plan in order to get faster results. A lot of research work has recently focused on investigating means and criteria to determine which operators to push-down with the aim to have cost effective execution plans hence optimize query processing [Liu03, Mou05, Pap03, Yiu06, and Yu05].

An example of a query optimization strategy that is based on the push-down paradigm is described in [Elm06] which raises a number of challenges concerning the optimization of multiple predicate spatio-temporal queries. It is based on Spatio-temporal Data Stream Management Systems (ST-DSMS) which handle mainly two types of queries namely the snapshot spatio-temporal queries and the continuous spatio-temporal queries. A snapshot query is usually evaluated (executed) on the fly only once when it is submitted. Whereas, the continuous queries are repeatedly evaluated and their answers are updated every time a new location update is reported from the moving objects. Previous approaches to those challenges [Son01, Xio05] have addressed the moving queries that are made up of only a *single predicate* such as a range predicate or a *k*-nearest-neighbor (*k*NN) predicate. However, they do not consider the case where the queries are formed of many predicates.

Multiple-predicate continuous spatio-temporal queries play a role of great importance because they cover a large variety of real-life applications. One example of such queries is to continuously monitor the nearest hotels (a *k*NN predicate) ahead of the user's way (a range predicate) while he is driving on a highway. Another example is to find in which region of a continuously monitored city the number of suspects is

greater than the number of police officers. The impact of time and/or space, the distribution of the objects and/or queries, and the existing spatio-temporal operators raises many challenges. The first challenge is how to build a query evaluation plan to answer multiple predicated continuous spatio-temporal queries. The second challenge is to identify whether the evaluation plan is optimal with respect to the system overhead or not.

Hence, the proposed idea in [Elm06] is building a uniform adaptive query optimization framework that includes: selectivity estimation, cost estimation, adaptive query optimization model, and an extension of query optimization to cover multiple multi-predicate spatio-temporal queries. The selectivity estimation for evolving spatio-temporal data takes into consideration some of their properties such as periodicity and correlation. The Spatio-Temporal Histogram (ST-Histogram) in [Elm05] was used to estimate the selectivity of a continuous spatio-temporal query operator. ST-Histograms are grid based where the universe is uniformly divided into disjoint cells. They are built by monitoring the actual selectivity of the outstanding queries. The selectivity of the spatio-temporal operators is sent from the query executer to the histogram manager periodically in form of statistics called feedbacks.

Queries are represented as spots of light that reflect the region's selectivity in each grid cell where the intensity of the light is proportional to the fraction of the histogram region illuminated by the query. When queries overlap, many light spots are directed to the overlapped region. This results in a lighter intensity which means a better accuracy in the selectivity estimate. ST-Histograms can accommodate the selectivity of their corresponding single query operator. An extension was proposed in order to accommodate multiple feedbacks especially the spatial relationship between queries. A cuboid cost function was proposed to estimate the cost of a query which in turn provides a framework for adaptive query processing. It is based on the following factors:

1. Average lifetime of an input update in the query pipeline – this is the execution time needed until it is reported to the user.

2. Average storage required as internal states per input update – this is the amount of storage needed and the probability of getting stored.
3. Average selectivity of input updates – this is the ratio of the number of moving objects whose location updates are forwarded to the query pipeline over the total number of objects.

A data-to-plan grid was proposed in [Elm06] as a framework for an adaptive query optimization model for continuous spatio-temporal queries. In order to prevent executing the same query plan on all location updates, the data-to-plan grid directs each location update to a relevant plan. Hence, the location updates are forwarded only to the queries whose answer may be affected by them. A dynamic plan formation mechanism is used to adapt to the changes that occur in the system parameters which provides the facility to add, remove, or reshuffle the operators in the query evaluation plan. An extension of query optimization to handle multiple multi-predicate spatio-temporal queries was proposed. The idea is sharing common operators and sub-plans between multiple queries. A query evaluation plan might be altered in order to allow the sharing of an operator between multiple queries. In summary, Elmongui [Elm06] raised the issue of optimizing the query plan of GIS continuous spatio-temporal query processing and showed that building an adaptive query optimization framework is the key in addressing the challenges that are related to them.

3.3.3 Sharing Paradigm and PushDown Strategy in Query Optimization

The existing query optimization strategies that use the sharing paradigm as well as the push-down strategy are concerned with processing scalable incremental spatio-temporal queries. An example is the research directions outlined in [Mok04b, Mok04c, Mok05a, Mok05b] which deals with the query processing of continuous real-time spatio-temporal queries in a location-aware services environment. Continuous queries are based on progressive data accumulation on the server. Their results are produced at regular intervals or may be triggered when a certain event occurs. Since these queries are issued by moving objects, any delay in the response might lead to non-efficient results and obsolete answers. Hence, the need for query

optimization techniques emerges. The proposed idea in [Mok04a] is the usage of the *sharing paradigm* as a means to achieve scalability which is defined as the ability to handle faster a large number of spatio-temporal queries. The sharing concept is of three types namely sharing the underlying space, the execution of multiple similar queries (query operator), and sharing the objects of the same interest as explained in [Mok03].

The first type of sharing as described in [Mok04a] is sharing the underlying space which investigates the type of queries that are concerned with finding whether one or more objects, such as one or more cars, are located inside a spatial area. All the queries have the same object but different areas. The set relations that could exist between the areas are namely the inclusion, intersection, and independence. The inclusion applies where one area is completely included in another, is denoted by R1 $\subset$ R2, and here the first query is considered as *spatially contained* in the second one. The intersection applies where two or more areas have some parts in common, is denoted by R1 $\cap$ R2 $\neq\varnothing$, and the queries are considered as *overlapping queries*. The independence applies where there exists no common areas between them, is denoted by R1 $\cap$ R2 $=\varnothing$, and the queries are considered as *non-overlapping queries*. An example of the *spatially contained queries* is where the first query is to find if a certain vehicle is located in region R1 and the second query is to find if the same vehicle is located in region R2 where R1 $\subset$ R2.

An analysis was done as described in [Mok04a] to identify the sequence of evaluating spatially contained range queries. Since the first query's region R1 is totally included in the second's R2, it is obvious that executing each query alone will lead to searching the same region R1 twice, once for each query. It is optimal to search once the common region R1 and use the results for both queries. This is called sharing the space. However, is searching or evaluating R1 before R2 more efficient than searching R2 before R1? It was suggested that Cost Models should be developed in the aim to decide upon the optimal sequence of evaluating *spatially contained* spatio-temporal queries.

Thus, a plan was illustrated as a decision tree to visualize a cost model that reflects the expected number of comparisons needed for every sequence of regions, R1 then R2, versus R2 then R1. The expected number of each sequence is calculated as the product of the number of comparisons needed and the probability of finding an object in the region, which, by assuming that the total area of the space that includes the regions is 1, can be calculated as being equal to the area of the region. Each alternative sequence or query plan has its own estimated cost based on the cost model. The optimal sequence is the one with the lowest estimated cost. The challenge is to generalize the idea to multiple spatially contained queries.

The same cost model can be used for the *non-overlapping queries*. The generalization states that queries with larger regions or areas are evaluated first. It is obvious that this is due to the fact that the probability that an object lies in a larger area is greater than in smaller ones. The same cost model can be used for the *overlapping queries* along with the generalization of the non-overlapping queries. The larger intersections are evaluated before the smaller ones because the probability of locating an object inside them is higher than in the latter. The decision tree here produces an output where the result of evaluating a sequence of comparisons depends on the result of the evaluation of the previous one. Hence, the challenge was to develop systematic or heuristic methods that could build such decision trees for any number of queries in order to achieve a scalable location-aware server.

Sharing the query operator is the second type of sharing as described in [Mok04b]. The type of queries, that were investigated, is concerned with continuously informing the user about a group of objects that are located in different areas such as the number of vehicles in a region and when the number of vehicles in another region exceeds a certain limit. In the vehicle example, all the queries have the same underlying structure hence they share the same *Vehicles* table, but they search various regions. Thus, they have in common an SQL part hence they share the same following operator:

SELECT V.ID
FROM Vehicles V
WHERE V.location *inside* R_i

Where, the spatial operator *inside* checks whether a point lies inside a region R_i or not. In order to execute the common SQL part for N queries, the following initial evaluation/execution plan is repeated N times: (1) read the *Vehicles* table and (2) compare each vehicle location to the query region using a point-rectangle inclusion test. In other words, the table Scan operation becomes an intensive I/O operation. In order to avoid these repetitions and optimize the query execution, a new Shared Global Plan was implemented and evaluated, against the initial query execution plan, where a *Region* table is created for the queries regions *Region (RID, Extents)* where *RID* is the query identifier and the *Extents* is the rectangular query region. In the suggested Shared Global Plan, the *Vehicles* table is read only once and joined with the *Region* table based on the condition *V.location inside R.extents*. The output of the Join is the set of tuples of the form (VID, location, RID) indicating that the vehicle *VID* is inside the query region *RID*. In order to justify using the plan, it was necessary to find whether or not the cost of the Join operation between the *Vehicles* table and the *Regions* table was less than the cost of reading the entire *Vehicles* table N times.

Sharing the objects of the same interest is the third type of sharing as described in [Mok04c]. The type of queries that were investigated is concerned with continuously informing the user about the number of various specific types of objects that are simultaneously located in a certain area such as the number of cars and the number of trucks that are simultaneously located in a region. All the queries have the different types of objects but the same areas. They also have in common the following SQL part:

SELECT Count (V.ID)
FROM Vehicles V
WHERE V.type = *type*
AND V.location *inside* R

In order to apply the *sharing* concept, avoid repetitions, and optimize the query execution, another new Shared Global Plan was implemented and evaluated in [Xio04], against the initial query execution plan, where a query $Region_i$ table, similar to the one explained above, is created for each of the specific objects of interest. The push-down approach [Che02] was applied, where the Selection operator is pushed below the Join operator in the execution/evaluation plan. In other words, the Selection

is performed before joining. The *Vehicles* table is read only once using the 'SELECT' operator that includes the 'WHERE'/'OR' for a list of types. Each tuple is forwarded to its corresponding Join operator, i-e, joined either with the *Region1* table or with *Region2* table, based on the conditions *V.location inside Ri.extents*. Thus, the Join operation is executed and performed only between the proper query region and the tuples that correspond to the particular objects of the same interest.

This has led to the consideration of the following: (1) to develop a cost model to compare applying the push-down approach versus not applying it, (2) to study the difference between using two separate *Region* tables against using one Region table followed by a Selection operator, (3) to implement the selectivity estimation to check the cost of having a separate table for each object type, and (4) to explore having one table per group of object type instead of one table per type. The push-down in [Mok04c, Mok05b] is pushing down the Selection operator in the global execution plan in order to be executed before the Join operator. Similar queries are grouped in a query table. Then, the moving objects are joined with the moving queries by using a spatial Join algorithm but without any indexing structure. Moreover, an incremental paradigm was introduced in [Mok04a] to apply the sharing approach. The incremental evaluation was achieved by reporting only the updates that occurred on the previous answer of queries. The input is either from the Selection result or from the Join result. Its output is either *Positive updates* which report the addition of a new object to the previous answer or *negative updates* which report its removal.

Three different joining policies introduced in [Xio04] namely Clock-triggered Join Policy (CJP), Incremental Join Policy (IJP), and Hot Join Policy (HJP), were used in [Mok05a]. The Clock-triggered Join Policy (CJP) is mainly introduced for comparison purposes. In order to save on the continuous re-evaluation of all the spatio-temporal queries, the spatial Join is reevaluated every T seconds by joining all the records in the objects table with all the records of the queries table. Having large T values may lead to outdated results to the user, whereas, having smaller T values may lead to an excessive number of computations. Thus, the one minute T interval was used as per [Kol99, Xio04]. The major drawback of this policy was that at every evaluation, all the objects are joined to all the queries even if they did not change their location since the last T interval of time. The Incremental Join Policy (IJP) is mainly

used to avoid the drawbacks of the CJP policy. It does not execute the spatial Join for the objects and queries that did not change their location since the last T interval of time.

The Hot Join Policy (HJP) enhances the IJP policy. At each evaluation time, the objects and queries are identified as hot if their movements affect the query answer and as cold if their movement has no effect at all. To do so, each query is assigned a unique No-Action Region in such a way that an object or a query can move inside this region without affecting the query answer. When an object or query moves out of the former calculated No-Action region, it is identified as hot, otherwise, cold. A prototype database engine called Pervasive Location-Aware Computing Environments (PLACE) was developed for spatio-temporal streams as described in [Mok04a] and used to implement the query processing and optimization algorithms. The hardware used was an Intel Pentium IV CPU 2.4 GHz with 512 MB RAM running under the Windows XP operating system. The software *Network-Based Generator of Moving Objects* [Bri02, Mok05b] was used to generate randomly a set of 100K moving objects and 100K moving queries, based on an input map and its street network. The RAM buffers were not used in order not to affect the experimental results.

The results of the evaluation of the push-down approach, as reported in [Mok04c, Mok05b], showed that the size of the complete answer (number of records) remained constant for any percentage of moving objects. Moreover, it was orders of magnitude of the size of the answer returned by the push-down approach. The three joining policies were implemented and their performance was experimentally evaluated. Their I/O cost was calculated as the number of Input/Output operations executed. Whereas, their CPU cost was calculated as the time consumed and spent by objects and queries in the RAM memory. An execution cycle was defined as the process of executing all the queries. Ten execution cycles were implemented varying the percentage of moving queries from 1% to 10% respectively.

The results, as reported in [Mok05a], showed that for "moving queries on stationary objects" and "stationary queries on moving objects", the CJP policy had a constant number of I/O regardless of the percentage of moving queries. This was due to the

fact that the whole table of queries was joined with the whole table of objects every time a cycle was executed. The IJP policy had a number of I/O that was significantly smaller than CJP. The same applied to the HJP policy when compared to IJP. Also, the CPU cost of the CJP was two orders of magnitude higher than the other two. The IJP policy had a much lower CPU cost than CJP. The HJP policy had a lower CPU cost than IJP. The differences in the cost results were due to the fact that each consecutive policy avoided more and more the Join operation.

And finally, the results, as reported in [Mok05b], showed that for "moving queries on moving objects", the I/O and CPU costs had similar values to the above two types, but with a larger cost difference. This was due to the fact that most moving entities avoided joining costs. It was concluded that for any type of query, the HJP policy outperformed the IJP which in turn outperformed the CJP. The incremental evaluation paradigm was implemented and its performance was experimentally evaluated for the "moving queries and moving objects". The server buffered the updates received and evaluated them every 5 seconds. The results showed that the size of the complete query answer (number of records) was orders of magnitude of the size of the worst incremental answer [Mok04c]. While varying the percentage of the moving objects from 0% to 10%, it increased up to seven times that of the incremental answer.

As a summary, the approach that is presented in [Mok03, Mok04a, Mok04b, Mok04c, Mok05a, and Mok05b] is based on the sharing paradigm as a means to achieve scalability. It extends the push-down approach which is executing the Selection before the Join operation, to include the continuous spatio-temporal queries. The incremental approach was introduced which produces the updates in the results instead of re-executing the whole query again. The results of the experiments showed a significant decrease in the I/O and CPU costs. However, the work addresses only the cases of multiple simple queries where there is only a single query per user therefore it would be a challenge to address multiple complex queries where there are multiple queries per user. Moreover, it introduced a new Shared Global Plan particularly implemented against the initial query execution plan, but does not perform sharing by merging the operators and objects, and does not handle sharing global execution plans by multiple complex queries with similar scenarios.

3.4 Transformation of Natural Language Spatio-Temporal Queries

Since GIS are mainly concerned with real world features, its objects can be divided into three categories namely spatial entities, temporal entities, and spatio-temporal entities. The spatial entities represent geographical objects and have properties and attributes that describe how the objects are located in space. The temporal entities represent time and may be divided into time intervals which are periods of time with a beginning and an ending or time instances which are time snapshots of a specific time. The spatio-temporal entities represent how geographical objects change over time based on their attributes and behaviors which describe how the objects are located in space at a specific time instance. The spatial domain of a geographical object is described using properties such as size, color, shape, location in space, etc. and the relation between objects is described using topological attributes such as inside, intersect, overlap, etc. The temporal domain is described as a distance in time using a time-point or time-interval and the relation between time distances is described using temporal operators such as before, after, overlap, start at, end at, etc. The spatio-temporal domain represents geographical phenomena and describes how a geographical object changes over time.

One form of spatio-temporal queries is the natural language queries where the user asks a question about the change evolution of a particular area or city over a certain period of time such as "How has changed Orono from 1984 until 1996?" This query is about a spatial domain which is the state of Orono and a temporal time interval which is between 1984 and 1996 inclusive. However, the Orono state is made up of towns, roads, lakes, etc. which in turn are made up of buildings, etc., which in turn are made up of rooms, etc. In order to process this query the query processor should first examine the hierarchical propagation and decomposition of the spatial domain, reach to its atomic element (the smallest element), re-compose the query according to the semantics that are defined in the query language, and execute it. The same should be done for the temporal domain. In fact the process of decomposing and re-composing a query is called transformation. After the transformation process, the query becomes "Do all the rooms of Orono exist and none was changed for every day from 1984 till 1996?"

The existing query optimization approaches that use the transformation strategy are concerned with processing spatio-temporal queries in detecting changes. An example is the Hierarchical Change Propagation in spatio-temporal queries which is described by Mountrakis et al. in [Mou00]. There are three levels of change in spatio-temporal queries, the verification of the existence of an entity, the identification of a change of an entity, and the detailed description of this change. To describe how these three levels of change are propagated, a new dynamic classification scheme and its interaction with a spatio-temporal model were introduced. A tree-based hierarchy was used to represent the spatial as well as the temporal domains.

A transformation function can be applied between these levels in order to provide the user with the possibility to navigate through the tree hierarchy levels. The two functions that the user can manipulate are the Minimum Spatial Element (MSE) and the Minimum Temporal Element (MTE). They assist the user navigation in multiple spatio-temporal granularities. A sliding rule paradigm was also employed in order to support the navigation process through multi-resolution data. The approach is novel in the sense that it incorporates the concept of granularity in the query process of a spatio-temporal model.

3.4.1 Structure of the Model

A short presentation of the proposed spatiotemporal model is hereby introduced. The structure of the model is designed in a way to accommodate queries of different levels of details. Accordingly, change is decomposed in two levels: *entity* and *object*. The model consists of a *Geographical Identity Register*, a *Change Indexing Register*, and a number of *Child Databases*. At the top level of the structure, the *Geographic Identity Database* (GID) is used to store the information about the existence through time of every geographic feature. An abstract representation joined with a lifespan, as described in [Cli87], is stored without any explicit spatio-temporal information. The geographic feature is treated as an entity at this level. At the middle level, the *Change Indexing Register* (CIR) is responsible for the continuous communication and updates between parent and children as well as the consistency in their corresponding records. This register provides the essential multi-dimensional indexing mechanism, as described in [Lan92], which handles the flags to all the attributes of the *child*

databases that have been modified. At the lowest level, the *child databases* are used to store or express the actual characteristics of change of an object. There is a parent/child relation between the GID and the *child databases*.

The child databases are selected based on many criteria that aim at providing independence between them in order to assure minimal redundancy, and completeness in order to assure their adequate description of all the essential aspects of change. Each *child database* corresponds to a specific dimension of change. Three independent components of change have been considered, which leads to generating three *child databases*. The first is a Geometric child database which contains information about the outline such as the edges of an image or an area, the second a Positional one which describes the position of the object such as X, Y, Z coordinates and orientation information, and the third is a Thematic one which describes the use of the object such as identifying a building as being residential.

3.4.2 Categorizing Queries by Level of Detail

The structure presented above is based on three levels of detail that are hereby analyzed. A hierarchical structure was introduced to categorize the spatio-temporal queries based on three levels of detail. The first level is the *Existence of Entity* in which a qualitative Boolean query can be performed to check whether an object exists or not. The query is applied and executed on the parent database, the GID. The object is treated as an entity with no further spatio-temporal attributes. This type of query is referred to as identity based query, as described in [Hor00], which is Boolean in nature (single-source query). The second level is the *Existence of an Object's Change* in which a YES/NO query can be performed to check whether a change of a specific object exists or not. The query is applied and executed on the Change Indexing Register (CIR) without the need to access any of the child databases. This query is differential in nature (at least 2 sources are needed). The third level is the *Characteristics of Object's Change* in which a detailed query can be performed to retrieve a complete description of all the attributes of the object that changed. Only the modified attributes of each database are accessed and returned. The query is also differential in nature.

3.4.3 Spatial Domain

The aim is to introduce a tree-based hierarchy of classes and provide applications of propagation of change in queries at the spatial domain. In order to be able to detect changes in the spatial domain, a hierarchical structure was used to categorize classes based on the class containment relationships such as part/whole or container/contained relations as described in [Art96]. A tree-based structure of entities in the spatial domain was used in order to introduce these relationships as constraints to activate change propagation. The tree structure is dynamic in the sense that new levels can be added up or inserted between existing ones such as inserting counties between the state and road levels, or adding new instances to an existing level such as adding a forest class at the same level as the road level.

Moreover, any branch of the tree can be moved within the hierarchy without affecting the performance of the approach for example a road can be moved under a town. This is due to the fact that the sliding rule approach focuses on the relative arrangement of entities without considering the absolute distances between successive nodes. The resolution reference that is established to measure the change in the spatial domain is the Minimum Spatial Element (MSE) which expresses the spatial resolution of a recorded change. The MSE can be assigned either an absolute or a relative value. An example of assigning an absolute value would be when the MSE of a building is a cube of 1m x 1m x 1m, any change larger than this unit is considered as a change in the spatial object. An example of assigning a relative value would be when the MSE of a building is a wing, room, or brick. The advantage of the MSE relies in the flexibility that it provides when it is assigned a relative value. When the user considers a building to be a MSE of a town, a propagation process of change resolution is automatically triggered because the object types (classes) are being chosen differently. The MSE functions are projected in the tree structure that is shown in Figure 3.1 similarly to a sliding rule.

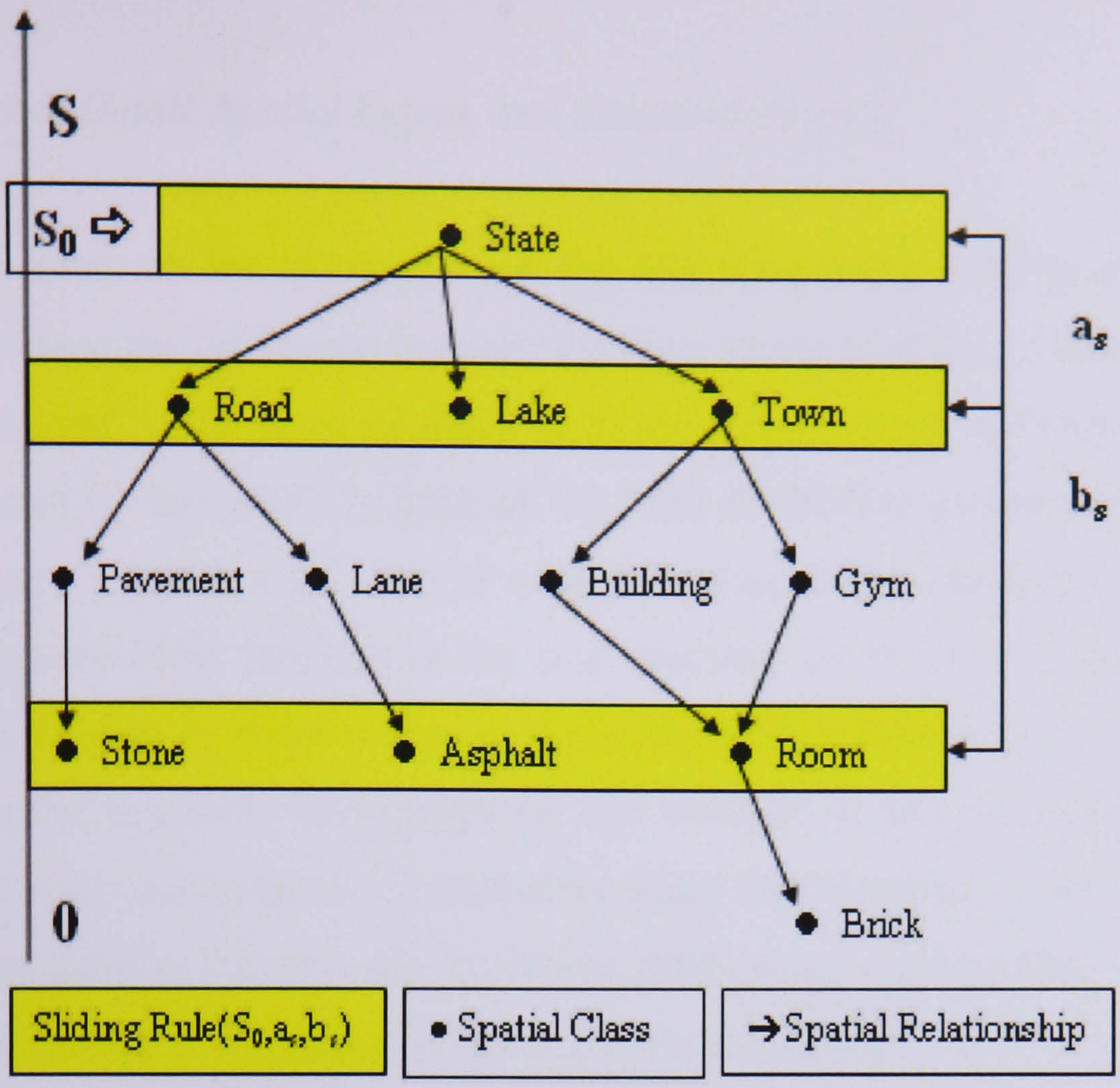


Figure 3.1: The Sliding Rule on Spatial Hierarchy.

The sliding rule has mainly three parameters. The first parameter is S_0 which specifies the original position of the rule and reflects the relationship between the spatial extent of the query and its corresponding class. The second and the third parameters are a_s and b_s which specify the distance in number of nodes for the two MSE functions. The MSE functions exist between classes and the containment relationships exist between objects. Conceptual conflicts can be avoided by combining class level and object level. For example, an object Memorial_Gym can have a part/whole relation with the object Orono_town, but there is no MSE relation between their classes Class_Gym and Class_Town. Another example would be that the Class_building can be a MSE for the Class_Town class, but the object My_House does not have a spatial relationship with any of the two objects classified as towns, simply because the building is located in another state.

During a query process, there is an interaction between the levels of details of the query and their implicit classification. The query is first decomposed into three parameters and second transformed into a range of resolutions. The three parameters are resolution, spatial extent, and temporal extent as shown below:

Query [*Level of Detail, Spatial Extent, and Temporal Extent*].

To elaborate more on the query process, the following query is to be analyzed and classified to show the difference between the three levels of details: "How has Orono changed last year?" The level of detail is 3, the spatial extent is "Orono", and the temporal extent is "last year". In general, the level of detail is assigned the value one when the query refers to a question of existence of an entity. The query is processed without using the MSE function or the tree structure. In the other cases where the level of detail might be equal to two or three, the system automatically moves down one class in the hierarchy and moves up one level of detail. The above described propagation rules can be better illustrated by using the following example. Having a database that contains 2 towns, 450 buildings, 5400 rooms, and 4 gyms, the user may define the following at the class level through the sliding rule:

Class_Building = MSE (Class_Town)

Class_Room = MSE (Class_Building)

He may define the following at the object level to ensure that the spatial relationships between the objects exist from time t_1 until time t_2:

Boardman_Hall = Part_of (Orono_Town, t_1, t_2)

GIS_LAB = Part_of (Boardman_Hall, t_1, t_2)

The following hierarchy represents the above defined MSE classification:

A) Class_Town

B) Class_Building

C) Class_Room

The Class_Town is assigned the letter A, Class_Building B, and Class_Room C. based on the class hierarchy defined above. When a query is performed at the spatial domain, the letter inherited from the class hierarchy (A, B, or C) is assigned to a code followed by a number representing one of the three levels of detail (1, 2, or 3). For example, if the query is "Did the town of Orono exist in 1921?", the generated code is A1 which is made up of the letter A that stands for the first class in the hierarchy Class_Town and the number 1 that stands for the first level of detail namely the Existence of Entity.

This level 1 query is addressed without using the MSE classification. If the query is "Has the town of Orono change during the past 4 years?" the generated code is A2. By applying the propagation rule, the letter A is converted into B, meaning that the query has been transferred to all the objects of the building class, and the number 2 is converted into 1, to compensate for change at the spatial level. Accordingly, the above query is rephrased and produces a new query which is "Do all the buildings of Orono exist and none was added for the past 4 years?" The way the classes have been defined above shows that the Class_Building class is the only one of interest to the user and that the class of the buildings is the only MSE for the class of the towns. If more details are needed about a town, a new set of MSE functions should replace the above ones and be declared as follows:

Class_Room=MSE (Class_Town)

Class_Gym=MSE (Class_Town)

This declaration is most convenient for the queries that belong to the level 3 type such as "How has the town of Orono changed from 1988 until 1996?" The same approach is followed by applying the propagation rule. The A3 code is converted into B2. After a second application of the propagation formula, the B2 code is converted to C1. The resulting query is now "Do all the rooms of Orono exist and none was added from 1988 until 1996?"

3.4.4 Temporal Domain

The aim here is to provide applications of propagation of change in queries at the temporal domain. To represent the temporal domain, the continuous time model of the reality is projected to the snapshot time model while still keeping track of the relationship that this projection is establishing. Every time interval is actually partitioned in one, two, or three sectors depending on the change format. Each sector (segment) corresponds to a state which could be either the sleep mode or the action mode. Figure 3.2 shows the two different modes on a single time scale, where the time instants T1 and T1+dt1 are respectively the beginning and the end of the time interval, and T1a and T1b are respectively the beginning and the end of the Action Mode.

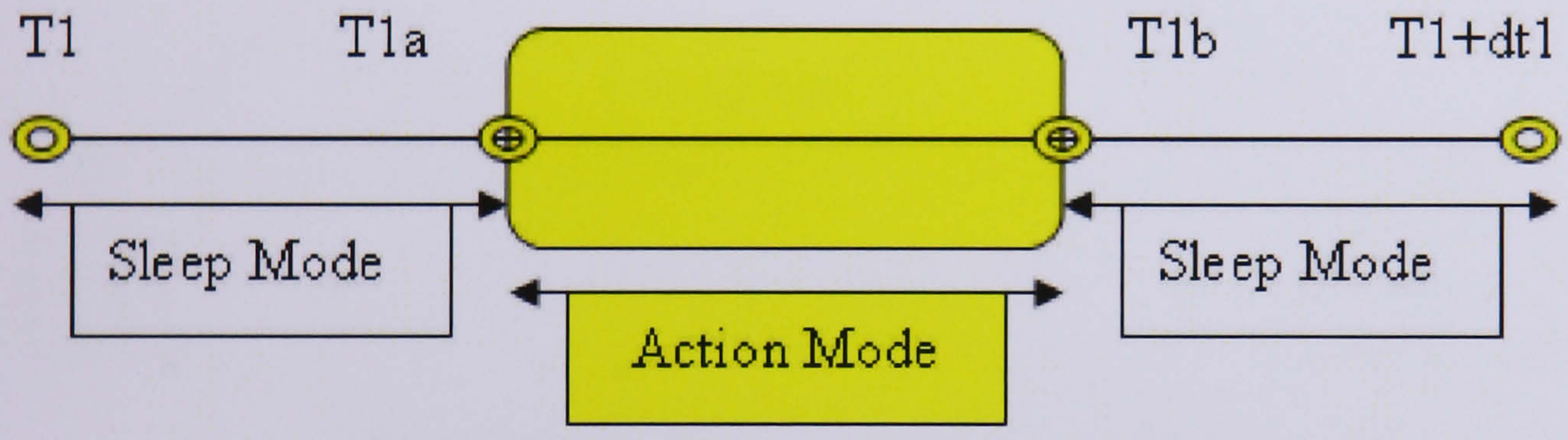


Figure 3.2: Sleep Mode versus Action Mode.

The action mode is an interval called "black box", has the label "action", and does not include any temporal information about the behavior of change. It can be discrete, continuous, periodic, or of any other form. The segmentation function is reapplied n times until a predefined interval acting as a threshold is reached. This predefined time interval expresses the minimum duration of change. Hence, it is defined as the *Minimum Temporal Element* (MTE). Figure 3.3 shows the segmentation based on MTE.

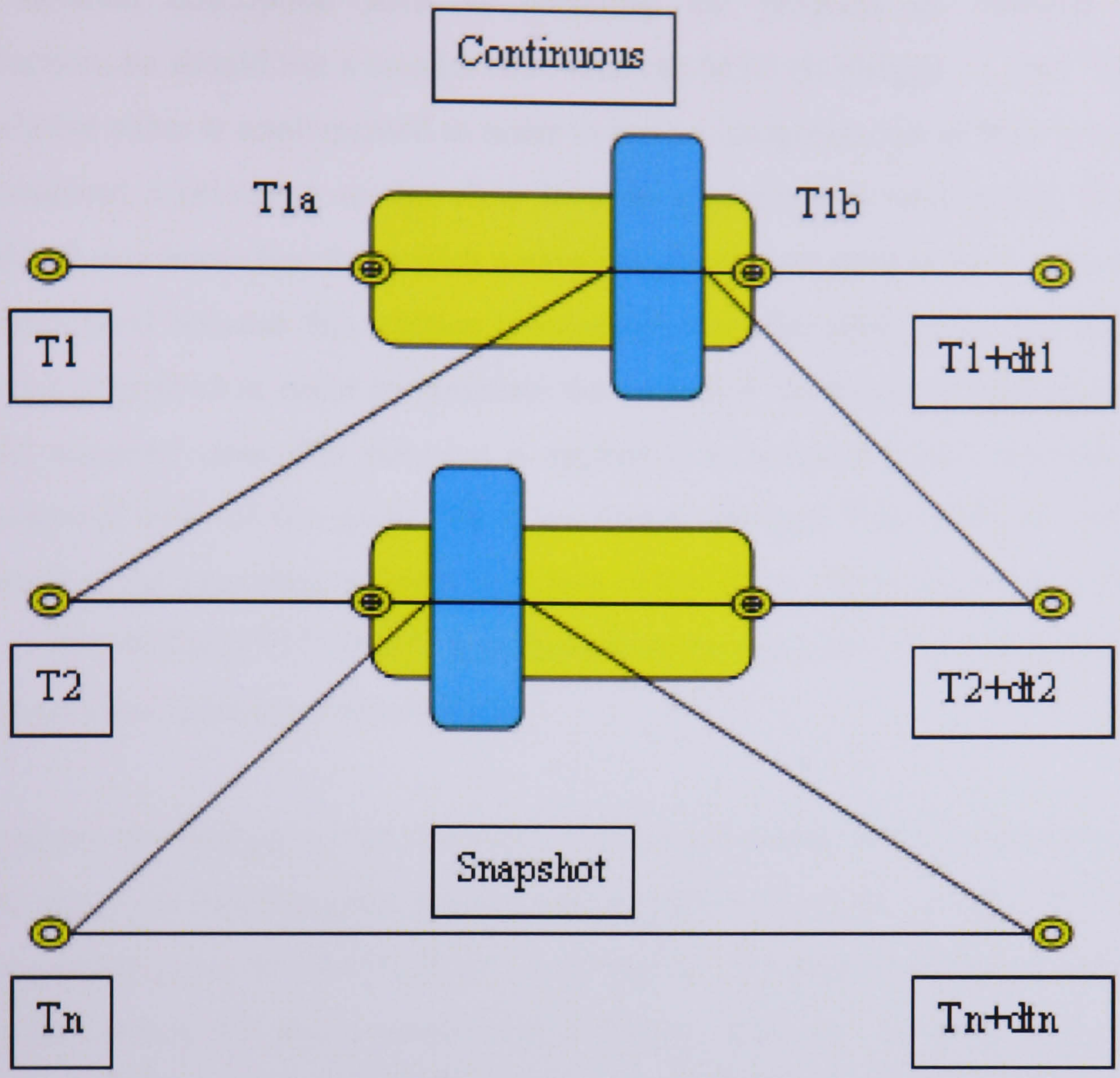


Figure 3.3: Segmentation based on MTE.

The continuous model of the reality is projected on the system snapshot model which in turn is back-projected to the continuous time line trying to reconstruct the object's temporal behavior. Sequential sleep or action intervals are joined together in order to reconstruct the new time line. The steps that take place during this phase are first the segmentation of the continuous model, identification, and reconstruction. A multi-resolution time line is employed with a coarser resolution where no change appears and a finer where there are changes detected. When a new time snapshot T_{new} is to be added and it has been found that it belongs in the time interval (Ti, Ti+1), then if (Ti+1-Ti) < MTE the above process is reapplied locally.

The concept of the Minimum Temporal Element is similar to the concept of MSE in the fact that it can have an absolute value or a relative value. The user uses the absolute value when he wants to establish a scaling factor in the temporal domain such as using information in a cadastre: MTE (Building)=1 month. In case he needs a

more detailed description such as detecting the progress of building under construction, he should use a smaller time interval: MTE (Building) =1 day. The use of a relative value is implemented in order to have a categorization of time in classes. The temporal relationship at the class level is a part/whole relationship. Time is considered as a linear function which means that there is no need to have a relation at the object level because this relation is inherited from the class level. The temporal hierarchy is applied in order to associate the temporal extension of each query to a specific temporal class. The rule that is applied is to assign the *most detailed* class. The temporal class of the query "How has Orono changed from 1988 to 1996?" is addressed at the year class whereas the class of the query "How has Orono changed from 11/06/1988 to 09/07/1996?" is addressed at the day class which means that the associated class is the most detailed one.

The queries that belong to the temporal domain are mainly divided into two types namely the point-based and the interval-based queries as can be easily understood in the description given by [Bet00, Bet97, and Sha98]. The point-based queries assign a timestamp during the query formulation such as "Did the car exist at 21:30 on 11/09/1999?" The detail level of this query is 1 which means that it addresses the change of the level of existence of an entity and that single-source type information can provide an answer to the query. The other two levels of queries are concerned with the existence of an object's change and the characteristics of such a change. They are differential which means that at least two sources are required so no timestamp can be assigned. This is why they are treated exclusively as interval-based which means that a certain duration of time has to be examined in order to get a valid answer. Since there is no continuous information in the model, every interval has to be projected in a discrete model. The same propagation rule that is applied in the spatial domain is also applied in the temporal one and can be better illustrated by using the following example where the user may define the following MTE functions:

Class_Month = MTE (Class_Year)

Class_Day = MTE (Class_Month)

If the query is "How has Orono changed in 1984?" it is transformed into "Did Orono change from January 1984 until December 1984?" The interval is converted to the units of the MTE through the MTE function. After reapplying the rule, the query is transformed into "Did Orono exist from 1/1/1984 until 12/31/1984?" which is now at the query level one. Another transformation is also conducted here in order to project the interval (1/1/1984-12/31/1984) to the snapshot model. A segmentation process is applied based on the temporal class Class_Day to which the temporal extension belongs. The temporal class is used as an MTE for the interval. The final form of the query is "Did Orono exist every day from 1/1/1984 until 12/31/1984?" which is a level one, point-based query.

3.4.5 Spatio-Temporal Domain

The objective is to provide applications of propagation of change at the spatio-temporal domain. Combining the MSE and the MTE functions allows the queries at the spatio-temporal domain to be introduced and addressed in the system. Table 3.1 shows the whole process that the system follows based on the functions introduced below:

Level of Detail	**MSE**	**MTE**
1) Existence of Entity	A) Class_Town	A) Class_Year
2) Existence of Object's Change	B) Class_Building	B) Class_Month
3) Characteristics of Object's Change	C) Class_Room	C) Class-Day

Input Query	**How has the town of Orono changed from 1984 until 1996?**
Step 1 **Decomposition**	Level of Detail = "How has changed" Spatial Extent = "Orono" Temporal Extent = "from 1984 until 1996"
Step 2 **Translation**	Level of Detail = 3 Spatial Class = Class_Town Temporal Class = Class_Year
Step 3 **Propagation**	Level of Detail = 2 Spatial Class = Class_Building Temporal Class = Class_Month
Step 4 **Propagation (2)**	Level of Detail = 1 Spatial Class = Class_Room Temporal Class = Class_Day
Step 5 **Snapshot Projection**	Level of Detail = 1 Spatial Class = Class_Room MTE = Class_Day Temporal Class = Class_Day
Step 6 **Recomposition and Reconstruction**	Level of Detail = "Does it exist" Spatial Extent = "Rooms of Orono" MTE = "Every Day" Temporal Extent = "from 1/1/1984 until 12/31/1996"
Output Query	**Do all the rooms of Orono exist and none was added for every day from 1/1/1984 until 12/31/1996?**

Table 3.1: Workflow of the Query Reconstruction Process

As a conclusion, the use of a dynamic classification scheme was presented by Moutrakis in [Mou00] with the aim to describe change and its propagation in spatio-temporal queries. Three levels of detail called levels of change have been identified namely existence of entity, existence of object's change, and characteristics of object's change. A minimum spatial element (MSE) has been introduced in the aim to act as a threshold to avoid the return of redundant information in the spatial domain and a similar one called minimum temporal element (MTE) for the temporal domain. The dependence of the model architecture with the levels of detail in a query has been discussed. The functions that could be defined under the model architecture with the presence of a dynamic hierarchical classification have been elaborated using examples.

Moreover, the sliding rule approach was used in order to apply the two minimum element functions to data that are structured hierarchically and a propagation function was defined as being responsible for specifying the initial position of the rule that related a certain query to specific classes of the data hierarchy. Therefore, the distance between the classes in the minimum element functions was measured by the number of nodes between them and two zoom-in parameters were introduced to express these distances. The user may navigate through different representations in the resolution domain through defining his semantic hierarchies. Examples have been presented in order to elaborate the navigation process.

3.5 Summary

In this chapter various optimization strategies have been reviewed. The Sharing Paradigm approaches the multiple single predicate queries by using sharing memory for caching input data, intermediate results, and query results. However it does not deal with multiple queries with multiple predicates. It discusses sharing of the underlying space and object of interest for the type of queries that deal with finding if an object is located inside an area. However, it does not consider the type of queries that deal with proximity analysis. It shows how a new Shared Global Plan can be implemented against the initial execution plan. However, it does not perform sharing by merging the operators and objects, and does not handle sharing global execution plans by multiple complex queries with similar scenarios. The push-down strategy

approaches re-ordering the operators of an execution plan based on the Selectivity of each operator according to a histogram, discusses how the updates in queries are executed instead of re-executing the whole queries several times through building a uniform adaptive query optimization framework based on selectivity estimation and cost estimation, and proposes sharing sub-plans. However it does not propose sharing Global Execution Plans for complex queries that have similar scenarios. The Sliding Ruler Paradigm incorporates the concept of granularity in the query process of a spatio-temporal model. However it does not deal with the elimination of the functions that are common to several execution plans.

In order to address the limitations of the existing strategies and aproaches a new visual query language optimization framework is needed, which is proposed in chapter 5. The new query optimizer should be developed for multiple dynamic complex queries and it is based on the Query Melting paradigm through common sub-expression elimination and sharing objects of interest, spatial areas, time intervals, underlying space, intermediate results, and Global Evaluation Plans. It applies the sliding ruler mechanism during the common sub-expression elimination phase. The query optimizer and the sliding ruler are both presented in Chapter 5.

Chapter 4 - Development and Evaluation of a new Visual Query Language

4.1 Introduction

With the increase in volume of non-expert mobile GIS users of proximity analysis and the need to formulate dynamic complex queries, new types of visual query languages specifically designed for the small screens of mobile devices are required. Thus, in this chapter a new iconic visual query language (IVQL) is presented which aims to provide the facility to handle dynamic complex queries, to allow a simple visual query formulation with an icon based user friendly interface and to be extensible in the sense that many extra icons can be added to it at any time and under any category. IVQL user interface is the front-end of the system where the visual queries are processed. The query processor main job is to read the queries, parse them into multiple simple ones, decompose them, optimize their functions and methods, generate their global execution plan, then execute it, produce the result of the query on a single map, and return it back to the end user for example in the form of a multi-media messaging system (MMS).

In order to facilitate the formulation of a query and not over-burden the mobile user with multiple formats for different types of queries, one standard format is adopted which is composed of the following constructs: 1) an operator that represents the task of the query such as Find Nearest Neighbor, Find Within a Buffer, and Find the Time Left to Reach a Destination, 2) a value that represents the number of objects to find, or the life time of a dynamic query, i.e., how long the dynamic query is supposed to keep running, and 3) an object that represents the facility or object that the query should search for such as Restaurant, Hospital, and Airport. IVQL is based on smiley icons that are used to visualize operators, values, objects, and on themes that are used to classify each group of icons according to their category such as Entertainment, Transportation, and Tourism. The formulation process of a simple visual query is a procedure of selecting the icons of the operator, value, theme that displays the icons of the objects that belong to this category, and the object. The formulation of a

complex query consists of formulating multiple simple queries separated by the 'and' operator hence the user is supposed to select the 'and' icon between them. The new visual query language is described in [Els06a] with its architecture, components and query constructs.

In order to evaluate IVQL, different aspects of usability are taken into account, thus the method of evaluation covers the evaluation of the expressive power of the icons, the user interface and the query building process, through usability testing which is used to measure the ease of use through user testing and user satisfaction. The evaluation is implemented using a prototype of IVQL User Interface, and its results are analyzed and discussed. The evaluation is introduced in [Els06b] detailing all the aspects to be evaluated as well as the method of evaluation related to each one of them.

This chapter starts by presenting the software architecture of the system where the visual queries that are initiated by IVQL are processed. The major components of the system are the IVQL user interface which forms the front end of the system, the IVQL query processor which reads the queries, parses them into multiple simple ones, decomposes them, optimizes their functions and methods, generates their global execution plan, then executes it, and the Geodatabase where the global execution plan is executed in order to produce the result of the query on a single map that is later returned back to the end user. In section 4.3 the elements of IVQL are discussed. Since the smiley icons are the core of the language they are used to visualize operators, values, themes and objects which are shown on the toolbars of the interface where the queries are formulated and verified/modified before being executed. Section 4.4 is concerned with the constructs of the language that are explained showing how a simple query is visually represented and how multiple simple queries can be concatenated or joined in order to form a complex query. Section 4.5 is concerned with the evaluation of IVQL where a prototype of IVQL is designed, implemented, and evaluated. Section 4.6 is concerned with the results of the evaluation and discussion of the results which include the evaluations of the smiley icons and the query formulation process. Finally, section 4.7 elaborates the conclusions that are drawn.

4.2 Software Architecture

The system consists of a number of components each of which plays an important role in processing the initiated visual queries starting from the query formulation and ending with the resulting map. The IVQL user interface forms the front end of the software. It is installed on the target mobile device where the user can formulate the visual query that is translated into a text query called the Pivot Language where each icon is replaced by its name, sent to the GIS server and saved in a file for later processing. The query processor, which is implemented as a middleware between the text query and the Geodatabase, processes each visual query separately. First, the query processor reads the query. If it is a simple one, it determines which operator is used to formulate it and accordingly considers its corresponding template execution plan. If it is a complex one, it parses it into multiple simple ones, associates each simple query with the template that corresponds to its operator, performs some tasks to optimize the queries, re-arranges the functions and methods, and generates a global execution plan for the whole complex query.

Second, it executes the execution plan, whether it is the template execution plan of a simple query or the global execution plan of a complex one, through using the Object Components which must be loaded into the development environment. Some of the components are the ArcObject Geodatabase which is used for handling workspaces, datasets, and feature classes, the ArcObject Controls which allow the developer to insert controls in their forms, the control MapControl which is used to add a map to the form and later zoom in, zoom out, and pan it, the ArcObject Carto which is used to handle Layers and add layers to maps, the ArcObject NetworkAnalyst which is used to make closest facility layers, add incidents, add facilities, set the parameters such as the number of facilities to find, the impedance attribute (Meters,..), show the path, travel directions, etc., and the ArcObject AnalysisTool which is used to handle a variety of tools such as the Buffer that draws a buffer circle around the user location and the Clip that finds facilities in the buffer area.

Third, it receives the result of the execution of each function/method which is performed on the underlying spatial data such as maps which are stored in the File-Based Spatial Data and organized by the RDBMS, and overlays it on the same single

map. Finally, the query processor sends the resulting map to the user. The Spatial Data Manager Environment (SDE) is a spatial data application server that has client/server architecture with software components used to perform fast spatial operations on very large data sets in order to rapidly serve GIS data to a large number of users while maintaining simplified data management. The Geodatabase which contains the File-Based Spatial Data and the RDBMS is managed by the SDE and its model provides users with the ability to add behavior, properties, rules, and relationships to their data. The software architecture is shown in Figure 4.1.

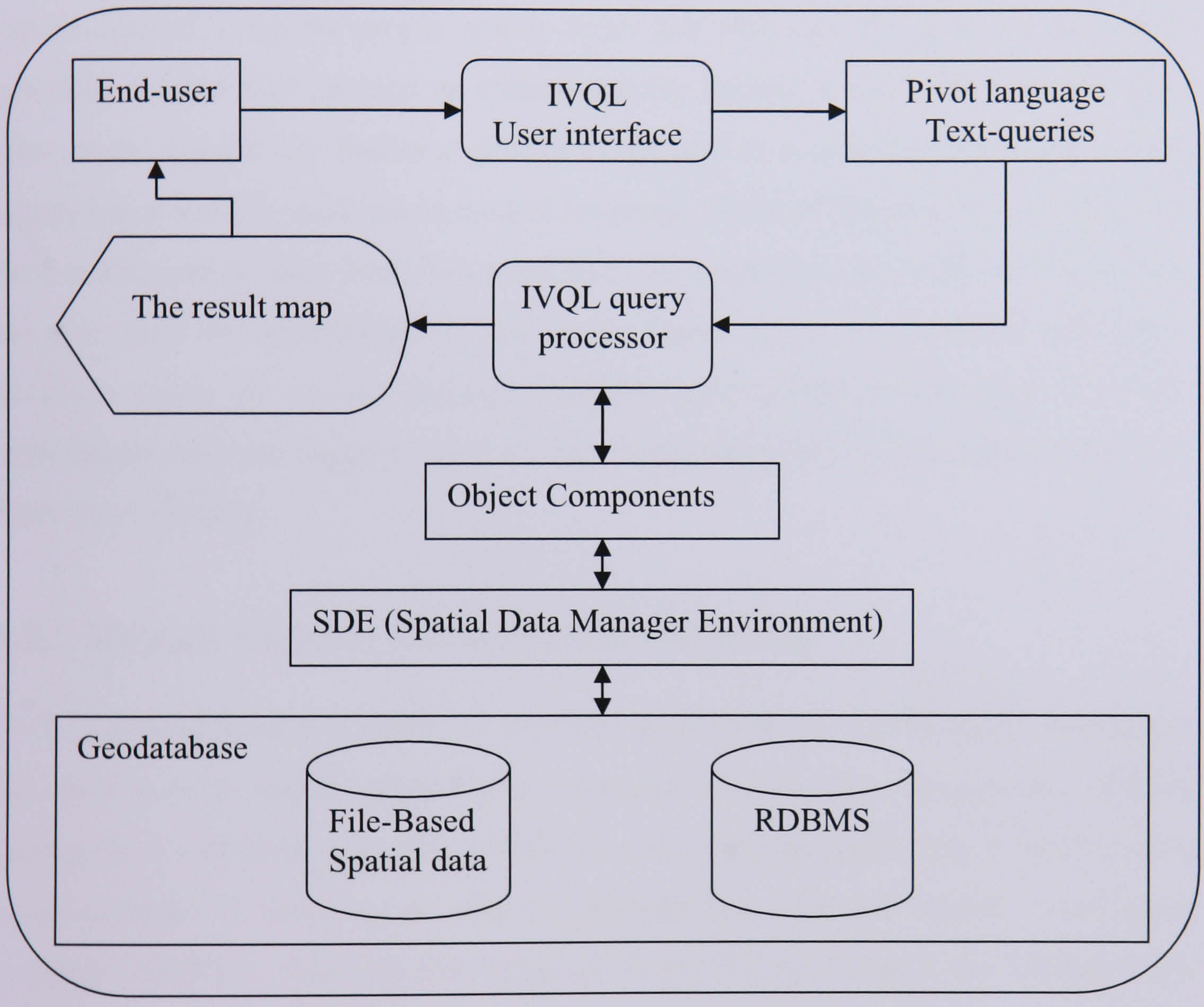


Figure 4.1: The Software Architecture.

4.3 IVQL Query Representation

IVQL has a global query representation based on smiley icons that are usually intended for users from different countries with different cultures and languages since they are text free and expected to have a high level of expressiveness. A smiley icon is a pictorial graphical icon that may express feelings such as happy, sad, angry, etc.,

represent activities such as running, swimming, etc., objects such as airplane, car, etc., and places such as London Bridge, Pyramids, etc. Since GIS are mostly concerned with the real world maps, most of their features can be represented by smiley icons.

Such a query representation can be adopted in many Location Based Services applications which are mainly concerned with finding facilities such as restaurants, hotels, theatres, hospitals, etc. It is particularly suitable for Proximity Analysis applications which are concerned with finding the nearest facilities to the user and the facilities that are located within a certain buffer area from the user. These facilities can be queried using the proper smiley icons that represent the operator, and which specifies what to find (nearest or within) and the facility to find. Thus, smiley icons seem to be suitable for building queries easily and in a user friendly manner while expressing a real life sentence in natural language. Some of the smiley icons that were used in this project have been borrowed from the askjeeves.com web site that granted the university the permission to use them. These icons are animated (.gif files). However, since the current version of J2ME development environment of mobile applications does not support .gif files, they were converted to .png files which made them static pictures.

4.3.1 Themes, Objects, Locations and Instances

IVQL can be applied to a variety of real life applications such as tourism, emergency, fire departments, police departments, customer relationship management (CRM), management information systems (MIS), etc. The tourism application is hereby taken as an example in order to be able to elaborate the elements of the visual query language. In IVQL, there are four types of icons. The first type is the Theme which represents a category, a type, or a group of objects such as, entertainment, transportation and tourism. The second type is the Operator which represents what the function is supposed to find such as Find Nearest Neighbor and Within a Buffer. The third type is the Value which represents the number of objects to find or the time interval of a dynamic query. The fourth type is the Objects which represents a location such as restaurant, hospital and hotel. Figure 4.2 shows the smiley icons of the three themes along with the objects that belong to each of them.

Figure 4.2: Basic Elements of the IVQL Visual User Interface.

4.3.2 Visual Representation of Operators and Queries

The queries can be formulated using the smiley icons by casual and non-expert users who have no prior knowledge of GIS information systems and query languages to databases. These icons can be easily read and understood which makes them the basis of an iconic visual query language, which does not use any text. No text is used in the language. Thus, it can be considered as an important step towards the globalization of languages. The use of smiley icons makes the process of formulating a query easier. The operators that are used in the IVQL visual query language represent and depict actions and instructions selected by the user in order to find the shortest path between two locations, the nearest neighbor, or all locations of a certain type within a certain distance. The difference between the icons used in IVQL and the one reviewed in the literature is in the purpose for which the icon is used. In [Mur00] the icons are used to represent the logical operators such as equals, less than, and so on. In [Smi04, Smi05] a line is used to represent the association operation. Whereas, the operators icons that are used in [[Lau03, Pao03, and Bon02] depict the topological operators such as intersection, union, and overlap. The smiley icon in Figure 4.3 is used in IVQL to depict the command: *FIND*.

Figure 4.3: The Smiley Icon that Visualizes the FIND Command.

Figure 4.4 shows the icon that represents the command: *FIND THE SHORTEST PATH.*

Figure 4.4: The Icon that Represents the Command: FIND THE SHORTEST PATH.

To *find the nearest* location of a certain type such as, find the nearest hospital or find the nearest restaurant, the user clicks the icon shown in Figure 4.5.

Figure 4.5: The Icon that Represents the Command FIND THE NEAREST.

In order to find all locations of a certain type such as restaurants or hotels, that are located within a certain distance from the user's current location, the user clicks the smiley icon shown in Figure 4.6. A list box is displayed containing numerical values. This list box allows the user to choose and select the distance in meters within which he wants to find all locations of a certain type.

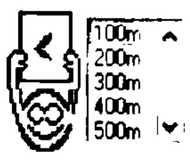


Figure 4.6: The Smiley Icon that Represents the Command FIND WITHIN A DISTANCE.

The constructs that form a Simple Query are an operator, a value, and an object. The operator specifies what the user wants to find such as the nearest, within a buffer, the time left, etc. as explained earlier. The value represents the number of objects to find, the distance, or the time duration. The object is the facility to search for such as hotel. The difference between the object icons in IVQL and the ones reviewed in the literature is in the form, IVQL does not use text at all. In LVIS [Bon02] the objects are listed in a list-box in text and in both LA [Smi05] and PHENOMENA [Lau03] the objects are represented by an icon with a text under it. The value icon used in IVQL may have different meanings based on the operator that it is being used with. When inserted in a buffer query such as Within 100 Hotel the value represents the distance of 100 meters. When inserted in the query Nearest 100 hotel it represents the number 100. When inserted in a dynamic query such as 'NearestAndTimeLeftToNearest' 100 Hotel, it represents the interval of time 100 minutes.

The dynamic query means find the nearest hotel and keep on informing me about the time still left to reach it for the next 100 minutes. In case the user wants to build complex queries that include a combination of simple queries, the joining command 'and' may be used. The user formulates the first simple query, clicks the 'and' operator icon, then formulates the second simple query. If more than two simple queries need to be formulated, the user follows the same steps explained above and

clicks the 'and' operator between every two simple queries. The size of the query, in terms of the number of characters, depends on the used technology, for example is the short messaging system is used on mobile devices, the maximum number of characters that an SMS message can contain is 256. So, mobile users can not build complex queries that are made up of more than 256 characters unless the multi-media message system (MMS) is used. Figure 4.7 shows the depiction of the 'and' operator.

Figure 4.7: The Smiley Icon that depicts the 'and' Operator.

4.3.3 Visual User Interface

IVQL provides the user with two major toolbars, one horizontal and one vertical. It has a middle area where objects are displayed and a query formulation area at the bottom of the interface. Each toolbar displays a set of expressive smiley icons that are easily understood by any user. The horizontal toolbar contains the smiley icons that represent proximity relations and operations that should be applied on icons displayed in the vertical toolbar. The vertical toolbar displays icons that represent themes and spatial object types. The middle area displays the theme objects on which the spatial operation is to be applied. The query formulation is done first by selecting a smiley icon from the horizontal toolbar. The icon is automatically moved to the query formulation area which appears at the bottom of the interface. Second, the user selects a theme of interest from the vertical toolbar. A group of all objects that belong to the selected theme are displayed in the middle area of the interface. Finally, the user selects the object needed. The selected object is then moved to the query formulation area. The user interface of IVQL is shown in Figure 4.8. The difference between the query formulation area in IVQL and the visual query languages reviewed in the literature comes in its form and its content. Kaleidoquery in [Mur00] and Filter-Flow in [Mor04] depict the query as a filter flow in a screen-size area. LA in [Smi04] depicts it as a network in a screen-size area too. Whereas in IVQL and the rest of the reviewed query languages such as GeoQA in [Sto00], LVIS in [Bon02], PHENOMENA in [Lau03], and VISAL TOOLS in [And07], the query is in the form

of a list of consecutive icons shown in a small query formulation area at the bottom of the screen.

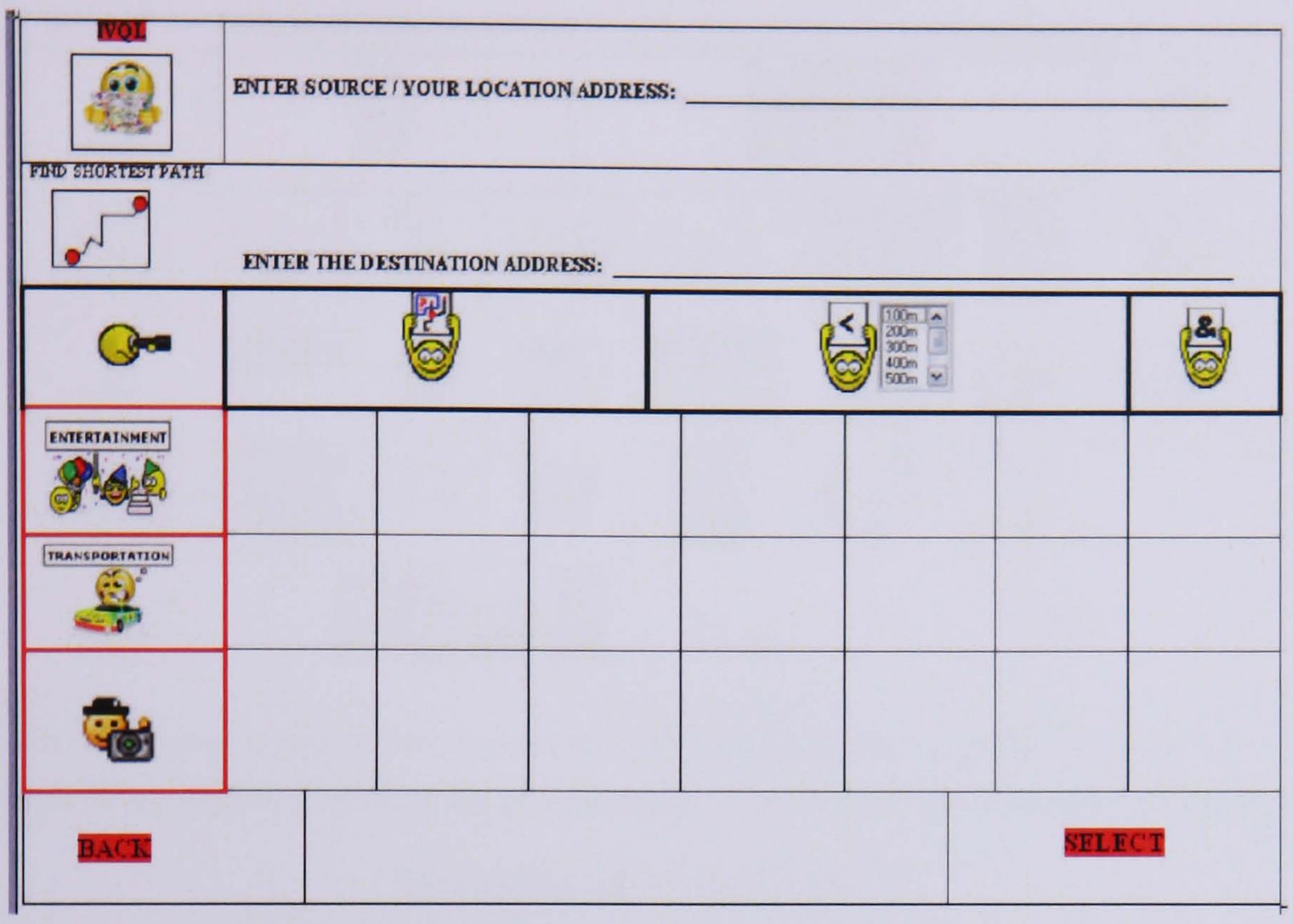


Figure 4.8: The IVQL User Interface.

The first two rows of the IVQL user interface are used to allow the user to text input the source and destination addresses. He can input his current location and input the address of the destination then query the shortest path between these two locations by selecting the "SHORTEST PATH" icon. In the first row, the location of the user could be automatically determined by a GPS, LBS (Location Based Services), RADAR, etc. So, there will be no need to include the Text Input of the user's location. The same applies to the second row. With the possibility of having an open-line connection, the user can query a list of hospitals and get back an output showing their locations on the map as well as a list box that includes the name and address of each hospital. In such a case, the user can select the hospital from the list box and use it as an input to the destination and then ask for the shortest path. Hence, there will be no need to Text Input the destination. To formulate complex queries, the user selects the 'and' icon and repeats the steps explained above. Figure 4.9 shows the IVQL user interface where the query: FIND WITHIN 500M ALL RESTAURANTS is formulated.

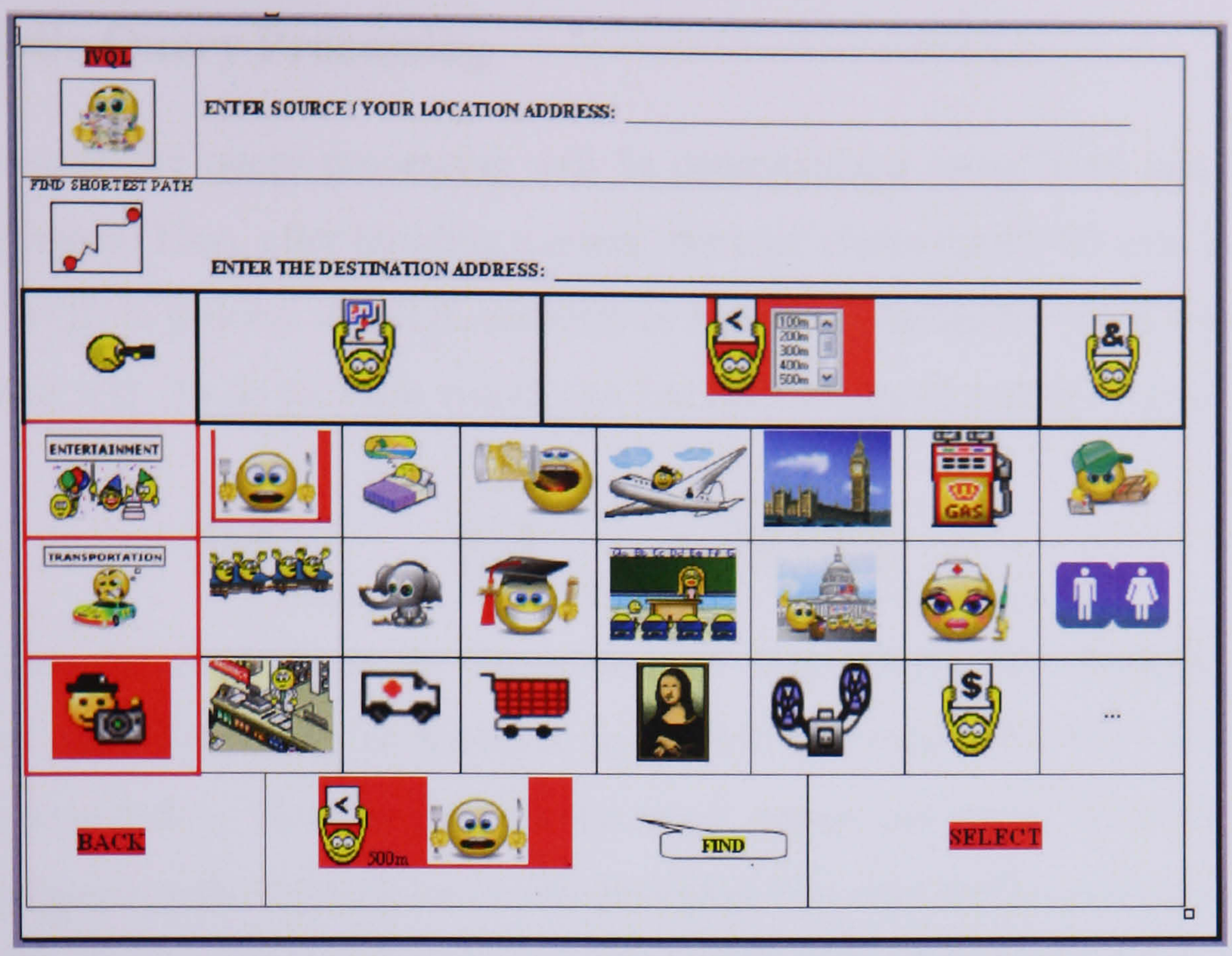


Figure 4.9: The Smiley Icons Representing the Tourism Theme Elements with a Formulated Visual Query.

IVQL is an extensible visual query language. Each of the sets of icons, that represent operators, themes, and objects, can be extended to include as many icons as the user defines and for any type of application. This is implemented by providing as many pages of icons as needed by the user. If the area of a panel is not enough to display all the icons, the icon "more" "…" appears as the last icon in the list to inform the user that there are more icons that could be displayed. The user selects it and a new list of the next group of icons is displayed in the panel. The advantage of IVQL over the visual query languages reviewed in the literature is that IVQL is a generic visual query language that can be applied in any field. Whereas, each of the reviewed visual query languages is developed for a particular application, for example, Kaleidoquery in [Mur00] and LA in [Smi05] are developed for non-spatial queries, Filter-flow in [Mor04] and GeoQA in [Sto00] for spatial and non-spatial, and LVIS in [Bon02], PHENOMENA in [Lau03], and VISUAL TOOLS in [And07] for spatio-temporal queries. Moreover, IVQL is extensible as elaborated earlier whereas none of the reviewed visual query languages provides extensibility.

4.4 Mobile Query Processing

In this section the query processing will be demonstrated using SMS and MMS on mobile devices. Thus, after building a query, the user clicks the FIND icon in order to instruct IVQL to process it. IVQL converts or translates the smiley icons visual query into normal text. To do so, each visual icon has been assigned a TEXT name, such as


nearest

within

restaurant.

IVQL uses the icon name to formulate the Text Query for example 'nearest restaurant'. The Text query is displayed as an SMS message in the SMS environment of the mobile device. The user chooses the 'send' option and inputs the phone number that is assigned to the GIS server by any provider. The sent SMS message is received by the GIS server and saved in a special file sequentially. The query interpreter that is installed on the GIS server processes the SMS messages sequentially using the FIFO philosophy. Each message processed is first translated into a query plan and then executed on the GIS database. The resulting map of the query is sent to the user as an MMS message to the user. Figure 4.10 shows the structure of IVQL environment and the process through which a smiley icon query undertakes in order to be executed.

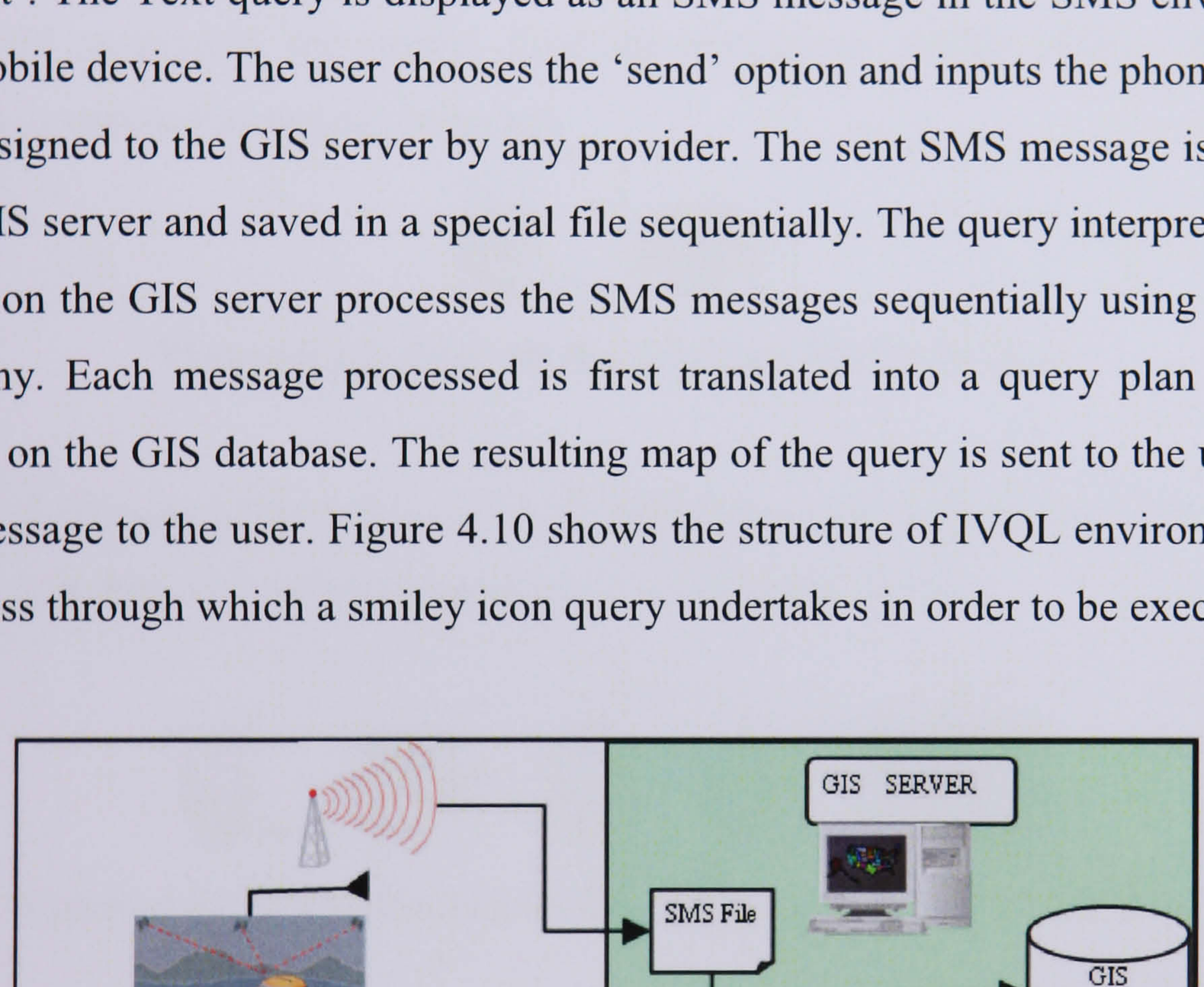


Figure 4.10: The Visual Query Processing.

The IVQL user interface is implemented for mobile phones by using the J2ME platform installed on the Symbian Operating System. The IVQL query interpreter is implemented on the server computer to query the spatio-temporal database such as the ArcGIS. IVQL is expected to have a high level of cost effectiveness, with potentially a large number of launched queries. Figure 4.11 represents the query: Find the nearest golf club and display the shortest path between my location and the golf club found.

Figure 4.11: Find Nearest Golf Club and Display the Shortest Path.

Figure 4.12 represents the query: Find all restaurants within 500m. This query expresses a sentence in natural language.

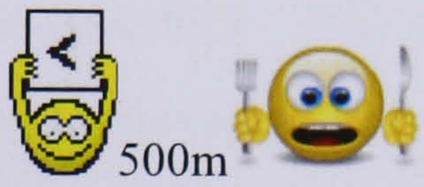


Figure 4.12: Find all Restaurants Within 500m.

Figure 4.13 represents the query: Find all universities and schools within 500m. This query is equivalent to a sentence in terms of information content.

Figure 4.13: Find all Universities and all Schools Within 500m.

Figure 4.14 represents the query: Find the shortest path from my location to a specific address *destination address*.

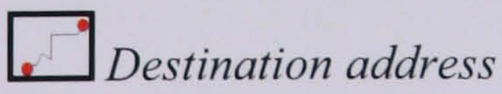


Figure 4.14: Find the Shortest Path to a Destination Address.

Figure 4.15 represents the query: Find all bus stations within 500m and all train stations within 700m. The iconic query has an expressive power.

Figure 4.15: Find Bus Stations Within 500m and Train Stations Within 700m.

4.5 The Evaluation of IVQL

The term usability is used during the design and implementation of a software system and its user interface, to reflect how much the system under development is easy to learn, effective to use, easy to remember how to use and provides an enjoyable experience. It forms the basis for evaluating the systems and provides a framework for this evaluation. Pressman in his book [Pre05] defines usability as a measure of how well a computer system facilitates learning, enables them to be efficient and makes them satisfied with the system. Moreover, usability is a major aspect of human-computer interaction (HCI) that is concerned with the design, evaluation and implementation of interactive computing systems for human use. HCI has moved beyond designing user interfaces to support all human activities and facilitate the user experience through designing user interfaces and systems that would make his work efficient, improve his learning, provide him with enjoyable and exciting entertainment, enhance his communication and understanding, and support him with new forms of creativity and expression.

With the increase of the use of communication technology, more application areas, more technologies and more issues are considered when designing user interfaces and computing systems. Hence, to insure that a system or a user interface is best designed and developed, the user has been involved in usability testing as a feedback through the iteration of the following cycles 'design-test-redesign' which are major phases of the design process model. The iterative design is a continuous process that examines prototypes of new systems to check that the designers understand the user's requirements. In order to achieve this model easily, two processes are being used, the first is to build interactive prototypes that can be easily communicated and assessed, and the second one is to evaluate what is being built throughout the process. A prototype is a small scale model or a piece of software with limited functionality and capability written in the target language and used for system evaluation and as a feedback in the design phase. Designers are supported in choosing between alternatives based on the user's opinion.

The aspects that are usually considered in any regular query language evaluation include the query writing as implemented in [Smi05] and the ease of use as implemented in both [mur00] and [Mor04]. Whereas, to evaluate a visual query language, more aspects are to be included such as the user interface, query building process, expressive power of the visual language, the proper representation of the icons used, the icons recognition, interpretation, comprehension and memorization, and the ease of use as implemented in [Bon02]. Before performing an evaluation, the proper form of evaluation strategy should be chosen as it normally dictates the form of the results that will be obtained. There are two main types of evaluation. The first type is the formative evaluation that is done at different phases of the development to check that the system meets the user's needs. The second type is the summative evaluation that is done once after the development of the system in order to assess the quality of the finished product. Both types of evaluation rely on a combination of techniques such as user's observation, interviews, questionnaires and user testing. Two paradigms are actually mostly used to implement the usability testing, the first one is the user testing and the second one is the user satisfaction [Kha04, Pre05].

In the user testing paradigm, the user's performance is tested in order to check that the system is usable for the tasks that it has been developed to achieve. It is also used to evaluate how well tasks are performed or to compare a prototype to an existing product. The focus is on the number of correct or wrong answers given by the user as well as the time taken by the user to complete a certain task. Then, a thorough study is done to examine the type of errors encountered while the user was completing the task in order to explain and elaborate why the errors took place. The user testing provides quantitative data on the user's performance. In the user satisfaction paradigm, the user answers a questionnaire made up of questions that reveal his opinion about a certain aspect of the system. The questions could be of many types such as Yes/No questions, check-boxes for many opinions, Likert rating scale, or open-ended responses. The user's satisfaction is used to evaluate the user interface of the system, the ease of formulating a visual query, the clarity of the icons and the system information, the ease of learning and memorization, the system capabilities, the consistency of the language constructs and the level of difficulty of the visual query language. The user satisfaction provides qualitative data on the user's opinion.

The data analysis is performed on both the quantitative and the qualitative results once the results of the usability testing are recorded. The results of the user testing may be presented in the form of summative results to show the time taken by subjects to complete a certain task, the number of errors made per task, the type of errors made per task, the number of error made per unit of time, number of users who made a particular error, and the number of users who completed a task successfully. The results of the questionnaires may be presented clearly using many forms such as tables, percentages, graphs, and simple statistics including the mean, the mode, and the standard deviation. The advantage of the evaluation that is implemented in this work over the evaluations implemented in the review of literature is that all the aspects of usability of IVQL are evaluated whereas in each of the other reviewed evaluations one aspect of usability only is evaluated such as query writing in [Smi05], user interface in [Mor04], and expressiveness of icons and language in [Bon02].

4.5.1 Method of Evaluation

To evaluate IVQL, its user interface, the visual query formulation, the expressive power of the language and its ease of use, it is necessary:

1- To identify the types of tasks to be performed by the user such as icons recognition and representation (expressive power of icons) , the ease of use, the user interface, the query building and formulation process, and the expressive power of the visual query language.

2- To list the aspects that will be measured such as user testing and user satisfaction, and how they are going to be measured such as using scoring and Likert scale.

3- To implement the evaluation and describe its phases such as the subjects and the experiment.

4- To analyze the results.

4.5.1.1 Icons Recognition and Representation (Expressive Power of Icons)

To evaluate the expressive power of icons or metaphors, many attributes are used such as icon recognition, ease of use of icons, query formulation constructs, interpretability, comprehension, memory and preference. The attributes icon recognition and representation have been chosen to evaluate the icons that are being used in IVQL since they have proven to reflect the clarity of icons as shown in LVIS [Bon02]. The user testing and the user satisfaction have been both chosen to evaluate the expressive power of the smiley icons. In the user testing method, the user is provided with a list of 30 smiley icons that are supposed to be evaluated. Next to each smiley icon, the user is also provided with an empty rectangle in which he is supposed to write the description that he thinks is best represented by the smiley icon. In the user satisfaction method, the user is provided with a questionnaire made up of 5 Likert scale questions through which he is supposed to reflect his level of satisfaction regarding some attributes of the icons such as the use of icons throughout the system, the operator icons relation to the tasks that they are intended for, the position of the icons on the screen, reading the icons and the icons representing the error messages.

4.5.1.2 Ease of Use

The main purpose of evaluating the ease of use is to reflect and measure how easy it is for the user to write, read, interpret, comprehend, and memorize queries as well as his ability to solve problems related to query formulation. Table 4.1, taken from Reisner in [Rei81], shows a list of some of the most important tasks that can be used to measure the ease of use of a visual query language. The query writing task was chosen for use in the evaluation of IVQL, similarly to the work done in [Smi04, Smith05], as it reflects the most clearly how well and easy it is to learn the visual query language. It also provides the proper data about how proficient a user is at building and formulating the visual queries. The user testing has been chosen to evaluate the ease of use of IVQL similarly to the work done in [Bon00, Bon01, and Bon02]. In this method, the user is given a list of 15 questions written in English and a table that contains a list of the 40 smiley icons that are used to formulate the queries.

Each icon has been uniquely numbered sequentially. Next to each question, the user is also provided with an empty rectangle in which he is supposed to write the list of the icon numbers that he wants to use in order to formulate his answer query. The questions have been prepared in a way to cover all levels of query formulation difficulty levels namely, the simple queries and the complex queries that include the 'and' operator to combine two or more simple queries.

Task	**Description**
Query writing	Users are given a question stated in natural language and are required to write a query in the given query language.
Query reading	Users are given a query in the query language and are asked to write a translation into the natural language.
Query interpretation	Users are given a query in the query language and a printed database with the data filled in. They are asked to find the data asked for by the query.
Question comprehension	Users are given a natural language question and a printed database and are asked to find the data asked for.
Memorization	Users are asked to memorize and reproduce a database.
Problem solving	Users are given a problem and a database and are asked to generate questions in natural language that would solve the problem. The questions should be answerable from the database.

Table 4.1: Tasks Used to Measure the Ease of Use of Query Languages.

4.5.1.3 User Interface

The main objectives of evaluating a user interface are first to reflect and identify the best smiley icons on which to base the design, second to check and ensure that the final user interface is consistent, and finally to improve the usability of the system under development. The user satisfaction method has been chosen to evaluate the user interface of IVQL. It is important to note here that none of the reviewed visual query languages used the user satisfaction in their evaluations of the usability aspects. Thus, it can be considered as an advantage of the IVQL evaluation over the others. In IVQL user satisfaction evaluation, the user is provided with a questionnaire made up of 7 Likert scale questions through which he is supposed to reflect his level of satisfaction regarding the difficulty of reading the smiley icons on the screen, how well the information is organized on the screen, whether the sequence of screens is confusing or not, the position of messages, the error messages, and whether the design aspects are suitable for all levels of users or not.

4.5.1.4 Query Building and Formulation Process

The main purpose of evaluating the query building and formulation process of IVQL is to reflect and measure how easy it is for the user to build and formulate queries using the smiley iconic visual query language. The selection of icons in the proper order and the process of building the queries both have been chosen for use in the evaluation of IVQL as they both provide the proper data about how proficient a user is at building and formulating the visual queries. The user satisfaction has been chosen to evaluate the query building and formulation process of IVQL. In this method, the user is provided with a questionnaire made up of 9 Likert scale questions through which he is supposed to reflect his level of satisfaction regarding the query building and formulation such as the selection of the operators, the selection of the categories, the selection of objects, building simple queries, the clarity of queries, the legibility of queries, remembering the language queries and its constructs, the sequence used to build the queries, and using the 'and' operator to add more queries.

4.5.1.5 Expressive Power of the Visual Query Language

The main purpose of evaluating the expressive power of the visual query language of IVQL is to reflect and measure how easy it is for the user to learn to operate the system and to perform tasks in a straightforward way in order to build and formulate visual queries. The ease of learning process and the visual language memorization aspects have been chosen for use in the evaluation of IVQL as they both provide the proper data about how powerful the visual query language is in expressing queries. The user satisfaction has been chosen to evaluate the query building and formulation process of IVQL. In this method, the user is provided with a questionnaire made up of 4 Likert scale questions through which he is supposed to reflect his level of satisfaction regarding the learning of the system and performing query building and formulation such as learning to operate the system, exploring new features by trial and error, remembering names and use of icons and operators, and performing tasks in a straightforward manner.

4.5.2 The Questions

Figure 4.16 shows that Part 1 of the test questions consists of a list of thirty icons that the subject is supposed to recognize and interpret. In the space provided to the right of each icon, he is supposed to write the name, place, or object that he thinks is best represented by the icon. This part is aimed at testing the user performance and hence leading to the measurement of the expressive power of the icons and their recognition. This aspect has been measured by scoring and counting the correct answers of the subject. All of the three types of icons were included namely operators, categories, and objects.

PART 1: In the space provided below, write the name (iconic representation) of what you think each icon represents.

Table 1

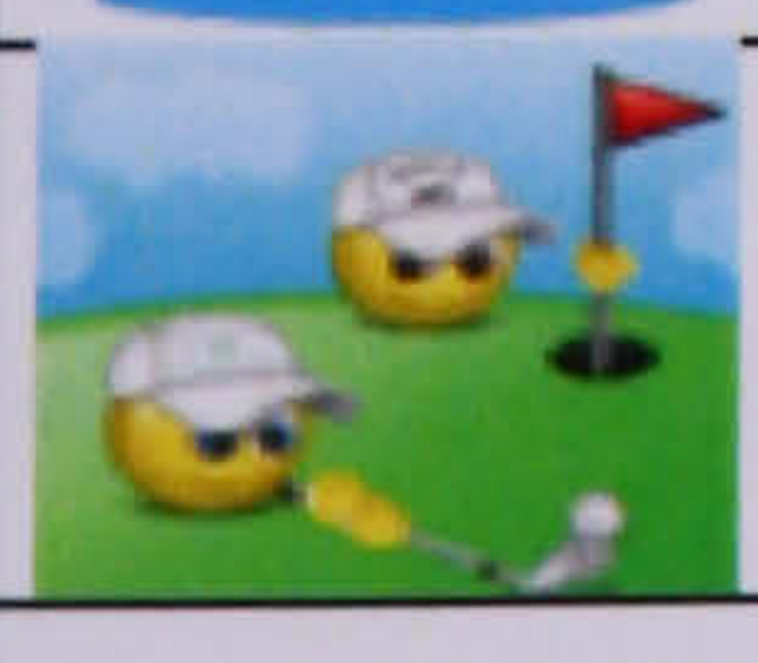
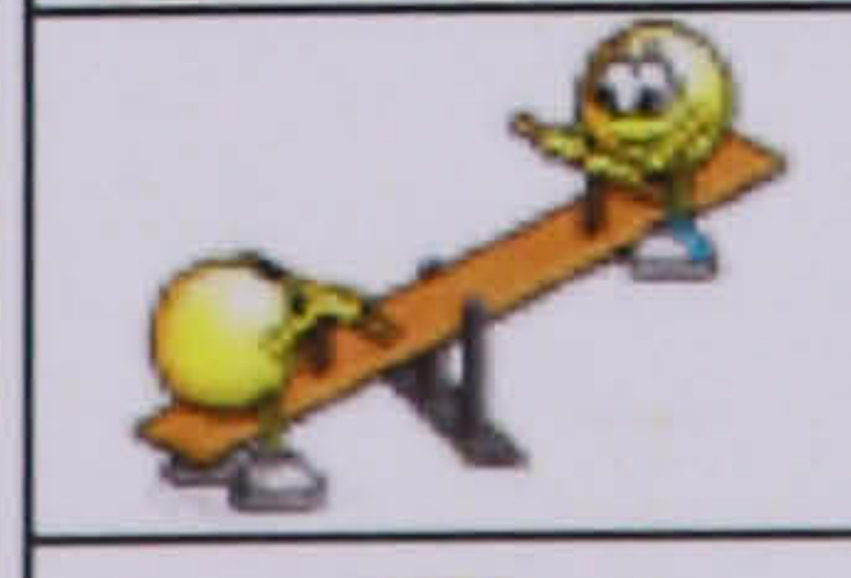
Icon	Name	Icon	Name
			
			
			
			
			
			
			
			
			
			
			
			
			
			
			

Figure 4.16: Part 1 of the Questions.

Table 4.2 shows that Part 2 of the test questions are mainly written in order to measure the ease of use of the visual query language through measuring the visual query writing task of the subjects. User testing is being used for this purpose. The answers of the subjects are scored and the number of correct answers is counted. Also, the errors in the answers of the subjects are recorded in order to analyze the type of errors committed and figure out the reason behind committing these errors. This part of questions consists of a list of fifteen queries questions written in English to be rewritten using the visual query language IVQL prototype. The subjects are also provided with a table containing forty smiley icons in its cells as shown in Table 4.3. Each icon has a unique identifier that will be used by the subjects to refer to the icons while answering the visual query writing. The answer of each question consists of a list of the icon identifiers that are used to build the visual query. These questions have been designed in a way to cover various levels of difficulty in visually formulating a query varying from simple to complex. Questions 1 to 8 are the simple queries questions and 9 to 15 the complex queries, noting that a query is considered complex if it includes the 'and' operator to combine two or more simple queries.

<u>PART 2:</u> Write the "iconic visual queries" that are used to answer the following questions. Each answer should be made up of a list of icon numbers that are used to formulate the queries. Each icon has been assigned a number in Table 2.

Question	*Icon numbers that are used to formulate the query*
1. Find the nearest casino.	
2. Find the nearest hotel.	
3. Find the nearest restaurant.	
4. Find the nearest telephone cabinet.	
5. Find all the bus stations that are located within 600 meters from my current location.	
6. Find all the bars that are located within 200 meters from my current location.	
7. Find all the golf clubs that are located within 400 meters from my current location.	
8. Find all the toilets that are located within 100 meters from my current location.	
9. Find all the banks that are located within 300 meters from my current location and all the train stations that are located within 500 meters from my current location.	
10. Find all the tennis courts that are located within 300 meters from my current location and all the gymnasiums that are located within 500 meters from my current location.	
11. Find all the metro stations that are located within 300 meters from my current location and all the monuments that are located within 400 meters from my current location.	
12. Find all the post offices that are located within 300 meters from my current location and all the gas stations that are located within 500 meters from my current location.	
13. Find the nearest taxi and the nearest hospital.	
14. Find the nearest museum and all the restaurants that are located within 300 meters from my current location.	
15. Find all the parks, children grounds, zoos, and picnic areas that are within 800 meters from my current location.	

Table 4.2: Part 2 of the Questions.

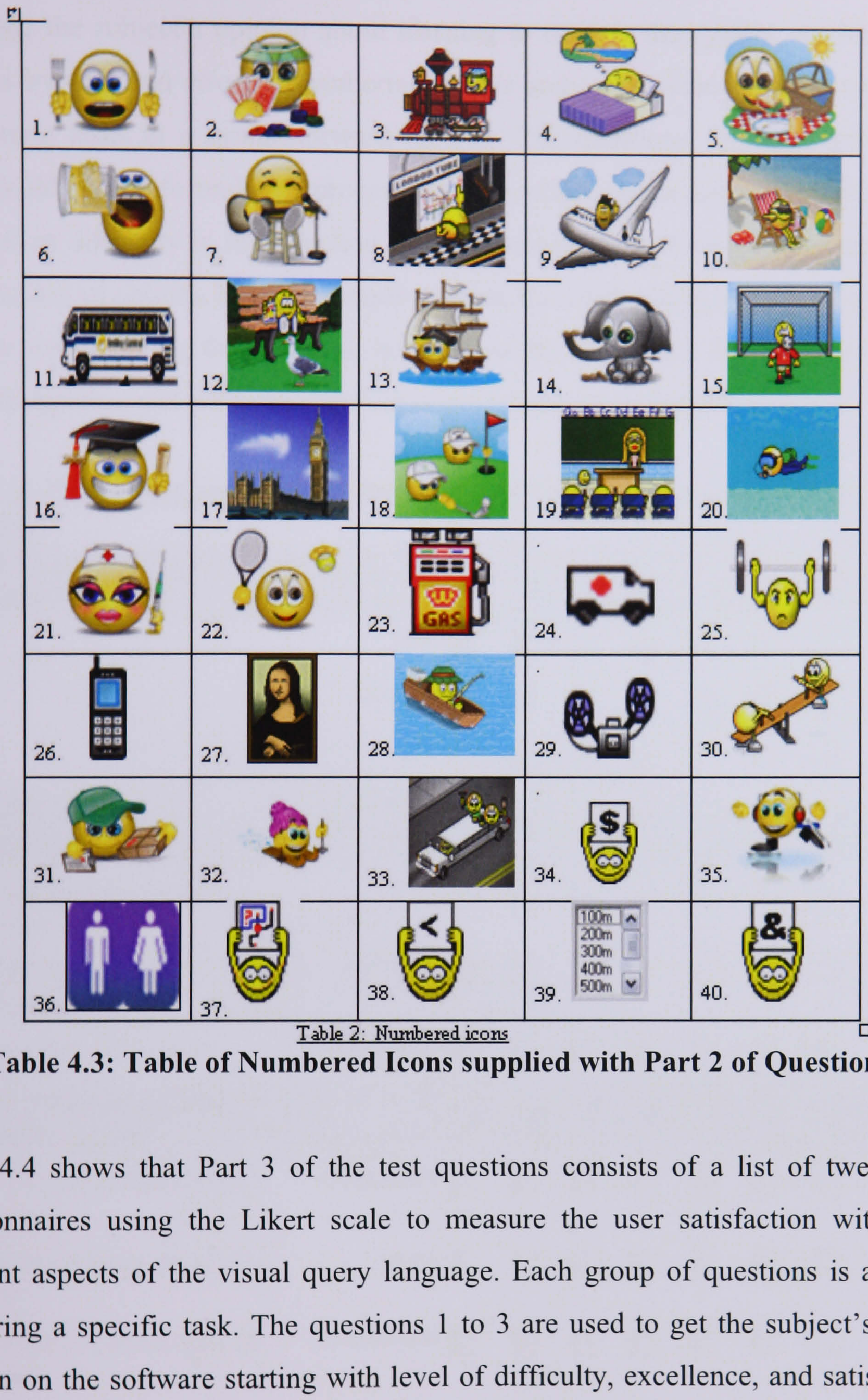


Table 2: Numbered icons

Table 4.3: Table of Numbered Icons supplied with Part 2 of Questions.

Table 4.4 shows that Part 3 of the test questions consists of a list of twenty five questionnaires using the Likert scale to measure the user satisfaction with many different aspects of the visual query language. Each group of questions is aimed at measuring a specific task. The questions 1 to 3 are used to get the subject's overall reaction on the software starting with level of difficulty, excellence, and satisfaction. The questions 4 to 6 are used to evaluate the user interface of the visual query language. They get the subject's opinion about the ease of reading icons on the screen, clarity in organizing the information, and the clarity in the sequence of screens. The questions 7 to 10 measure the consistency of the use of icons throughout the system, the relation between the operator icons and the tasks that they are supposed to perform, and the consistency of the messages that are displayed on the screen. The questions 11 to 14 measure the expressive power of the visual query language by

reflecting the subject's opinion about learning to operate the system, exploring new features by trial and error, remembering names and use of icons and operators, and performing tasks in a straightforward manner. The questions 15 to 25 measure the query building and formulation process through reflecting the subject's opinion about the level of difficulty in the selection of the operators, the selection of the categories, the selection of objects, building simple queries, the clarity of queries, the legibility of queries, remembering the language queries and its constructs, the sequence used to build the queries, and using the 'and' operator to add more queries.

.

PART 3: Please rate your satisfaction with the User Interface and Query building.

		1	2	3	4	5	6	7	
OVERALL REACTION TO THE SOFTWARE									
1.	poor	◉	☐	☐	☐	☐	☐	☐	excellent
2.	difficult	☐	☐	☐	☐	☐	☐	☐	easy
3.	not satisfying	☐	☐	☐	☐	☐	☐	☐	satisfying
SCREEN (USER INTERFACE)		**1**	**2**	**3**	**4**	**5**	**6**	**7**	
4. Reading icons on the screen	hard	☐	☐	☐	☐	☐	☐	☐	easy
5. Organization of information	confusing	☐	☐	☐	☐	☐	☐	☐	very clear
6. Sequence of screens	confusing	☐	☐	☐	☐	☐	☐	☐	very clear
ICONS AND SYSTEM INFORMATION		**1**	**2**	**3**	**4**	**5**	**6**	**7**	
7. Use of icons throughout system	inconsistent	☐	☐	☐	☐	☐	☐	☐	consistent
8. Operator icons related to task	never	☐	☐	☐	☐	☐	☐	☐	always
9. Position of messages on screen	inconsistent	☐	☐	☐	☐	☐	☐	☐	consistent
10. Error messages	unhelpful	☐	☐	☐	☐	☐	☐	☐	helpful
LEARNING		**1**	**2**	**3**	**4**	**5**	**6**	**7**	
11. Learning to operate the system	difficult	☐	☐	☐	☐	☐	☐	☐	easy
12. Exploring new features by trial and error	difficult	☐	☐	☐	☐	☐	☐	☐	easy
13. Remembering names and use of icons	difficult	☐	☐	☐	☐	☐	☐	☐	easy

14. Performing tasks is straightforward	never	☐	☐	☐	☐	☐	☐	☐	always
SYSTEM CAPABILITIES		**1**	**2**	**3**	**4**	**5**	**6**	**7**	
15. Correcting your mistakes	difficult	☐	☐	☐	☐	☐	☐	☐	easy
16. Design aspects are suitable for all levels of users	never	☐	☐	☐	☐	☐	☐	☐	always
QUERY BUILDING AND LANGUAGE CONSTRUCTS		**1**	**2**	**3**	**4**	**5**	**6**	**7**	
17. Selection of the operators	difficult	☐	☐	☐	☐	☐	☐	☐	easy
18. Selection of categories	difficult	☐	☐	☐	☐	☐	☐	☐	easy
19. Selection of objects	difficult	☐	☐	☐	☐	☐	☐	☐	easy
20. Building simple queries	difficult	☐	☐	☐	☐	☐	☐	☐	easy
21. Clarity of queries	unclear	☐	☐	☐	☐	☐	☐	☐	clear
22. Legibility of queries	illegible	☐	☐	☐	☐	☐	☐	☐	legible
23. Remembering the language queries	difficult	☐	☐	☐	☐	☐	☐	☐	easy
24. Sequence used to build the queries	difficult	☐	☐	☐	☐	☐	☐	☐	easy
25. Using & (AND) to add more queries	difficult	☐	☐	☐	☐	☐	☐	☐	easy
		1	**2**	**3**	**4**	**5**	**6**	**7**	

Table 4.4: Part 3 of Questions.

4.5.3 The Subjects

The subjects who have been selected to undertake the experiment are 56 undergraduate university students divided into two 28-subject groups classified as programmers and non-programmers. The programmers group is made up of those students who have a good background in computer science and are familiar with computers, programming languages, databases and query languages such as SQL, and geographic information systems. The non-programmers group is made up of those students who are from a non-computer science background, e.g., students still in their sophomore class coming from a variety of majors such as arts, science, business, etc, and who are currently taking their first introductory computer course. The age of all the subjects varies between 18 years and 21 years.

4.5.4 The Experiment

A pilot study, known as a pre-test session, is a small trial of the main testing process with the main purpose to make sure that the implementation of the evaluation testing is reliable. It informs the evaluators if the procedure can be conducted and that the questionnaires and test items work appropriately. A pilot study has been conducted with 6 non-programmer subjects in order to verify the evaluation technique, the test questions and the questionnaires. No difficulties were faced which proves that the tests are conductible and there will be no need to redo any evaluation testing due to unforeseen gaps. The form of the experiment used in the testing is the controlled experiment in which there are two subject groups, the first is the control group and the second is experimental group.

The main objective of having two different subject groups is the possibility to compare testing results analysis between the two groups on top of providing the ability to analyze the testing results of each subject group separately and independently. Johnson in [Joh98] states that the minimum number of subjects required to form a group in a controlled experiment is six and that the more subjects included in each of the groups the better and more considerable the testing results. As mentioned above, the number of subjects that was decided to include in each of the two subject groups is 28 making a total number of 56 subjects taking into consideration the aim of reaching better testing results.

The testing session of each subject group was conducted in a classroom equipped with 30 desktop computers making one computer available for each subject. A teacher's desktop computer connected to an LCD projector was also provided for the tester. The NetBeans version 5.0 Software was installed on all computers with the J2ME Mobility Pack and the Wireless Toolkit. The emulator DefaultColorPhone was used to emulate the prototype of the IVQL user interface on all the computers. Each session started with a small introduction of the purpose of the evaluation, a quick overview of the research under implementation, and a presentation of the IVQL user interface. A sample simple query formulation was demonstrated using the LCD projector as a training session. Each subject was then provided with the test questions and the questionnaires that were color photocopied. Some free time, between 5 to 10 minutes,

was given to allow the subjects to test and try on their own, using the prototype of IVQL and its query formulation. Then, their answers were written on the questions sheet after having used the IVQL prototype emulation for query building. Each session lasted around 2 hours.

4.6 Results and Discussion

The evaluation data have been collected from the answers provided by the subjects to each type of questions. Quantitative as well as qualitative data are being used to produce the necessary statistical results. Results are reported according to the three categories namely the smiley icons, the query formulation, and the user satisfaction questionnaire.

4.6.1 The Results of the Evaluation of the Smiley Icons

The answers of the subjects are graded based on a classification scheme that uses the letter C to represent the correct answers and the letter W to represent the wrong answers. An answer is considered correct when a subject has written the proper name of the object that an icon is supposed to represent. Close answers are considered correct even if a subject uses a different wording such as hotel, motel, etc. This qualitative scheme is used to produce the histogram charts that need nominal data. The same answers are also graded based on a quantitative scale in order to produce other statistical results. The correct answers are given the score 100 and the wrong ones 0. The data collected about the smiley icons are shown in Table 4.5 where each row corresponds to one subject and each column corresponds to one icon. The column entitled U contains the name of the university that the subject belongs to and the one entitled P contains either the letter P for Programmer or the letter N for Non-Programmer.

Answers of the 56 Subjects to the 30 Icons
C=Correct and W=Wrong

#	U	P	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30
1	A	P	C	C	C	C	W	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C
2	A	P	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C
3	A	P	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C
4	A	P	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	W	C	W	C	C	C	C	C	C	C	C	C	C	C
5	A	P	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C
6	A	P	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C
7	A	P	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C
8	A	P	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C
9	A	P	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C
10	A	P	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C
11	A	P	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C
12	A	P	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C
13	A	P	C	C	C	C	C	C	C	C	C	C	W	C	C	C	C	C	C	C	C	W	C	C	C	C	C	C	C	C	C	C
14	A	N	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C
15	H	P	C	C	C	C	W	C	C	C	C	C	C	C	W	W	C	C	W	C	C	C	C	C	C	C	C	C	C	C	C	C
16	H	P	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	W	C	C	C	C	C	C	C	C	C	W	C	C	C
17	H	P	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	W	C	W	C	C	C	C	C	C	C	C	C	C	C
18	H	P	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	W	C	C	C	C	C	C
19	H	P	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	W	C	C	C	C	C	C	C	C	C	C
20	H	P	C	C	W	C	C	C	W	C	C	C	C	C	C	C	C	C	W	C	C	C	C	C	C	C	C	C	C	C	C	C
21	H	P	C	C	C	W	C	C	C	C	C	C	C	C	W	C	C	W	C	C	C	C	W	C	W	C	C	C	C	C	W	C
22	H	P	C	C	C	C	C	C	C	C	C	C	W	C	C	W	C	C	W	C	C	W	C	C	C	C	C	C	C	C	C	C
23	H	P	C	C	C	W	C	C	C	C	C	C	W	C	C	C	C	C	W	C	C	W	C	C	C	C	C	W	C	C	C	C
24	H	N	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C
25	H	N	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C
26	H	N	C	C	C	W	C	C	C	C	C	C	C	C	C	C	C	C	W	C	C	C	C	C	C	C	C	C	C	C	C	C
27	H	N	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	W	C	C	C	C	C	C	C	C	C	C	C	C	C
28	H	N	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C
29	H	N	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C
30	H	N	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C
31	H	N	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	W	C	C	C	C	C	C	C	C	C	C
32	H	N	C	C	C	C	C	C	C	C	C	C	C	C	C	W	C	C	W	C	C	C	C	C	C	C	C	C	C	C	C	C
33	H	N	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	W	C
34	U	P	C	C	C	C	C	W	C	C	C	C	W	C	C	C	C	C	W	C	C	W	C	C	C	W	C	C	C	C	C	C
35	U	P	C	W	C	C	C	C	C	C	C	C	C	C	C	C	C	C	W	C	C	W	C	C	C	W	C	C	C	C	C	C
36	U	P	C	C	C	C	W	W	C	C	C	C	C	C	W	C	C	C	W	C	C	W	C	C	C	W	C	C	W	C	C	C
37	U	P	C	C	W	W	C	W	W	C	C	C	C	C	W	C	C	W	W	W	C	W	C	C	C	W	C	C	C	C	C	C
38	U	P	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C
39	U	P	C	C	C	C	C	C	C	C	C	W	C	C	C	C	C	C	C	C	W	C	C	C	W	C	C	C	C	C	C	C
40	U	N	C	W	C	C	C	C	C	C	C	C	C	C	C	C	C	C	W	C	C	W	C	C	C	W	C	C	C	C	C	C
41	U	N	C	C	C	W	C	W	C	C	C	C	C	C	C	C	C	C	C	C	C	W	C	C	W	C	C	C	C	C	C	C
42	U	N	C	C	C	C	W	C	W	C	C	C	C	C	C	C	C	C	W	C	C	W	C	C	C	W	C	C	C	C	C	C
43	U	N	C	C	C	C	C	W	C	C	C	C	C	C	C	C	C	W	W	C	C	W	C	C	C	W	C	C	C	C	C	C
44	U	N	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	W	C	C	C	W	C	C	C	C	C	C	C	C	C	C
45	U	N	C	W	C	W	C	W	C	C	C	C	C	W	C	W	C	W	W	C	C	W	C	C	W	W	C	C	C	W	C	W
46	U	N	W	W	W	W	W	W	W	C	C	C	C	W	C	W	C	C	W	W	C	W	W	C	C	W	C	C	C	W	C	W
47	U	N	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	W	W	C	W	C	W	C	C	C	C	C	C	C	C
48	U	N	C	W	C	C	C	C	C	C	C	C	C	C	W	C	C	C	C	C	C	W	C	C	C	C	C	C	C	C	C	C
49	U	N	C	C	C	C	C	W	C	C	C	C	C	C	C	W	C	W	W	C	C	W	C	C	C	C	C	C	W	C	C	C
50	U	N	C	W	C	C	C	W	W	C	C	C	C	C	C	C	C	C	C	C	C	W	C	C	W	W	C	C	C	C	C	W
51	U	N	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	W	C	C	C	C	C	C	C	C	C	C
52	U	N	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	W	C	C	C	W	C	C	C	C	C	W
53	U	N	C	W	C	W	C	C	C	C	C	C	C	C	C	W	C	W	W	C	C	W	C	C	W	W	C	C	C	C	C	C
54	U	N	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	W	C	C	C	C	C	C
55	U	N	C	C	C	C	W	W	C	C	C	C	W	C	W	C	C	C	W	C	C	W	C	C	C	C	C	C	W	C	C	W
56	U	N	C	C	C	C	W	W	C	C	C	C	C	C	C	C	C	W	C	C	C	W	C	C	C	W	C	C	C	C	C	C

Table 4.5: The Grades of the Subjects Answers to the Icon Recognition Questions

The collected data is represented visually using a graphical chart namely the histogram bar chart. The histogram in Figure 4.17 shows for each icon the total number of correct answers and its respective value as converted to a score over 100.

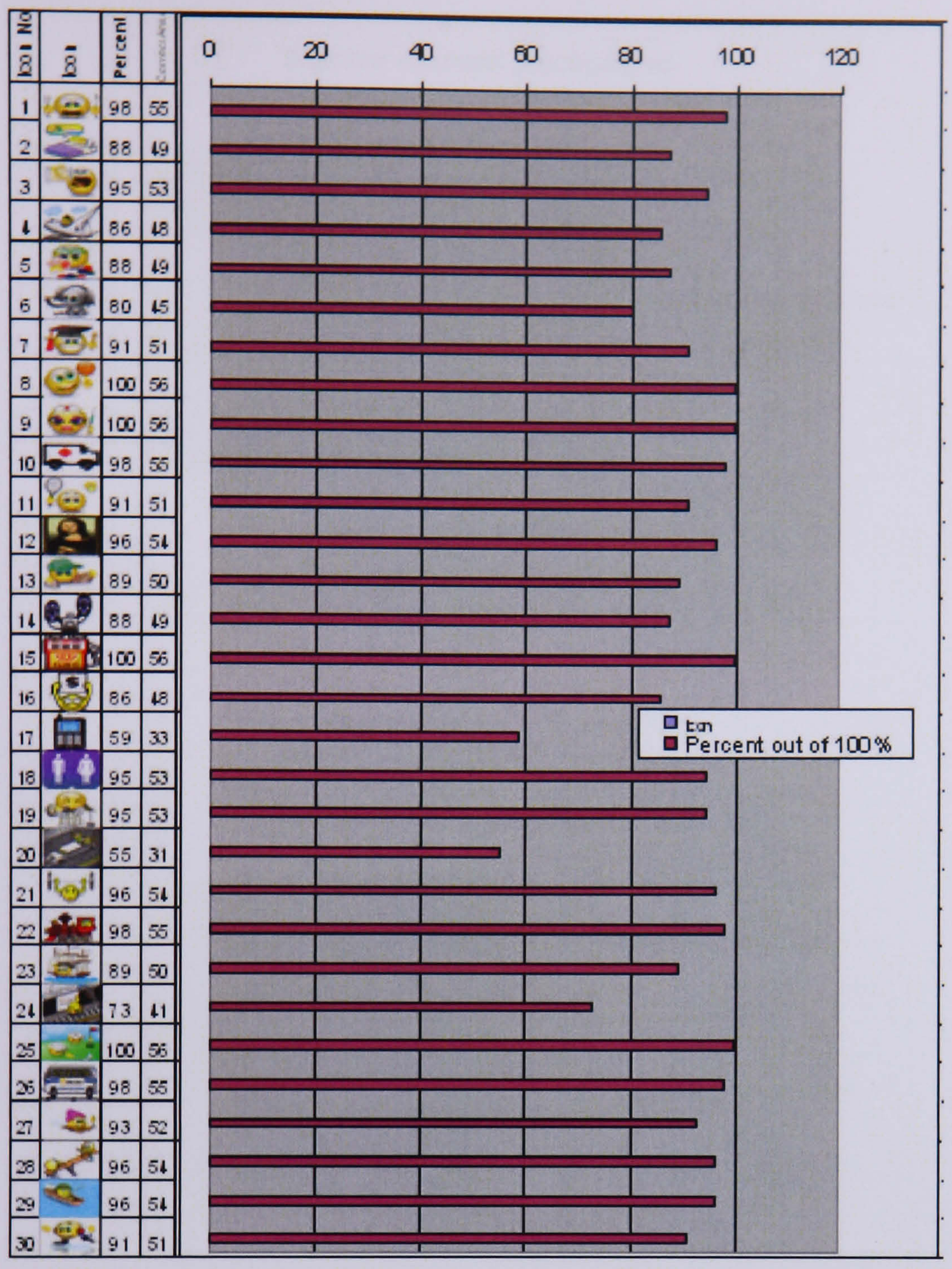
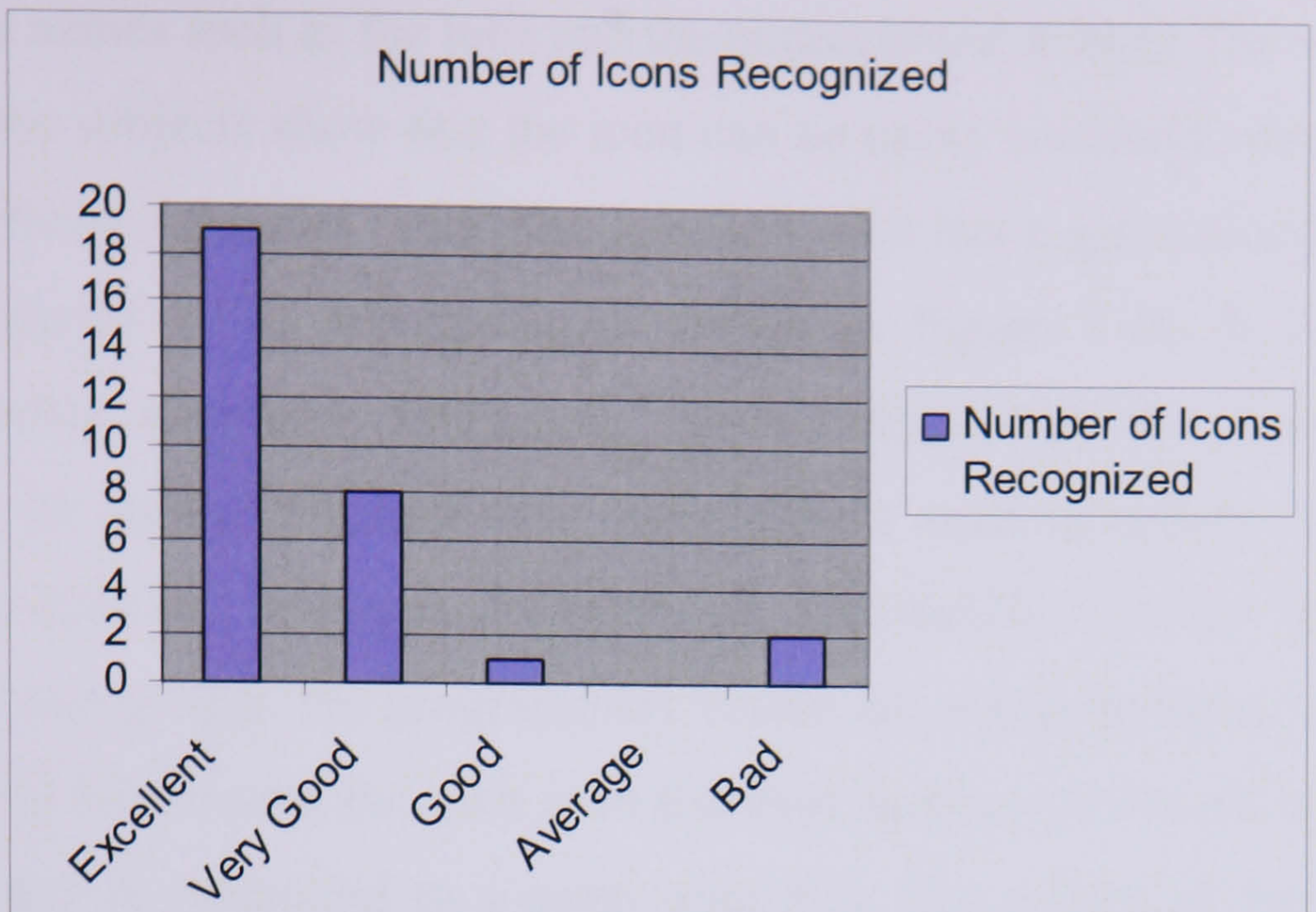


Figure 4.17: The Total Number of Correct Answers.

The icons that are recognized easily are the ones with the higher grades whereas the ones recognized with difficulty with the lower grades. Hence, the level of the ease of icon recognition can be divided into the following scale:

* Excellent recognition: a grade greater than or equal to 90
* Very Good recognition: a grade between 80 and 89
* Good recognition: a grade between 70 and 79
* Average recognition: a grade between 60 and 69
* Bad recognition: a grade below 60

Figure 4.18 shows the number and percentage of icons as distributed over the icon recognition levels.



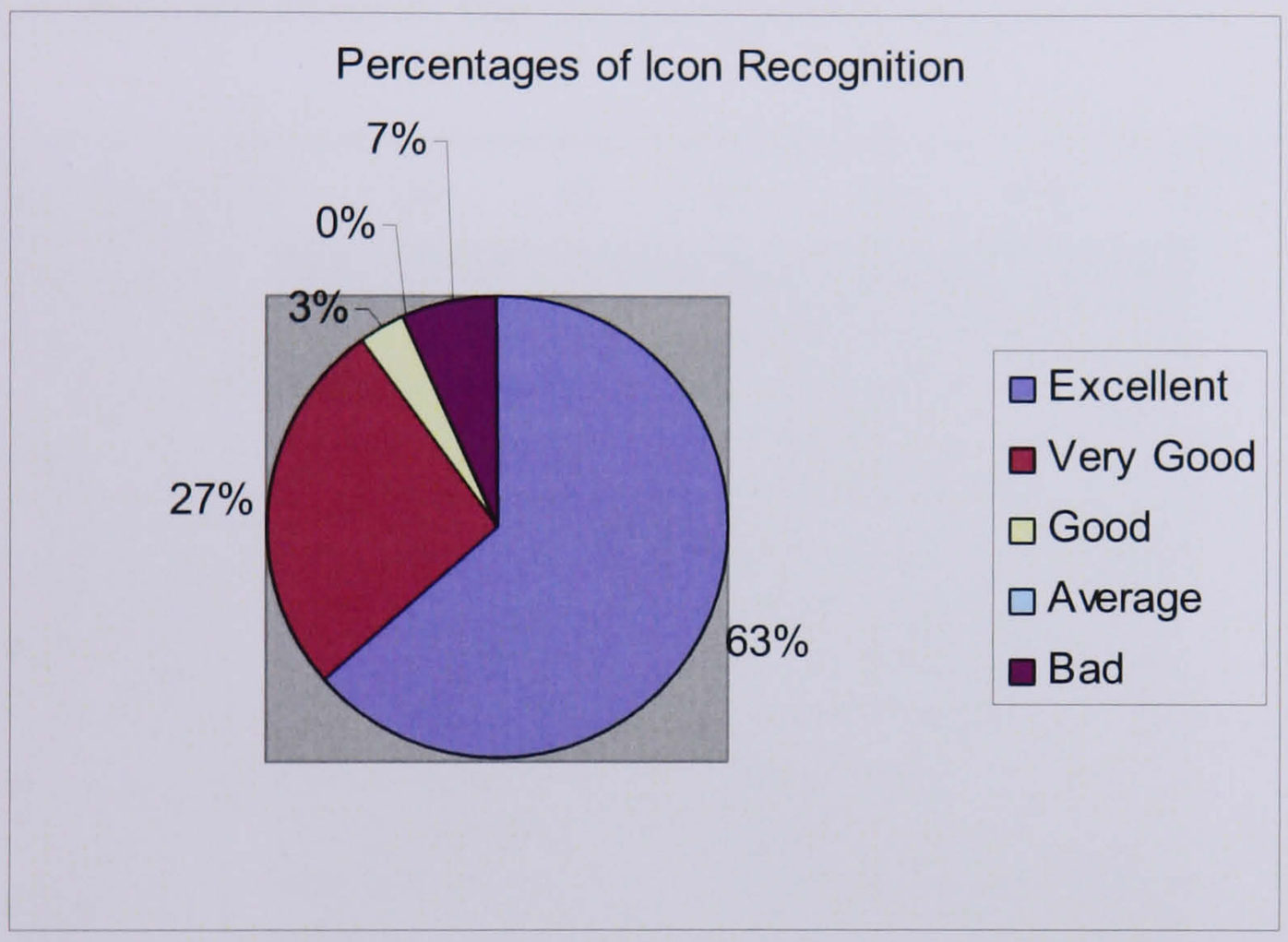


Figure 4.18: The Number and Percentages of Icon Recognition.

Nineteen icons representing 63% of the total icons have an excellent recognition, eight icons representing 27% have a very good recognition, one icon representing 3% has a good recognition and two icons representing 6.67% have a bad recognition. The total average of all the icons scores is 90/100. It can be concluded that in general the smiley icons are very easily recognized and can be used to visualize and represent objects, places, or locations. The two icons that are badly recognized are the ones numbered respectively 20 and 24 in the histogram shown in Figure 4.16. The first icon is supposed to represent a taxi company. The wrong answers supplied by the subjects show that the icon can be easily confused with other objects such as Limozine, amusement, celebration, delivery, and rent a Limozine. The second icon is supposed to represent a metro station. The close answers that are considered correct

include many names such as the tube and the underground station. The wrong answers supplied by the subjects show that the icon can be easily confused with other objects such as fast diesel, taxi, and train. The only icon that has a good recognition level is the one numbered 17 in the histogram shown in Figure 4.16. It is supposed to represent a public telephone. The wrong answers supplied by the subjects show that the icon can be easily confused with other objects such as mobile station, mobile store, mobile operator, operators, mobile shop, and mobile. In order to compare the scores of the two groups, the programmers versus the non-programmers, a histogram is used. Figure 4.19 shows for each icon the total number of correct answers and its respective value as converted to a score over 100. The results of the programmers group appear in blue and those of the non-programmers group in red.

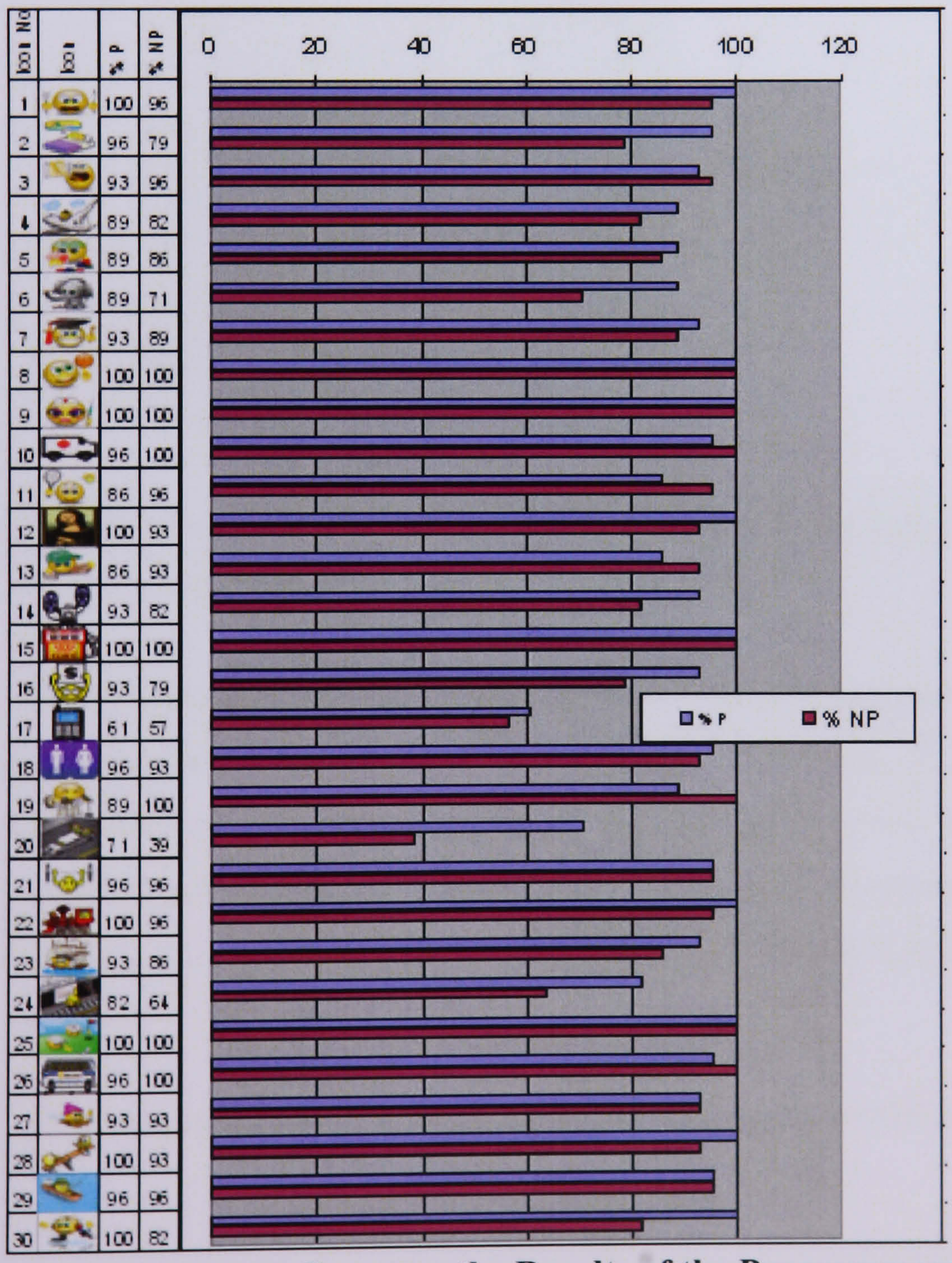


Figure 4.19: Histogram to Compare the Results of the Programmers Group versus the Non-Programmers Group.

The mean of the icons scores for each of the programmers group, non-programmers group, and both groups are respectively 93, 88, and 90 out of 100 as shown in Table 4.6 and Figure 4.20.

Average Means of Icons	Programmers (28)	Non-Programmers (28)	Both Groups (56)
Average of Correct Answers	26	25	51
Percentage out of 100	93	88	90

Table 4.6: The Means of the Programmers, Non-programmers, and Both Groups.

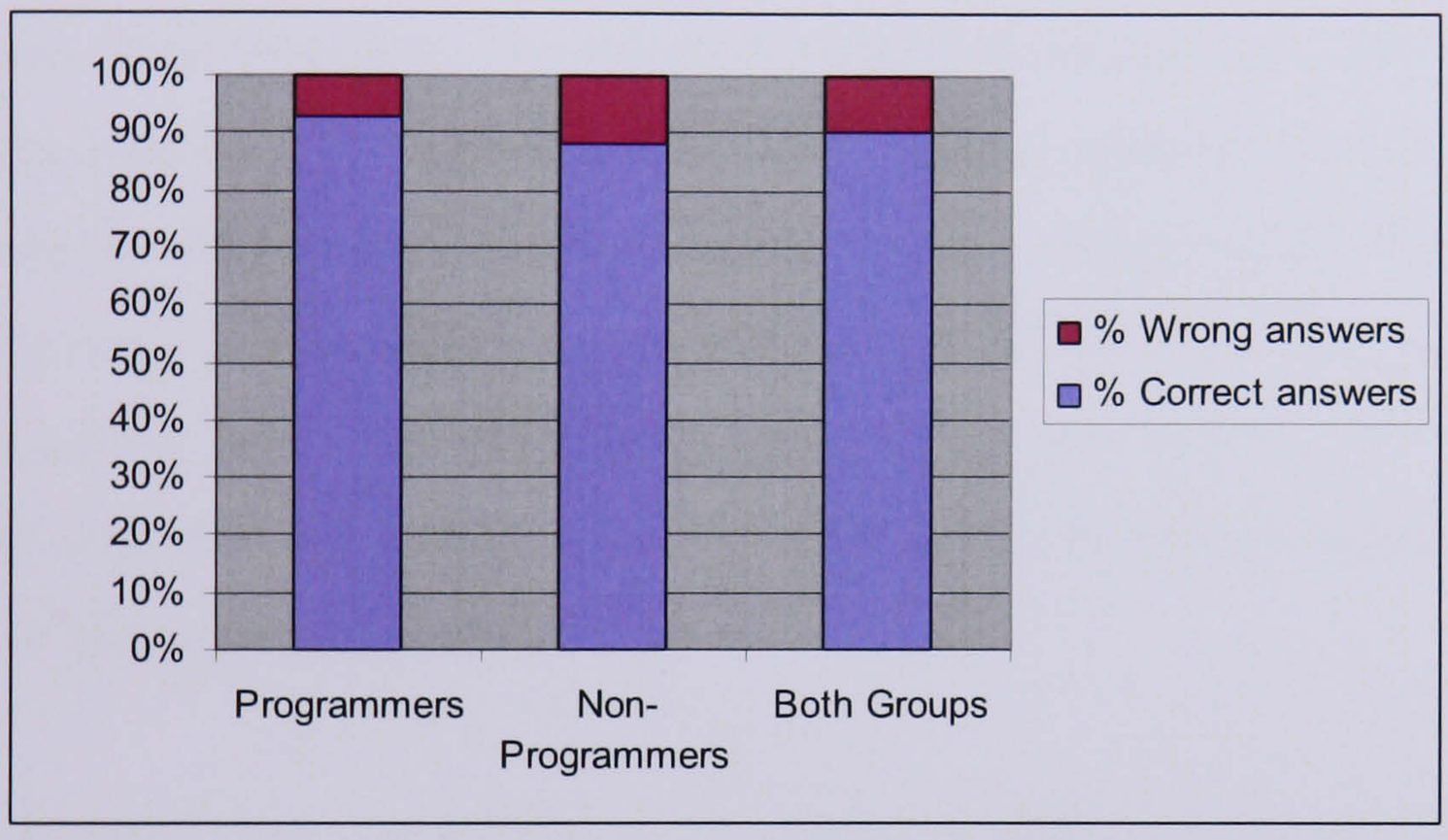


Figure 4.20: Percentage of Correct and Wrong Answers of the Programmers, Non-programmers, and Both Groups.

Hypothesis Testing [Blu03] is a procedure based on sample evidence and probability theory to determine whether the *hypothesis* is a reasonable statement. To test if there is a significant difference between the mean averages of the two groups, the programmers and the non-programmers, the *null hypothesis* H_0 is stated assuming that the means are equal. The alternate hypothesis H1 indicates that there is a difference:

$H_0 : \mu_1 = \mu_2$ Where μ_1 is the mean of the programmers and μ_2 the non-programmers
$H_1 : \mu_1 \neq \mu_2$

A *t-test* is used to calculate a *t-stat* value which determines whether the *null hypothesis* should be rejected or not. If the result shows that the means are different,

another *null hypothesis* H_0 is stated assuming that the mean of the programmers is less than or equal to the mean of the non-programmers. Another *t-stat* is calculated to determine whether the new null hypothesis should be rejected or not.

$H_0 : \mu_1 \leq \mu_2$
$H_1 : \mu_1 > \mu_2$ Where μ_1 is the mean of the programmers and μ_2 the non-programmers

The *t*-test is derived directly from the standard method used to estimate significance of a deviation from the mean in a student-t distribution, and is intended for use where the number of samples is small and the number of subject in at least one group is less than thirty. The *independent samples t-test* is used to compare one aspect between two subject groups. From the t-test, two values are obtained namely the *t* and the *p* values. The *t* statistic measures the difference between the two variables means, with the *p* value defining the significance of the *t* statistic. If the *p* value is 0.05, then given that the null hypothesis is true there is a 5% chance that the observed result – or a result more extreme than this – would occur by random chance. Values of *p* less than or equal to 0.05 indicate a significant result when the word significant means outside the 95% range of the null hypothesis.

When the samples are considered small, both of their variances unknown but presumed equal, then the appropriate test statistic is the student t statistic. The t statistic is defined as follows:

$$t = \frac{\overline{X_1} - \overline{X_2}}{S_{x_1 x_2}\sqrt{\frac{1}{n_1} + \frac{1}{n_2}}} \; where \begin{cases} \overline{X_1} is\_the\_mean\_of\_first\_group \\ \overline{X_2} is\_the\_mean\_of\_second\_group \\ n_1 is\_the\_number\_of\_participants\_in\_first\_group \\ n_2 is\_the\_number\_of\_participants\_in\_second\_group \\ S_{x_1 x_2} = \sqrt{\frac{S^2{}_{x_1} + S^2{}_{x_2}}{n_1 + n_2 - 2}} \\ S_{x_1} is\_the\_standard\_deviation\_of\_first\_group \\ S_{x_2} is\_the\_standard\_deviation\_of\_second\_group \end{cases}$$

Since the number of participants in each group is 28 ($n_1 = n_2 = 28$) both n_1 and n_2 could be replaced by $n = 28$. A simple substitution of n in the formula results in producing the following equivalent formula:

$$t = \frac{\overline{X_1} - \overline{X_2}}{S_{x_1x_2}\sqrt{\frac{2}{n}}} \; where \begin{cases} \overline{X_1} is\_the\_mean\_of\_first\_group \\ \overline{X_2} is\_the\_mean\_of\_second\_group \\ n\_is\_the\_number\_of\_participants\_in\_each\_group \\ S_{x_1x_2} = \sqrt{\frac{S^2{}_{x_1} + S^2{}_{x_2}}{2n-2}} \\ S_{x_1} is\_the\_standard\_deviation\_of\_first\_group \\ S_{x_2} is\_the\_standard\_deviation\_of\_second\_group \end{cases}$$

The *t*-test of the current work has been calculated using the Microsoft Excel 2003 Data Analysis Tools to compare the mean scores of the two groups with respect to each icon separately where the following values are considered:

* the confidence level is 95% ,
* the significance level is 5%,
* alpha α is equal to 0.05,
* the degrees of freedom *df* is 54 which is the number of subjects less the number of groups,
* *t* critical for one-tail is 1.673565, and
* *t* critical for two-tail is 2.004879.

The results of the statistics are shown in Table 4.7.

Icon #	P mean score	NP mean score	t Statistics	p-value1tail	p-value2tails
1	100.00	96.43	1.000	0.161	0.322
2	96.43	78.57	2.060	0.022	0.044
3	92.86	96.43	-0.585	0.281	0.561
4	89.29	82.14	0.754	0.227	0.454
5	89.29	85.71	0.397	0.346	0.693
6	89.29	71.43	1.695	0.048	0.096
7	92.86	89.29	0.461	0.323	0.647
8	100.00	100.00			
9	100.00	100.00			
10	96.43	100.00	-1.000	0.161	0.322
11	85.71	96.43	-1.406	0.083	0.166
12	100.00	92.86	1.441	0.078	0.155
13	85.71	92.86	-0.854	0.198	0.397
14	92.86	82.14	1.206	0.116	0.233
15	100.00	100.00			
16	92.86	78.57	1.532	0.066	0.131
17	60.71	57.14	0.267	0.395	0.791
18	96.43	92.86	0.585	0.281	0.561
19	89.29	100.00	-1.800	0.039	0.077
20	71.43	39.29	2.510	0.008	0.015
21	96.43	96.43	0.000	0.500	1.000
22	100.00	96.43	1.000	0.161	0.322
23	92.86	85.71	0.854	0.198	0.397
24	82.14	64.29	1.513	0.068	0.136
25	100.00	100.00			
26	96.43	100.00	-1.000	0.161	0.322
27	92.86	92.86	0.000	0.500	1.000
28	100.00	92.86	1.441	0.078	0.155
29	96.43	96.43	0.000	0.500	1.000
30	100.00	82.14	2.423	0.009	0.019

Table 4.7: The *t* Statistics of Each Icon Used for the Significant Differences of the Means.

The *t statistic* of each icon is compared to the *t critical for two-tail* which is 2.004879. If the *t statistic* is less than 2.004879, it can be concluded that there is no significant difference between the means of the programmers sample group and the non-programmers. For example, the *t statistic* of the first icon is 1 which is less than 2.004879. Hence, it can be reported that there is no significant difference between the means of the programmers and the non-programmers with respect to recognizing the first icon. The green *t statistic* values in Table 4.3 show that twenty seven out of thirty icons were recognized at the same level by the subjects of both groups.

If the *t statistic* is greater than 2.004879, it can be concluded that there is a significant difference between the means of the programmers sample group and the non-programmers. In such a case, the *t statistic* is compared to the *t critical for one tail* which is 1.673565. If it is greater, it can be concluded that the programmers sample group performed better than the non-programmers, and vice-versa. For example, the *t statistic* of the second icon which is 2.060 is also greater than 1.673565. Hence, it can be reported that the programmers sample subject performed better than the non-programmers, with t=2.060 and p=0.022. The programmers are able to recognize the second icon at a higher level of recognition than the non-programmers. There is a 2.2% chance that this result occurred by random chance.

The yellow *t statistic* values in Table 4.3 show that only three icons out of thirty reflect a significance difference between the means of the sample groups. The first one is the icon number 2 (Hotel) with t=2.060 and p=0.022. The second one is icon number 20 (Taxi) with t=2.510 and p=0.008. The third one is icon number 30 (Ice Skating) with t=2.423 and p=0.009. In the three cases, it can be reported that at a 5% level of significance, the programmers were able to recognize the icons better than the non-programmers with a *p* value less than or equal to 0.022. An important question is raised here: do the meanings of these three icons offer any explanation as to why this might be? Their meanings probably do not offer any explanation. Moreover, if there is a 5% = $1/20$ chance of a "False Positive", in the 30 icons this should happen on the average $\frac{30}{20} = 1\,1/2$ times. The probability of it happening:

- never = $\frac{30!}{0! \times 30!} = (0.05)^0 (1-0.05)^{30} = 0.214$
- once = $\frac{30!}{1! \times 29!} = (0.05)^1 (1-0.05)^{29} = 0.339$
- twice = $\frac{30!}{2! \times 28!} = (0.05)^2 (1-0.05)^{38} = 0.299$

From the above probabilities, the following probabilities can be obtained. The probability of having:

- P(at least one "False Positive") = 1 – 0.214 = 78.6%
- P(at least two "False Positive") = 1 – 0.339 = 44.7%
- P(at least three "False Positive") = 1 – 0.299 = 19%

Hence, with respect to the likelihood of false-positives, it can be concluded that given the 95% criterion, one would expect 1.5 "False Positives" amongst the 30 independent icons (0.05 x 30). Since it was found that the probability of having at least three "False Positives" is 19%, there may not, therefore, be anything particularly significant about the icons 2 (Hotel), 20 (Taxi), and 30 (Ice Skating); they may simply be random anomalies.

The *t*-test has also been applied to compare the overall mean scores of the two groups with respect to all the icons together. The results show that the *t statistic* is equal to 1.503 and the *t critical* is 2.001717. Hence, there is no significant difference between the means. In general, it can be reported that the smiley icons are very easily recognized by both sample groups and that there is no significant mean difference between the programmers and the non-programmers sample groups with respect to the icons recognition. However, a thorough investigation of the results shows that there are 23 cases where a difference was observed. 17 cases showed the programmers ahead of the non-programmers and 6 cases only showed the programmers ahead of the non-programmers. It can be said that 74% ($^{17}/_{23}$) of the times when there was a difference, the programmers performed better than the non-programmers. Given the null-hypothesis there is a 50% chance that programmers perform better than the non-programmers and 50% the other way around. By using the Beta distribution, the following values can be calculated:

- Mean = $0.5 \times 23 = 11.5$
- Standard deviation = $\sqrt{23(0.5)(1-0.5)} = 2.39$
- Approximate Gaussian Z criterion for 95% confidence = $11.5 + (1.96 \times 2.39)$ = 16.18. So, it generally fails the 95% confidence.
- $z = \frac{17-11.5}{2.39} = 2.301$
- The value corresponding to z in the z-table is 0.489
- p-value = 0.5 – 0.489 = 1.1%

So, it can be said that based on the 1-tail test, only 1.1% of this result occurred by chance and that there is a significant (albeit small) bias towards the programmers, though not especially important. Hence, it can be concluded that there is some evidence for thinking that the programmers perform better than the non-programmers.

4.6.2 The Results of the Evaluation of the Query Formulation

The answers of the subjects are graded based on a classification scheme that uses the letter C to represent the correct answers and the letter W to represent the wrong answers. An answer is considered correct when a subject has written the correct sequence of the icon numbers that are used to formulate a query. The same answers are also graded based on a quantitative scale in order to produce statistical results. The correct answers are given the score 100 and the wrong ones 0. The data collected about the query formulation are shown in Table 4.8 where each row corresponds to one subject and each column corresponds to one query. As before, the column entitled U contains the name of the university that the subject belongs to and the one entitled P contains either the letter P for Programmer or the letter N for Non-Programmer.

Answers of the 56 Subjects to the 15 Queries
C=Correct and W=Wrong

#	U	P	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	A	P	100	100	100	100	100	100	100	100	100	100	100	100	0	0	0
2	A	P	100	100	100	100	100	100	100	100	100	100	100	100	100	100	0
3	A	P	100	100	100	100	100	100	100	100	100	100	100	100	100	0	100
4	A	P	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100
5	A	P	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100
6	A	P	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100
7	A	P	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100
8	A	P	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100
9	A	P	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100
10	A	P	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100
11	A	P	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100
12	A	P	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100
13	A	P	100	100	100	100	100	100	100	100	0	100	100	0	100	100	0
14	A	N	100	100	100	100	100	100	100	100	100	100	100	100	0	100	100
15	H	P	100	100	100	100	100	100	100	100	0	0	0	0	100	100	0
16	H	P	100	100	100	100	100	100	100	100	0	100	100	100	0	0	0
17	H	P	100	100	100	100	100	100	100	100	100	100	100	100	100	100	0
18	H	P	100	100	100	100	100	100	100	100	100	100	100	100	100	100	0
19	H	P	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100
20	H	P	100	100	100	100	100	100	100	100	0	0	0	0	0	0	0
21	H	P	100	100	100	100	0	0	0	0	100	100	100	100	0	0	0
22	H	P	100	100	100	100	100	100	100	100	0	100	0	0	0	100	0
23	H	P	100	100	100	100	100	100	100	100	100	0	0	100	100	0	0
24	H	N	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100
25	H	N	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100
26	H	N	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100
27	H	N	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100
28	H	N	100	100	100	100	100	100	100	100	100	100	100	100	100	0	0
29	H	N	100	100	100	100	100	100	100	100	100	100	100	100	100	100	0
30	H	N	100	100	100	100	100	100	100	100	100	100	100	100	0	0	0
31	H	N	100	100	100	100	100	100	100	100	100	100	100	100	0	0	0
32	H	N	100	100	100	100	100	100	100	100	100	100	100	100	100	0	0
33	H	N	100	100	100	100	100	100	100	100	0	0	0	0	0	0	0
34	U	P	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100
35	U	P	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100
36	U	P	100	100	100	100	100	0	100	100	0	0	100	100	100	100	0
37	U	P	100	100	100	0	100	0	100	0	0	100	0	0	0	100	0
38	U	P	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100
39	U	P	100	100	100	100	100	100	100	100	100	100	100	100	100	100	0
40	U	N	0	0	0	0	100	100	100	100	0	100	100	100	0	0	100
41	U	N	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100
42	U	N	100	100	100	100	100	100	100	100	100	100	0	0	100	100	0
43	U	N	100	100	100	100	100	100	100	100	100	100	0	100	0	100	0
44	U	N	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100
45	U	N	100	100	100	100	100	100	100	100	0	0	0	0	0	0	0
46	U	N	100	100	100	100	100	100	100	100	100	100	0	100	0	0	0
47	U	N	100	100	100	0	100	100	100	100	100	100	0	100	0	100	0
48	U	N	100	0	100	100	100	100	100	100	100	100	100	100	100	100	100
49	U	N	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100
50	U	N	100	100	100	100	100	100	100	100	100	100	0	100	100	100	0
51	U	N	100	100	100	100	100	100	100	100	100	100	100	100	100	100	0
52	U	N	100	100	100	100	0	0	0	0	0	0	0	0	0	0	0
53	U	N	100	100	100	100	100	100	100	100	100	100	100	100	100	100	0
54	U	N	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100
55	U	N	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100
56	U	N	100	100	100	100	100	100	100	100	100	100	100	0	100	100	0

Question
1. Find the nearest casino.
2. Find the nearest hotel.
3. Find the nearest restaurant.
4. Find the nearest telephone cabinet.
5. Find all the bus stations that are located within 600 meters from my current location.
6. Find all the bars that are located within 200 meters from my current location.
7. Find all the golf clubs that are located within 400 meters from my current location.
8. Find all the toilets that are located within 100 meters from my current location.
9. Find all the banks that are located within 300 meters from my current location and all the train stations that are located within 500 meters from my current location.
10. Find all the tennis courts that are located within 300 meters from my current location and all the gymnasiums that are located within 500 meters from my current location.
11. Find all the metro stations that are located within 300 meters from my current location and all the monuments that are located within 400 meters from my current location.
12. Find all the post offices that are located within 300 meters from my current location and all the gas stations that are located within 500 meters from my current location.
13. Find the nearest taxi and the nearest hospital.
14. Find the nearest museum and all the restaurants that are located within 300 meters from my current location.
15. Find all the parks, children grounds, zoos, and picnic areas that are within 800 meters from my current location.

Table 4.8: The Grades of the Subjects Answers to Query Formulation Questions.

The collected data is represented visually using a graphical chart namely the histogram bar chart. The histogram in Figure 4.21 shows for each query the total number of correct answers and its respective value as converted to a score over 100.

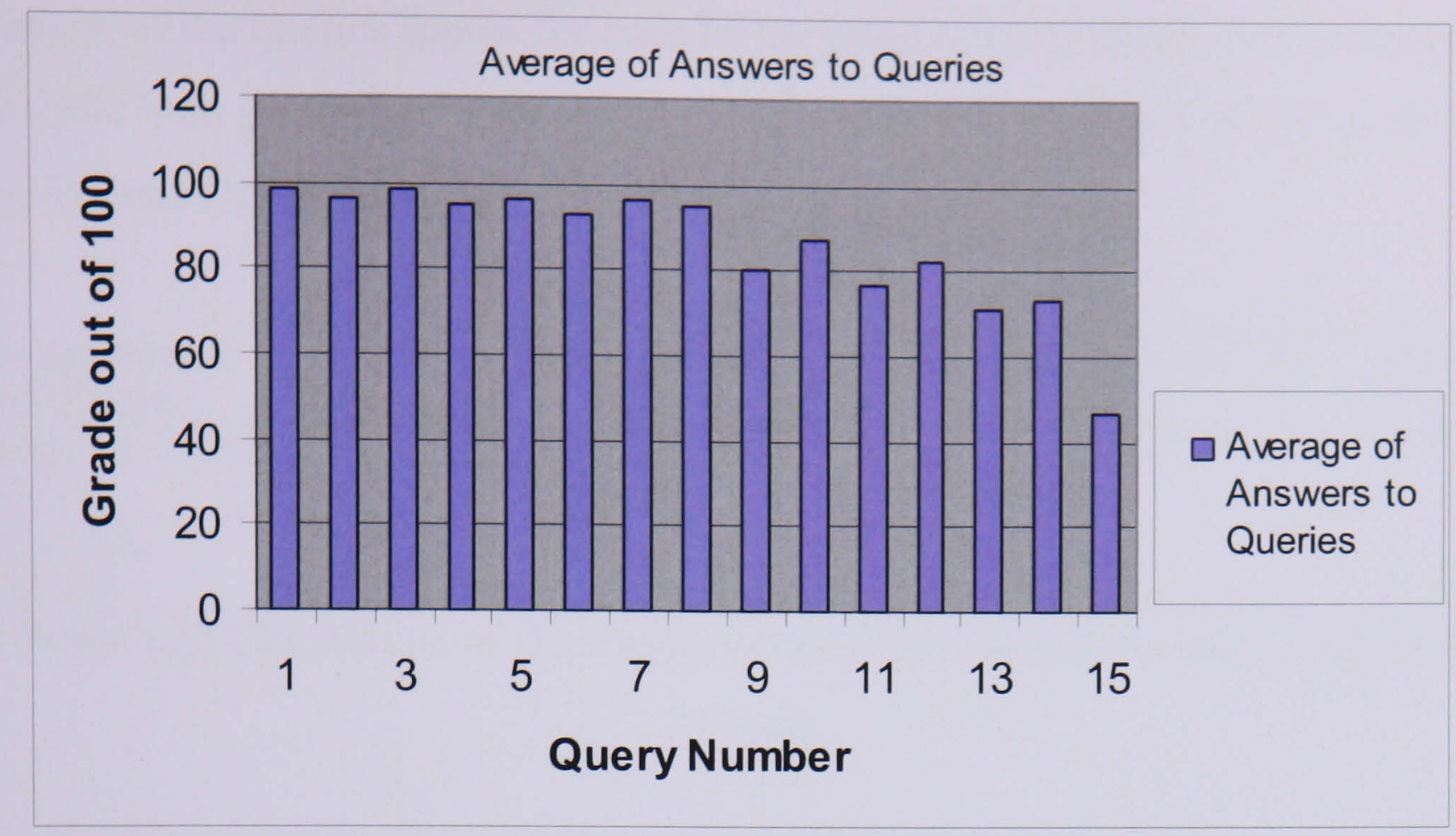


Figure 4.21: The Average of Correct Answers to Query Formulation.

The total average of all the queries scores is 85.71 out of 100. It can be concluded that in general the visual queries are very easily formulated. In order to compare the queries scores of the two groups, the programmers versus the non-programmers, a histogram is used. Figure 4.22 shows for each query the total number of correct answers and its respective value as converted to a score over 100. The results of the programmers group appear in blue and those of the non-programmers group in red.

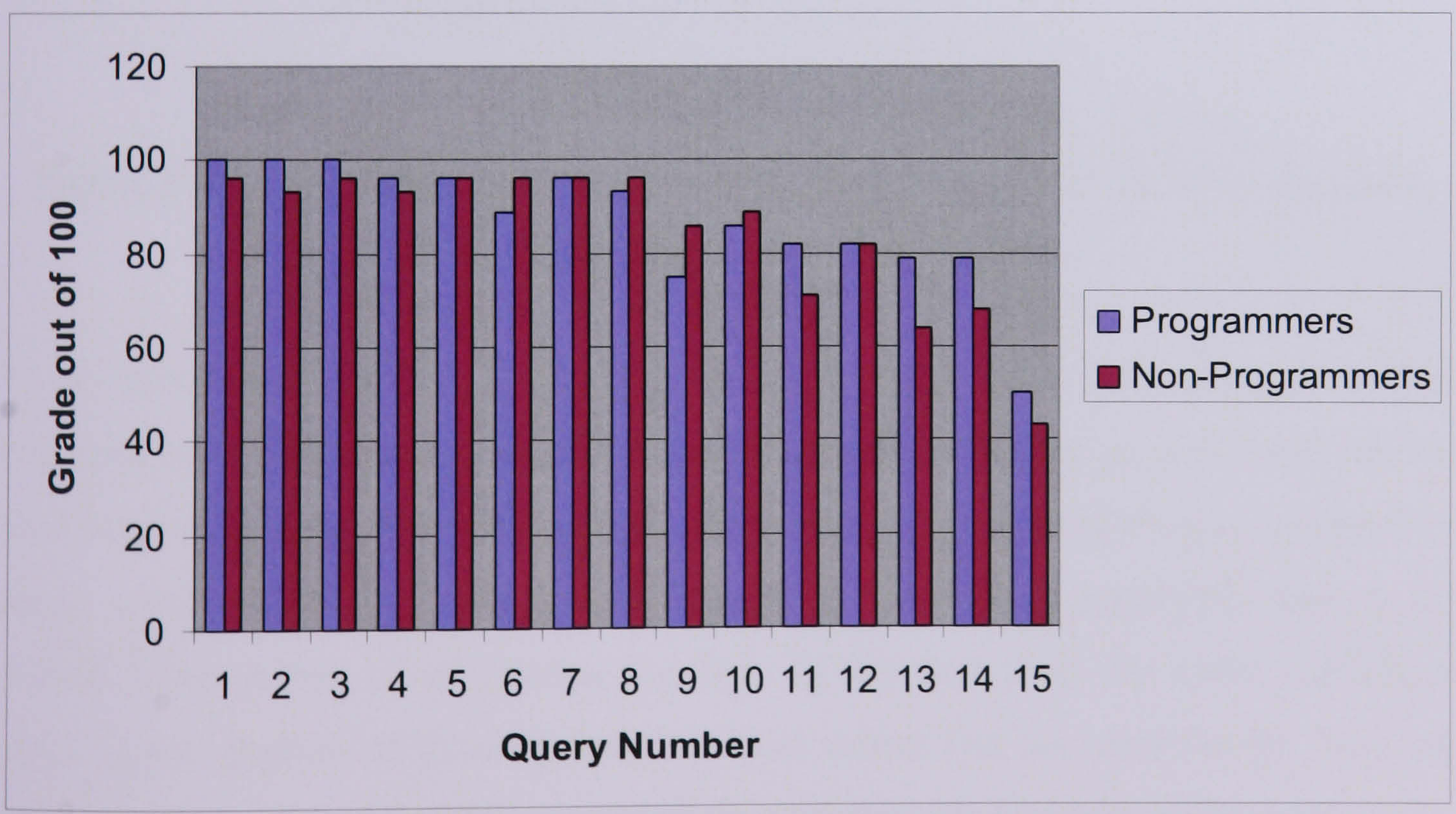


Figure 4.22: Histogram to Compare the Results of the Programmers Group versus the Non-Programmers Group.

The mean of the queries scores for each of the programmers group, non-programmers group, and both groups are respectively 86.9, 84.52, and 85.71 out of 100 as shown in Table 4.9 and Figure 4.23.

Average Means of Queries	Programmers (28)	Non-Programmers (28)	Both Groups (56)
Average of Correct Answers	24.33	23.67	48
Percentage out of 100	86.9	84.52	85.71

Table 4.9: The Means of the Programmers, Non-programmers, and Both Groups.

Figure 4.23: Percentage of Correct and Wrong Answers of the Programmers, Non-programmers, and Both Groups.

To get more statistical analysis about the mean averages and in order to check if there is a significant difference between the mean averages of the two groups P and NP, the *t*-test statistic is used. The *independent samples t-test* has been applied to compare the mean scores of the two groups with respect to each query separately then to the overall mean scores of all queries together. In the first case, the same confidence level, alpha, degrees of freedom, and t critical values that are used for the icons are considered for the queries. The results of the statistics are shown in Table 4.10.

Query #	P mean score	NP mean score	t Statistics	p-value2tails
1	100.00	96.43	1.000	0.322
2	100.00	92.86	1.441	0.155
3	100.00	96.43	1.000	0.322
4	96.43	92.86	0.585	0.561
5	96.43	96.43	0.000	1.000
6	89.29	96.43	-1.029	0.308
7	96.43	96.43	0.000	1.000
8	92.86	96.43	-0.585	0.561
9	75.00	85.71	-1.000	0.322
10	85.71	89.29	-0.397	0.693
11	82.14	71.43	0.940	0.351
12	82.14	82.14	0.000	1.000
13	78.57	64.29	1.177	0.244
14	78.57	67.86	0.896	0.374
15	50.00	42.86	0.528	0.600

Table 4.10: The t Statistics of Each Query Used for the Significant Differences of the Means.

The *t statistic* of each query is compared to the *t critical for two-tail* which is 2.004879. If the *t statistic* is less than 2.004879, it can be concluded that there is no significant difference between the means of the programmers sample group and the non-programmers. For example, the *t statistic* of the first query is 1 which is less than 2.004879. Hence, it can be reported that there is no significant difference between the means of the programmers and the non-programmers with respect to answering correctly the query formulation of the first query. The green *t statistic* values in Table 4.6 show that all the fifteen queries were formulated by the subjects of both groups with no significant means difference. It is worth noting that the programmers performed better than the non-programmers in the first 5 simple queries and the last 5 complex queries whereas the non-programmers performed better than the programmers in the intermediate ones.

The *t*-test has also been applied to compare the overall mean scores of the two groups with respect to all the queries together. The results show that the *t statistic* is equal to 0.440 and the *t critical* is 2.048407. Hence, there is no significant difference between the means. In general, it can be reported that there is no significant difference between the means of the programmers and the non-programmers sample groups with respect to the query formulation. The queries that are evaluated are divided into two categories namely the simple queries and the complex ones. The simple queries are

the ones numbered from 1 to 8 and the complex ones from 9 to 15. The average score of each category is calculated. The mean of the simple queries scores for each of the programmers group, non-programmers group, and both groups are respectively 96.43, 95.54, and 95.98 out of 100. The mean of the complex queries scores for each of the programmers group, non-programmers group, and both groups are respectively 76.02, 71.94, and 73.98 out of 100 as shown in Table 4.11 and Figures 4.24.

Average Means of Simple vs. Complex	Programmers (28)	Non-Programmers (28)	Both Groups (56)
Percentage of Simple Queries	96.43	95.54	95.98
Percentage of Complex Queries	76.02	71.94	73.98

Table 4.11: The Means of the Programmers, Non-programmers, and Both Groups.

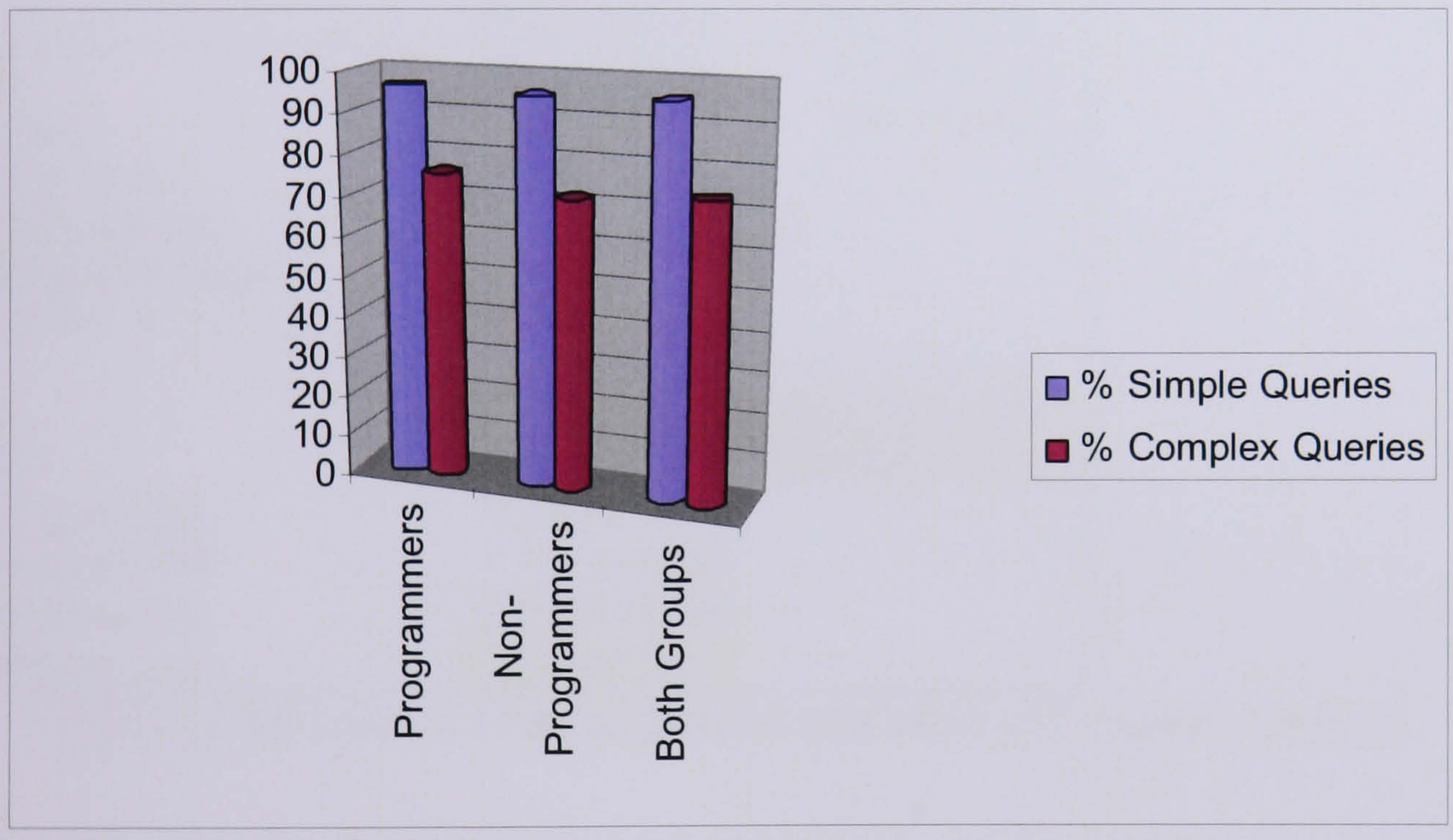


Figure 4.24: Percentage of Correct and Wrong Answers of the Programmers, Non-programmers, and Both Groups.

To get more statistical analysis about the mean averages of the simple queries the *independent samples t-test* has been applied to compare the mean scores of the two groups. Another *independent samples t-test* has also been applied to check if there is a mean difference between the programmers and the non-programmers at the 95% confidence level and 5% significance level. The results of the statistics are shown in Figure 4.25.

t-Test: Two-Sample Assuming Equal Variances		
	SIMPLE QUERIES 1-8	
	Programmers	*Non-Programmers*
Mean	96.25	95.25
Variance	15.07142857	1.928571429
Observations	8	8
Pooled Variance	8.5	
Hypothesized Mean Difference	0	
df	14	n1+n2-2=8+8-2
t Stat	0.685994341	<2.14 so accept Ho
P(T<=t) one-tail	0.25195975	
t Critical one-tail	1.761310115	
P(T<=t) two-tail	0.503919499	
t Critical two-tail	2.144786681	

t-Test: Two-Sample Assuming Equal Variances		
	COMPLEX QUERIES 9-15	
	Programmers	*Non-Programmers*
Mean	76.14285714	71.85714286
Variance	144.4761905	251.1428571
Observations	7	7
Pooled Variance	197.8095238	
Hypothesized Mean Difference	0	
df	12	n1+n2-2=7+7-2
t Stat	0.570077156	<2.17 so accept Ho
P(T<=t) one-tail	0.289571251	
t Critical one-tail	1.782287548	
P(T<=t) two-tail	0.579142503	
t Critical two-tail	2.178812827	

Figure 4.25: *t-test* of Simple Queries and *t-test* of Complex Queries.

The results of the above two t-tests are summarized in Table 4.12.

Query Type	P mean score	NP mean score	t Statistics	p-value2tails
Simple 1-8	96.250	95.250	0.686	0.504
Complex 9-15	76.143	71.857	0.570	0.579

Table 4.12: The t Statistics of Each Query Type.

The *t statistic* of the simple queries is compared to the *t critical for two-tail* which is 2.144786681. The *t statistic* is less than 2.144786681, thus it can be decided not to reject the hypothesis Ho which states that the two means are equal. The Hypothesized Mean Difference is 0. There is no significant difference between the means of the programmers sample group and the non-programmers with respect to simple queries.

The *t statistic* of the complex queries is compared to the *t critical for two-tail* which is 2.178812827. The *t statistic* is less than 2.178812827, thus it can be decided not to reject the hypothesis Ho which states that the two means are equal. The Hypothesized Mean Difference is 0. There is no significant difference between the means of the programmers sample group and the non-programmers with respect to the complex queries.

4.6.3 The Results of the Evaluation of the User Satisfaction

The evaluation of the user satisfaction has been conducted using a questionnaire made up of twenty five questions that are scored by means of a 7-point Likert scale. The answer 7 reflects the highest user satisfaction and the answer 1 reflects the lowest. The data collected about the user satisfaction questionnaire are shown in Table 4.13 where each row corresponds to one subject and each column corresponds to one question. As before, the column entitled U contains the name of the university that the subject belongs to and the one entitled P contains either the letter P for Programmer or the letter N for Non-Programmer.

Answers of the 56 Subjects to the Questionnaire
25 Questions / 7-point Likert Scale

#	U	P	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
1	A	P	7	7	7	6	7	6	6	5	7	7	7	7	7	7	7	7	6	7	7	7	7	7	6	7	7
2	A	P	6	6	6	6	5	6	6	7	6	7	6	6	6	6	7	6	7	7	7	6	6	7	6	6	6
3	A	P	7	7	7	6	7	6	7	6	6	7	7	7	7	6	7	7	6	7	7	7	7	7	7	6	6
4	A	P	6	7	7	5	5	5	6	7	7	7	7	7	6	6	7	6	7	7	6	6	6	7	6	7	7
5	A	P	7	6	7	6	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7
6	A	P	7	6	7	7	6	7	7	6	7	7	7	7	7	7	7	7	7	7	6	7	7	7	7	7	7
7	A	P	7	7	7	6	7	7	7	7	7	7	6	7	7	7	7	6	7	7	7	7	7	7	7	7	7
8	A	P	7	6	6	6	7	6	7	5	7	7	7	7	7	7	7	7	7	7	7	7	6	7	7	7	7
9	A	P	6	7	7	5	7	7	6	7	7	6	7	7	7	7	7	7	7	6	7	7	6	7	7	7	7
10	A	P	7	6	7	7	7	7	7	7	7	7	7	7	7	7	7	6	7	7	7	7	7	7	7	7	7
11	A	P	7	7	6	6	7	7	7	6	6	7	7	7	6	7	5	6	7	7	7	6	6	7	6	7	7
12	A	P	5	6	6	4	4	6	4	4	4	6	7	6	5	5	5	4	4	4	4	6	5	5	5	5	7
13	A	P	5	5	6	4	6	7	7	7	7	7	5	5	5	7	1	7	7	7	7	7	7	7	7	5	6
14	A	N	7	7	7	6	6	7	6	6	7	7	7	6	6	7	7	6	7	7	7	7	7	7	7	7	7
15	H	P	6	4	6	5	6	6	5	4	4	6	7	6	7	6	6	7	5	4	4	6	6	5	5	5	6
16	H	P	6	6	6	7	6	6	6	6	7	7	7	7	7	6	7	6	7	7	7	7	7	7	6	7	6
17	H	P	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7
18	H	P	7	7	7	6	7	7	6	6		7	7	7	7	7	7	7	7	7	7	6	6	7	7	7	7
19	H	P	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7
20	H	P	6	5	7	7	7	6	7	7	6	6	6	7	7	7	6	7	7	7	7	6	7	7	7	7	6
21	H	P	7	5	6	5	5	5	6	5	6	6	7	7	6	6	6	7	6	7	6	7	7	6	7	6	7
22	H	P	7	6	6	7	6	7	7	7	6	7	5	6	7	7	7	6	7	7	7	6	6	6	7	7	7
23	H	P	7	7	7	7	7	7	6	7	7	7	6	6	6	7	6	7	7	7	7	6	6	7	7	7	7
24	H	N	5	6	5	5	5	6	5	6	5	6	7	6	7	7	7	6	7	7	7	7	7	6	7	6	7
25	H	N	7	7	5	6	7	6	7	6	7	7	7	7	6	7	7	7	6	7	7	7	7	7	7	7	7
26	H	N	6	6	6	6	6	5	6	6	6	7	7	6	7	7	6	7	7	6	7	7	6	6	7	7	7
27	H	N	6	7	7	6	6	6	6	6	6	5	5	5	5	5	6	6	6	6	6	5	5	5	6	6	6
28	H	N	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7
29	H	N	6	7	7	6	7	5	5	6	6	7	7	6	6	7	7	7	6	7	7	7	7	6	7	6	6
30	H	N	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7
31	H	N	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7
32	H	N	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7
33	H	N	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7
34	U	P	6	5	4	5	6	6	4	6	6	2	6	6	6	6	5	5	6	6	4	6	6	5	6	5	2
35	U	P	5	6	6	7	6	7	5	6	7	5	7	7	7	7	5	6	6	6	6	7	7	7	7	7	6
36	U	P	6	6	6	3	5	5	4	4	6	6	7	5	6	7	4	7	7	7	7	7	6	6	7	7	5
37	U	P	6	5	6	5	6	6	6	4	6	4	6	6	5	5	4	6	6	6	6	6	6	5	7	6	4
38	U	P	6	5	7	5	5	7	6	2	6	5	5	6	6	4	5	5	7	7	5	7	5	6	6	6	5
39	U	P	5	6	4	5	2	4	6	5	7	3	5	6	6	6	6	5	6	6	4	6	6	5	6	6	6
40	U	N	6	6	6	6	6	6	7	6	6	7	7	7	7	7	7	7	7	7	6	7	7	7	7	7	7
41	U	N	7	5	6	6	5	6	7	6	7	7	6	7	6	6	5	6	5	6	6	7	7	7	7	6	4
42	U	N	6	4	7	5	6	6	5	4	7	4	6	5	6	5	5	6	6	6	5	6	5	5	6	6	6
43	U	N	5	6	6	5	3	4	6	3	2	7	6	6	6	6	1	2	5	5	5	6	6	6	7	2	5
44	U	N	6	5	5	3	4	5	3	1	5	6	7	6	6	6	4	5	6	7	7	7	7	7	7	6	5
45	U	N	6	6	6	5	4	4	6	6	7	6	6	5	7	5	5	4	6	6	6	4	5	5	6	7	5
46	U	N	5	4	4	5	4	6	6	6	6	5	7	5	5	5	5	5	6	6	6	5	6	4	5	6	6
47	U	N	5	4	6	6	6	5	6	6	5	5	7	6	6	6	5	6	6	6	5	6	6	6	7	6	6
48	U	N	6	6	6	6	5	6	4	2	6	6	6	6	6	6	4	6	6	6	6	6	6	6	6	6	5
49	U	N	6	6	7	5	7	7	5	3	7	7	6	6	7	6	6	5	7	6	6	7	7	7	6	7	5
50	U	N	5	6	5	4	6	5	5	5	6	6	7	7	7	7	7	6	5	7	7	7	7	7	7	7	7
51	U	N	5	5	7	3	6	6	5	4	7	5	6	7	7	6	6	5	7	7	7	7	6	6	7	6	7
52	U	N	7	7	6	6	7	7	6	6	5	4	7	5	7	7	5	4	7	7	7	6	6	6	6	6	7
53	U	N	3	3	3	1	3	4	3	1	5	3	7	5	5	4	7	3	4	4	3	5	5	3	3	6	3
54	U	N	6	5	6	4	5	5	5	4	6	5	5	6	5	5	6	6	6	6	6	6	6	5	5	4	4
55	U	N	5	6	5	4	6	3	5	6	6	4	6	4	6	4	6	6	4	6	3	6	5	6	5	3	6
56	U	N	6	4	6	5	7	7	5	5	6	6	5	6	7	5	5	7	6	5	5	6	4	4	6	6	7

Table 4.13: The Grades of the Subjects Answers to the User Satisfaction Questionnaire.

The collected data is represented visually using a graphical chart namely the histogram bar chart. The histogram in Figure 4.26 shows for each question the mean of the answers and its respective value as converted to a score over 100.

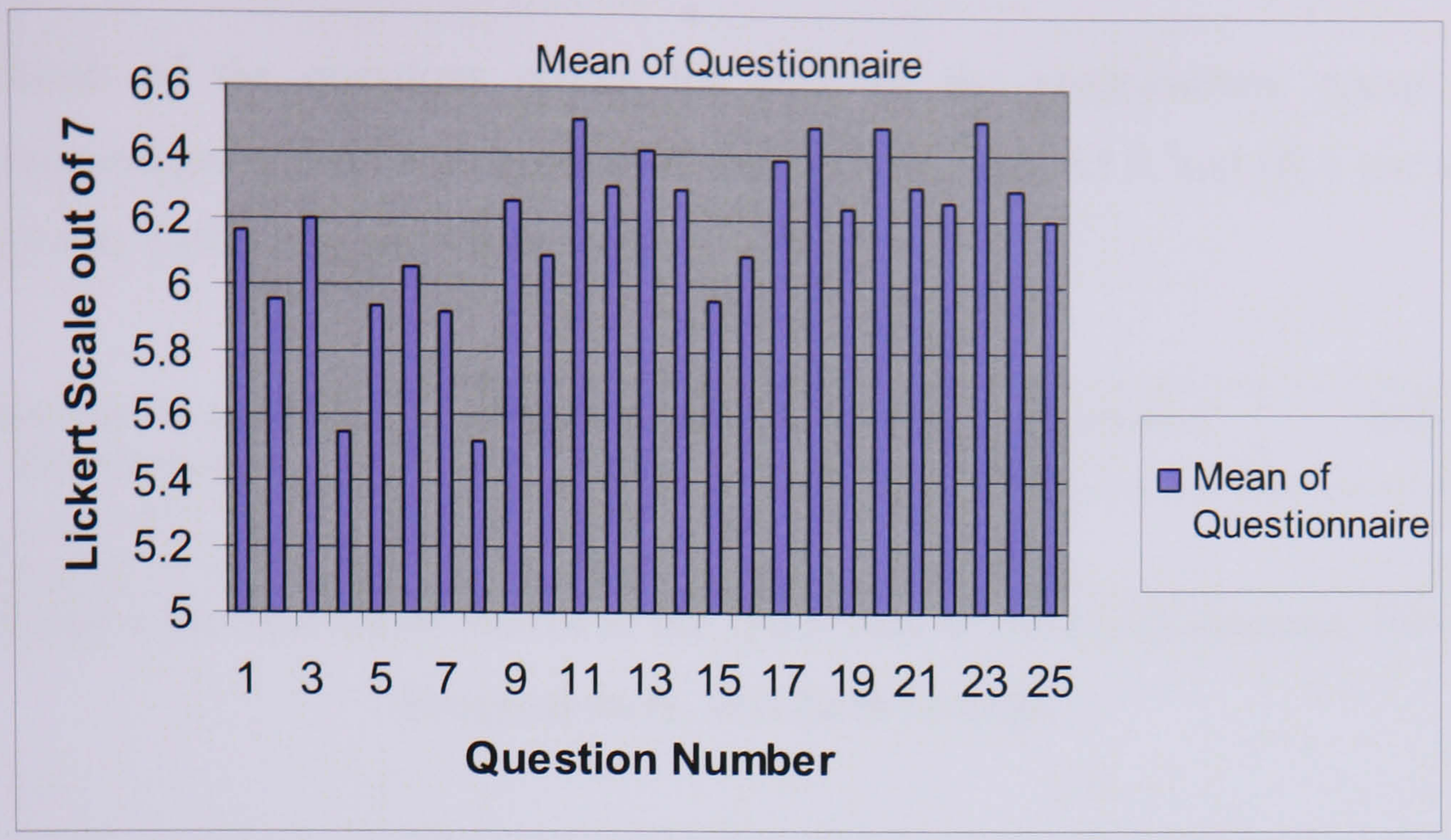


Figure 4.26: The Average of Questionnaire Scores.

The aspect of the visual query language and its user interface are reflected by the scores submitted by the subjects on each question. The score 7 means that the subject had the best and highest preference and satisfaction whereas the score 1 means that the subject had the worst and lowest preference and satisfaction. The highest the score is the better the aspect is. The mean scores of the two subject group are compared to check if the is a significant mean difference. Figure 4.27 shows the mean scores of each question of the questionnaire for the programmers sample group in blue and the non-programmers in red.

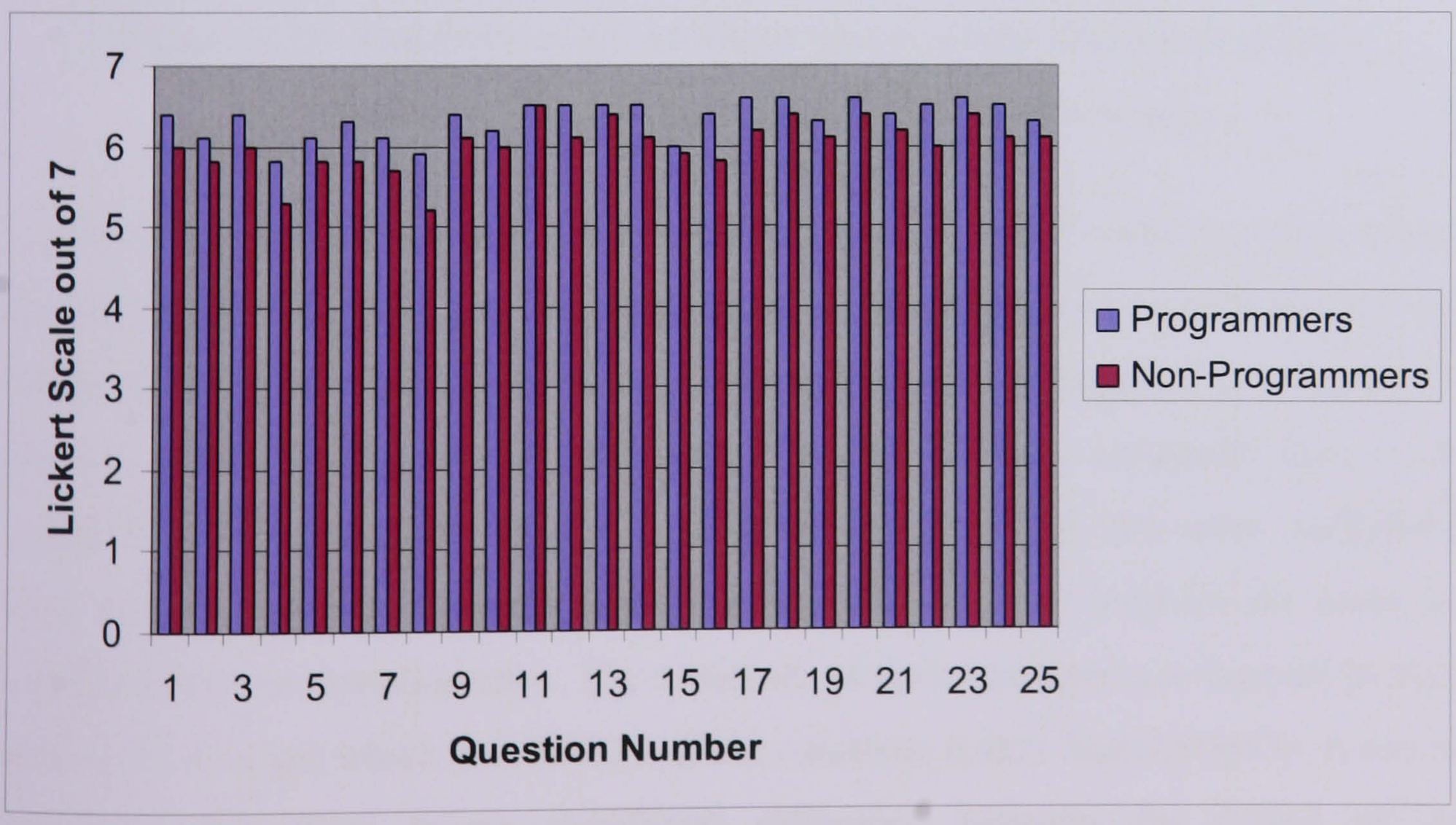


Figure 4.27: Histogram to Compare the Results of the Programmers Group versus the Non-Programmers Group.

The mean of the questions scores for each of the programmers group, non-programmers group, and both groups are respectively 90.4, 85.9, and 88.1 out of 100 as shown in Table 4.14 and Figure 4.28.

Average Means of Questionnaire	Programmers (28)	Non-Programmers (28)	Both Groups (56)
Average out of 7	6.33	6.01	6.17
Average out of 100	90.4	85.9	88.1

Table 4.14: The Mean Scores of the Questions of the Programmers, Non-Programmers, and Both Groups.

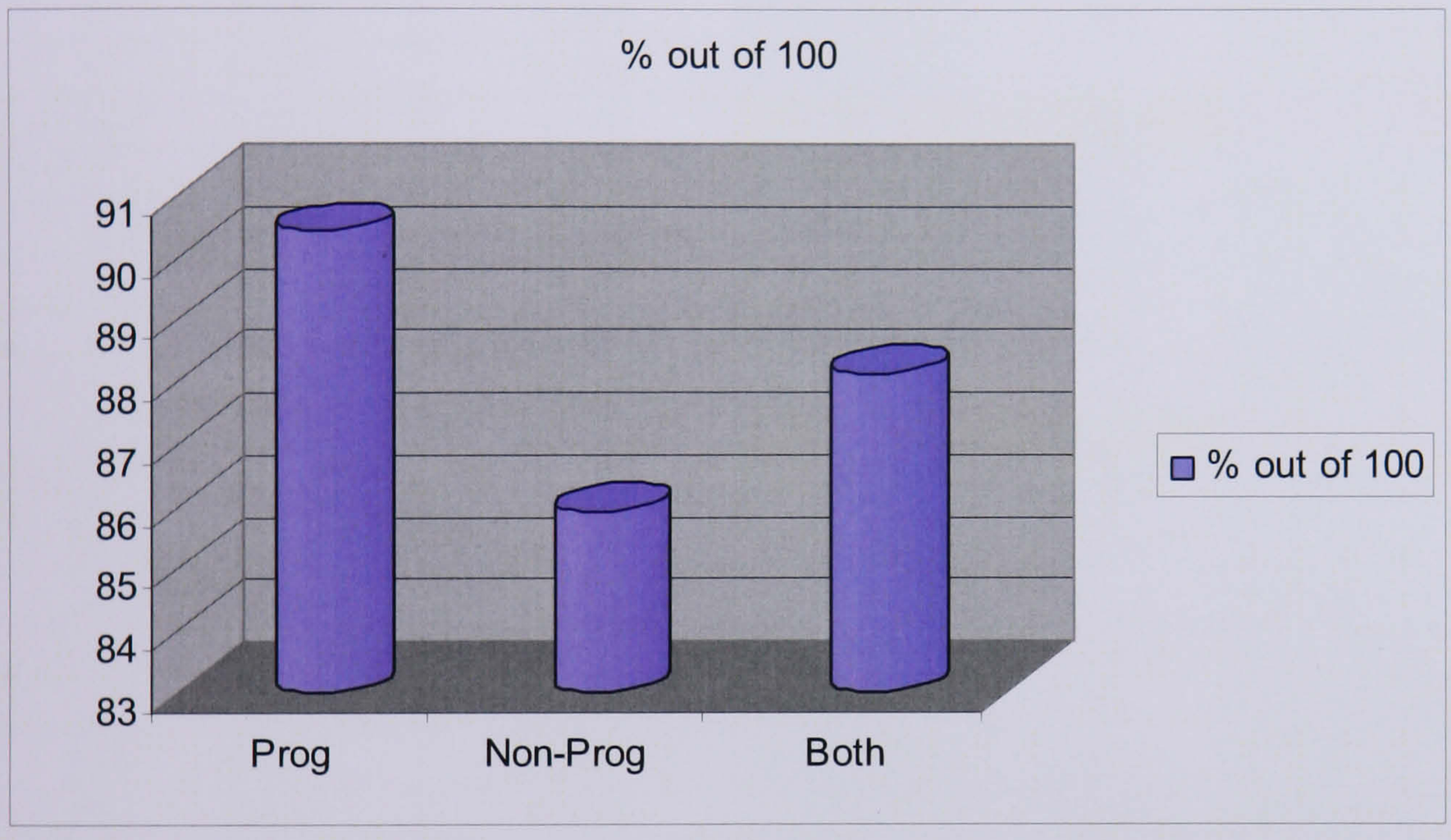


Figure 4.28: The Percentage of Mean Scores of the Questions of the Programmers, Non-Programmers, and Both Groups.

To get more statistical analysis about the mean averages and in order to check if there is a significant difference between the mean averages of the two groups P and NP, the *t*-test statistic is used. The *independent samples t-test* has been applied to compare the mean scores of the two groups with respect to each question separately then to the overall mean scores of all queries together. In the first case, the same confidence level, alpha, degrees of freedom, and t critical values that are used for the icons are considered for the questionnaire. The *t statistic* of each question is compared to the *t critical for two-tail* which is 2.004879. If the *t statistic* is less than 2.004879, it can be concluded that there is no significant difference between the means of the programmers sample group and the non-programmers. For example, the *t statistic* of

the first question is 1.721 which is less than 2.004879. Hence, it can be reported that there is no significant difference between the means of the programmers and the non-programmers with respect to their user satisfaction of the aspect of the first question. The green *t statistic* values in Table 4.15 show that twenty three out of twenty five aspects of the visual query language and its user interface are found of the same level of user satisfaction by the subjects of both groups.

Question #	P mean score	NP mean score	t Statistics	p-value1tail	p-value2tails
1	6.357	5.964	1.721	0.045	0.091
2	6.071	5.821	0.903	0.185	0.371
3	6.357	6.036	1.284	0.102	0.205
4	5.786	5.321	1.370	0.088	0.176
5	6.071	5.786	0.875	0.193	0.385
6	6.321	5.786	2.027	0.024	0.048
7	6.143	5.679	1.627	0.055	0.109
8	5.857	5.179	1.627	0.055	0.110
9	6.407	6.107	1.133	0.131	0.262
10	6.214	5.964	0.743	0.230	0.461
11	6.500	6.500	0.000	0.500	1.000
12	6.536	6.071	2.299	0.013	0.025
13	6.464	6.357	0.563	0.288	0.576
14	6.464	6.107	1.486	0.071	0.143
15	6.036	5.857	0.481	0.316	0.633
16	6.357	5.821	1.834	0.036	0.072
17	6.571	6.179	1.777	0.041	0.081
18	6.607	6.357	1.160	0.126	0.251
19	6.321	6.143	0.597	0.276	0.553
20	6.571	6.393	0.972	0.168	0.336
21	6.393	6.214	0.850	0.200	0.399
22	6.464	6.036	1.668	0.051	0.101
23	6.607	6.393	0.991	0.163	0.326
24	6.464	6.107	1.316	0.097	0.194
25	6.321	6.071	0.810	0.211	0.421

Table 4.15: The t Statistics of Each Question Used for the Significant Differences of the Means.

If the *t statistic* is greater than 2.004879, it can be concluded that there is a significant difference between the means of the programmers sample group and the non-programmers. In such a case, the *t statistic* is compared to the *t critical for one tail* which is 1.673565. If it is greater, it can be concluded that the programmers sample group performed better than the non-programmers, and vice-versa. For example, the *t statistic* of the twelfth question which is 2.299 is also greater than 1.673565. Hence, it

can be reported that the programmers sample subject performed better than the non-programmers, with t=2.299 and p=0.013. The programmers have a higher user satisfaction than the non-programmers. There is a 1.3% chance that this result occurred by random chance.

The yellow *t statistic* values in Table 4.15 show that only two questions out of twenty five reflect a significance difference between the means of the sample groups. The first one is the question number 6 with t=2.027 and p=0.024. The second one is the question number 12 with t=2.299 and p=0.013. In both cases, it can be reported that at a 5% level of significance, the programmers have a higher user satisfaction with respect to the aspects of questions 6 and 12.

The *t*-test has also been applied to compare the overall mean scores of the two groups with respect to all the questions together. The results show that the *t statistic* is equal to 4.152027 and the *t critical* is 2.010635. The *t statistic* is greater than the *t critical* for two tails. The hypothesis is rejected. The means of the two groups are not equal. In such a case, the *one tail statistic* is used instead of the *two tails*. The *t stat* is greater than the *t critical one tail* which is 1.677224. The hypothesis that "programmers mean is less than or equal to the non-programmers mean" is also rejected. The programmers mean is greater than the non-programmers mean in all 25 questions. It is concluded that programmers have a higher user satisfaction with respect to the visual query language and its user interface. Moreover, it is clearly noted that the programmers performed better than the non-programmers in 24 questions where a difference in the results is observed. If the null hypothesis is true, which means that there is a 50% chance of programmers performing better than non-programmers, then the probability of this happening by random chance is $(0.5)^{24} \approx 6 \times 10^{8}$ or around 1 chance in 17 million. Hence, it can be concluded that there is therefore a strong indication that the programmers perform better than the non-programmers though the magnitude of this difference is small. The t-test result is shown in Figure 4.29.

t-Test: Two-Sample Assuming Equal Variances		
USER SATISFACTION QUESTIONNAIRE - ALL QUESTIONS		
	Programmers	*Non-Programmers*
Mean	6.34	6.016
Variance	0.051666667	0.100566667
Observations	25	25
Pooled Variance	0.076116667	
Hypothesized Mean Difference	0	
df	48	
t Stat	4.152026685	>2.10 &>1.677 so P>NP
P(T<=t) one-tail	6.71434E-05	
t Critical one-tail	1.677224197	
P(T<=t) two-tail	0.000134287	
t Critical two-tail	2.010634722	

Figure 4.29.: *t-test* of all the Questions of the Questionnaire Used to Check for Mean Difference of Programmers and Non-Programmers.

4.6.4 Discussion

The user testing and the user satisfaction have been both chosen in order to evaluate the expressive power of the smiley icons, their level of recognition, the ease of use, the user interface, the query building and formulation, and the expressive power of the IVQL language. The results of the analysis show that the subjects found that the smiley icons have a good expressive power, a high level of recognition, and are easy to use. They also found that the user interface is very good, the query building and formulation easy, and that the IVQL language has a very good expressive power.

4.6.4.1 Discussion of the Results of the Evaluation of the Smiley Icons

The user testing reported the results about the expressive power of each icon. The attributes smiley icon recognition as well as smiley icon representation are used to reflect the clarity of each icon. The level of recognition is 93% which means that all the subjects were able to identify and recognize easily the majority of the icons. Two icons only proved to be recognized with difficulty or badly recognized namely the taxi and the metro icons. It is suggested to either change the icons or amend their

picture in a way to make them visualize better the objects that they are supposed to represent.

A t-test analysis has been done for each icon separately in order to check if there is a significant difference between the level of recognition of the programmers group and that of the non-programmers group. It can be reported that at a 5% level of significance, there is no significant difference between the means of the programmers and the non-programmers with respect to recognizing twenty seven icons out of thirty. The three icons that visualize respectively a hotel, taxi, and ice skating show a significant difference in the results. It can be reported that at a 5% significance level, the programmers were able to recognize the icons better than the non-programmers. However, since it was found that the probability of having at least three "False Positives" is 19%, there may not, therefore, be anything particularly significant about the icons 2 (Hotel), 20 (Taxi), and 30 (Ice Skating); they may simply be random anomalies.

The overall mean scores of the two subject groups, the programmers and the non-programmers, are respectively 93 and 88. The t-test has also been used to compare the mean scores of the two subject groups. The results show that at a 5% level of significance, there is no significant difference between overall means of the programmers and the non-programmers sample groups. Hence, it can be concluded that the smiley icons are very easily recognized by both sample groups and that there is no significant difference between their levels of recognition. However, it can also be said that based on the 1-tail test, only 1.1% of this result occurred by chance and that there is a significant (albeit small) bias towards the programmers, though not especially important. Hence, it can be concluded that there is some evidence for thinking that the programmers perform better than the non-programmers.

The user satisfaction questions 7 to 10 reported the results about the icons. They measure the level of consistency of the use of icons throughout the system, the relation between the operator icons and the tasks that they are supposed to perform, and the consistency of the messages that are displayed on the screen. On the average, all the subjects found that the use of the icons throughout the system was 84% consistent with a mean score of 5.917. The programmers group found it 88%

consistent whereas the non-programmers group found it 81% consistent. The t-test analysis can report that at the 5% level of significance, there is no significant difference between the means of the programmers and the non-programmers groups. It can be concluded that both subject groups had the same level of user satisfaction about the use of icons throughout the system.

On the average, all the subjects found that the operator icons are 79% related to the tasks that they are supposed to perform with a mean score of 5.52/7. The programmers group found them 84% related whereas the non-programmers group found them 74% related. The t-test analysis can report that at the 5% level of significance, there is no significant difference between the means of the programmers and the non-programmers groups. It can be concluded that both subject groups had the same level of user satisfaction about the operator icons being related to the tasks that they are supposed to perform. On the average, all the subjects found that the position of the messages on the screen was 89% consistent with a mean score of 5.91/7. The programmers group found it 92% consistent whereas the non-programmers group found it 87% consistent. The t-test analysis can report that at the 5% level of significance, there is no significant difference between the means of the programmers and the non-programmers groups. It can be concluded that both subject groups had the same level of user satisfaction about the position of the messages on the screen.

On the average, all the subjects found that the error messages and error prevention were 87% helpful with a mean score of 6.09/7. The programmers group found them 89% helpful whereas the non-programmers group found them 85% helpful. The t-test analysis can report that at the 5% level of significance, there is no significant difference between the means of the programmers and the non-programmers groups. It can be concluded that both subject groups had the same level of user satisfaction about the error messages and error prevention. Hence, it can be concluded that the smiley icons are used easily by both sample groups and that there is no significant difference between their levels of user satisfaction. A comparison between IVQL icons evaluation with the ones conducted in the reviewed visual query languages shows that the work done in [Bon02] evaluated the expressive power of the language by user testing whereas it is done in IVQL by user satisfaction. The results that are reported in [Bon02] show that the icons have a 65% expressive power whereas the

IVQL evaluation results show that the icons have a 90% expressive power, hence, reflecting a higher expressive power in favour of the IVQL icons.

4.6.4.2 The Discussion of the Results of the Evaluation of the Query Formulation

The user testing has been chosen in order to evaluate the ease of use of the IVQL language. The ease of use is determined through measuring the visual query writing task of the subjects. Questions 1 to 8 cover the simple queries formulation whereas questions 9 to 15 cover the complex queries formulation noting that a query is considered complex if it includes one or more 'and' operator to combine two or more simple queries. The total average of all the queries scores is 85.71 out of 100. It can be concluded that in general, the visual queries are very easily formulated using the IVQL visual query language. The t-test analysis that is used for each query leads to the fact that there is no significant difference between the means of the programmers and the non-programmers with respect to answering correctly each of the query formulation questions. In other words, both groups subjects answered each query with the same level of achievement, either both groups having high scores or both having low ones.

It can be reported that there is no significant difference between the overall average means of the programmers and the non-programmers with respect to answering correctly all the query formulation questions. It can be noted that both programmers and non-programmers performed better on the simple query formulation questions than on the complex query formulation. This is normally expected since it is due to the fact that complex queries are more difficult to write and formulate than simple ones. The t-test analysis can report that at the 5% level of significance, there is no significant difference between the means of the programmers and the non-programmers groups with respect to answering the simple query formulation questions. It can be concluded that both subject groups had the same level of performance in formulating simple queries.

The t-test analysis can report that at the 5% level of significance, there is no significant difference between the means of the programmers and the non-programmers groups with respect to answering the complex query formulation questions. It can be concluded that both subject groups had the same level of performance in formulating complex queries. The evaluation of the query building and formulation shows that people from different backgrounds like programmers and non-programmers are expected to perform the same when using the IVQL visual query language. They do not need to have any programming experience in order to be able to use easily IVQL to formulate queries or understand its visual language. However, it is worth noting that the programmers performed better than the non-programmers in the first 5 simple queries as well as the last 5 complex queries, whereas the non-programmers performed better than the programmers in the intermediate ones.

When comparing the results of the evaluation of IVQL query formulation with the ones reviewed in the literature it can be noted that in IVQL the programmers performed in general as good as the non-programmers. This fact proves that IVQL is equally understood by people from different backgrounds. In the evaluation done in [Mur00] the programmers performed better than the non-programmers due to the fact that the visual query resembles a lot the form of an SQL statement. In the one done in [Bon02] GIS users performed better than non-GIS users due to the fact the GIS users have a good background in querying GIS applications. Hence it can be concluded that the query formulation in IVQL can be more easily understood by the generic public than the ones that are reviewed in the literature.

4.6.4.3 The Discussion of the Results of the Evaluation of the User Satisfaction

The user satisfaction has been chosen in order to evaluate the user interface, the query formulation, and the expressive power of the IVQL language. The user interface is evaluated by the questions 1-6 and 15-16. The subjects reported that they found the software 88% excellent, 85% easy, and 89% satisfying. They found that reading icons on the screen was 79% easy, that the organization of information was 85%

clear, and that the sequence of screens was 86% clear. The t-test analysis of each aspect can report that at the 5% level of significance, there is no significant difference between the means of programmers and the non-programmers groups except for the sequence of screens where the programmers found it clearer than the non-programmers. Hence, it can be concluded that both groups have the same level of satisfaction about the user interface.

The query building and formulation is evaluated by the questions 17-25. The subjects reported that they found the selection of the operators 91% easy, the selection of categories 93% easy, the selection of objects 89% easy, building simple queries 93% easy, the queries 90% clear, the queries 89% legible, remembering the language queries 93% easy, the sequence used to build the queries 90% easy, and using the 'and' operator to formulate complex queries 89% easy. The t-test analysis of each aspect can report that at the 5% level of significance, there is no significant difference between the means of programmers and the non-programmers groups. Hence, it can be concluded that both groups have the same level of satisfaction about the query building and formulation.

The expressive power of IVQL is evaluated by the questions 11-14. the subjects reported that they found the learning to operate the system 93% easy, exploring new features by trial and error 90% easy, remembering the names and the use of icons 92% easy, and performing tasks is 90% straightforward. The t-test analysis of each aspect can report that at the 5% level of significance, there is no significant difference between the means of programmers and the non-programmers groups except for the exploring new features by trial and error where the programmers found it easier than the non-programmers. Hence, it can be concluded that both groups have the same level of satisfaction about the expressive power of the IVQL language. However, it is clearly noted that the programmers performed better than the non-programmers in 24 questions where a difference in the results is observed. If the null hypothesis is true, which means that there is a 50% chance of programmers performing better than non-programmers, then the probability of this happening by random chance is $(0.5)^{24} \approx 6 \times 10^{8}$ or around 1 chance in 17 million. Hence, it can be concluded that there is therefore a strong indication that the programmers perform better than the non-

programmers though the magnitude of this difference is small. Moreover, it is important to note that none of the reviewed visual query languages have used the user satisfaction in order to evaluate the aspects that are evaluated in IVQL. This fact might be considered as an advantage of the evaluation of IVQL over the others.

4.6.4.4 The Improved Operators Icons

Some of the subjects commented that the operators shown in Figure 4.30 namely, Nearest Neighbor with Path and Within Buffer, were not so easily understood.

Figure 4.30: The Old Not Easily Understood Operators.

The operators have been reviewed and changed based upon their suggestions to become as shown in Figure 4.31. The same subjects reported their satisfaction about the new ones.

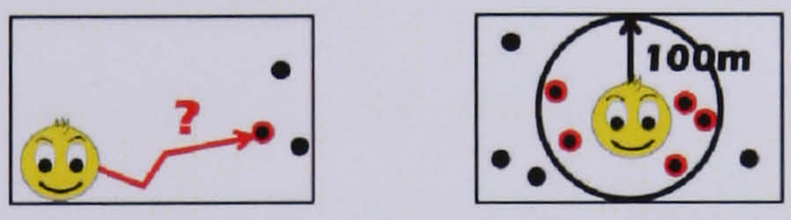


Figure 4.31: The Improved Operators.

Accordingly, new operators have been designed to visualize static and dynamic proximity operators as shown in Figure 4.32.

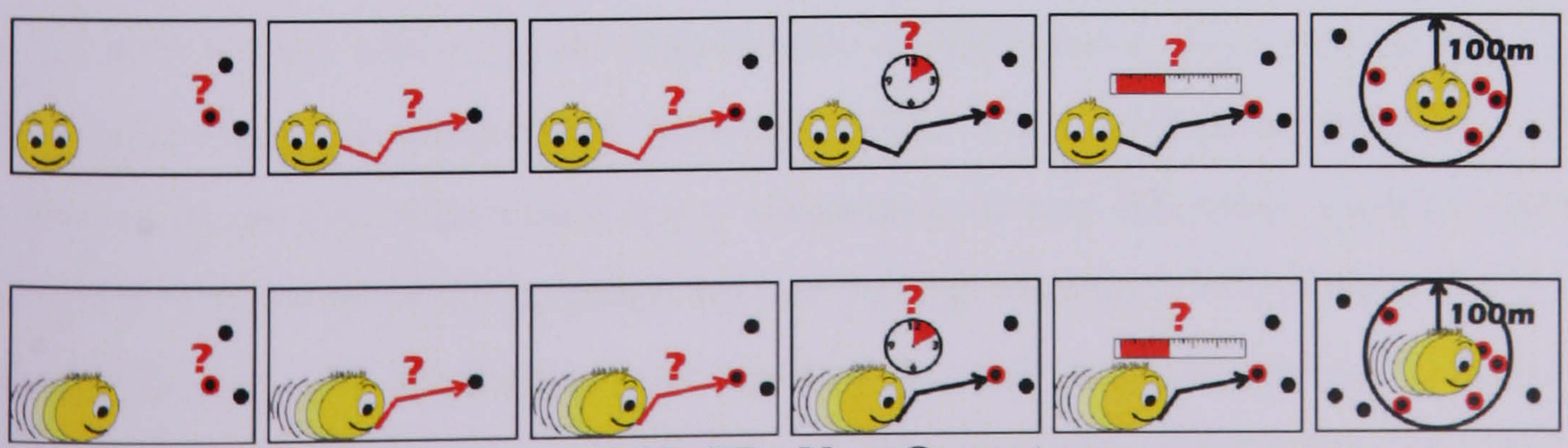


Figure 4.32: The New Operators.

4.7 Conclusion

This chapter presented a new visual query language for Mobile GIS which addresses many limitations of previous visual query languages. It also presented the software architecture which can process any visual query. The proposed language is based on smiley icons that cover operators, values, themes, and objects, which were extensively evaluated. The methods of evaluation were discussed and the evaluations which are based on questionnaires were carried out. The obtained results have been very encouraging. The user testing and the user satisfaction have been both chosen in order to evaluate the expressive power of the smiley icons, their level of recognition, the ease of use, the user interface, the query building and formulation, and the expressive power of the IVQL. They proved that the advantage of the evaluation implemented in this work over the evaluations implemented in the review of literature was that all the aspects of usability of IVQL were evaluated whereas in each of the other reviewed evaluations one aspect of usability only was evaluated such as query writing in [Smi05], user interface in [Mor04], and expressiveness of icons and language in [Bon02].

All the subjects were able to identify and recognize easily the majority of the icons without significant difference between the programmers and the non-programmers groups. Most of the subjects found that the use of the icons throughout the system were consistent, the operator icons were related to the tasks that they were supposed to perform. The visual queries were found very easy to formulate using the IVQL. Subjects were also able to answer correctly each of the query formulation questions of both simple queries and complex queries. With respect to the user interface, the subjects found the software excellent, easy, and satisfying. They reported that reading icons on the screen was easy, the organization of information clear, and the sequence of screens clear. A comparison between IVQL icons evaluation with the ones conducted in the reviewed visual query languages showed that other work evaluated the expressive power of the language by user testing whereas it was done in IVQL by user satisfaction. The results that were reported by other work showed that their icons had a 65% expressive power whereas the IVQL evaluation results showed that the icons had a 90% expressive power, hence, reflecting a higher expressive power in favour of the IVQL icons.

With respect to the query building and formulation, the subjects found all the aspects easy namely, the selection of the operators, the selection of categories, the selection of objects, building simple queries, formulating the queries, understanding the queries, remembering the language queries, the sequence used to build the queries, and using the 'and' operator to formulate complex queries. With respect to the expressive power of IVQL, the subjects found that the following aspects were easy namely the learning to operate the system and remembering icons. The subjects understood quickly the smiley icons, user interface, query building and formulation, and proved to be satisfied with the aspects of the expressive power of the language. They also found that the user interface is very good, the query building and formulation easy, and that the IVQL language has a very good expressive power. The results of the analysis show that the subjects found that the smiley icons have a good expressive power, a high level of recognition, and are easy to use. They show that people from different backgrounds like programmers and non-programmers are expected to perform the same when using the IVQL visual query language. They do not need to have any programming experience in order to be able to use easily IVQL to formulate queries or understand its visual language.

When comparing the results of the evaluation of IVQL query formulation with the ones reviewed in the literature it was noted that in IVQL the programmers performed in general as good as the non-programmers. This fact proved that IVQL is equally understood by people from different backgrounds. In the evaluation done other work the programmers performed better than the non-programmers due to the fact that the visual query resembled a lot the form of an SQL statement or GIS users performed better than non-GIS users due to the fact the GIS users had a good background in querying GIS applications. Hence it was concluded that the query formulation in IVQL could be more easily understood by the generic public than the ones that were reviewed in the literature. Moreover, it was important to note that none of the reviewed visual query languages used the user satisfaction in order to evaluate the aspects that were evaluated in IVQL. This fact was considered as an advantage of the evaluation of IVQL over the others.

As a summary, the prototype of an iconic visual query language was designed and implemented with the aim to address the challenge of having a visual dynamic complex query language for Mobile GIS. It was constructed and evaluated taking into consideration all the advantages and limitations of the reviewed visual query languages.

Chapter 5 - Query Melting

5.1 Introduction

In order to efficiently process multiple dynamic complex queries for proximity analysis, which are formulated by multi-users who are accessing the GIS server simultaneously, the existing query optimization strategies were critically analyzed in Chapter 3 which concluded with the need to develop new strategies. On one hand, this can be achieved by a thorough examination of the execution plans of the GIS operators with the aim to determine the existence of commonalities between their functions, steps, processes, and objects, as well as to specify which functions can be swapped or re-ordered. On the other hand, an investigation is needed to determine how the common functions are eliminated and the other ones re-ordered, hence to be able to employ a new Melting Ruler Mechanism that is responsible for sharing spatial areas of various queries, temporal intervals, objects of interest, and operator templates functions. Moreover, it is necessary to determine the possible combinations of *n* Operator Templates that can be used to formulate a complex query, where each combination is considered as a scenario, and to investigate the time cost where multiple users formulate similar scenarios, hence to be able to develop a new Decision Making Mechanism (TCOP) that is responsible for sharing Global Execution Plans (GEP) and resulting in time cost optimization.

The rest of the chapter explores the commonalities that exist between the GIS operators, defines the Query Melting Paradigm, identifies the components of the Query Melting Processor, and elaborates the Melting Ruler Mechanism. Section 5.2 introduces the concept of commonalities in GIS such as commonalities between the execution plans of static operators, dynamic operators with one-predicate, and dynamic operators with multiple predicate respectively, with the aim to determine which can be shared and how sharing can be applied. Section 5.3 introduces the query melting paradigm and elaborates how it includes the sharing paradigm, push-down approach, traditional query optimization steps, and sharing GEP paradigm. Section 5.3.1 presents the components of the query melting processor and explains how each component operates in order to apply the melting ruler mechanism. Section 5.3.2

introduces the decision making mechanism of TCOP and explains how it operates in order to employ the sharing GEP paradigm. Section 5.4 introduces the icons that represent the operators, values, and objects in the dynamic complex queries that will be used in the rest of the chapter. Section 5.4.1 explains how each GIS operator can be translated into an execution plan, called Template, and shows examples. Section 5.4.2 describes the query melting process of various combinations of complex queries and shows how they are converted into a global execution plan. Finally, some concluding remarks are presented in section 5.5.

5.2 Commonality in GIS

A thorough examination of different GIS and LBS applications shows that they have in common some functionalities, operations, and objects in execution plans. A predicate is the operator used by the user while formulating a query such as Find Within a buffer, Find the k Nearest Facilities (kNN), Find the Time Left to reach the nearest facility, etc. There are two types of operators namely the static and the dynamic. Each operator is decomposed as per its own query evaluation plan which is made up of a list of functions/operations that are to be executed in the proper sequence.

5.2.1 Commonality in Query Execution Plans of Static Operators

The simple form of a query evaluation plan is a simple plan of an operator. Figure 5.1 shows the query evaluation plan of an operator, where each dot represents a function and the execution is done in the top down direction. Multiple queries could be formulated for different facilities such as find the nearest 3 bookstores, find the nearest 5 schools, and find the nearest 7 coffee shops. The same evaluation plan but for different objects has to be repeatedly executed for each of the queries Q_1, $Q_2 \ldots Q_n$ as shown in Figure 5.2(a). Some of the functions that are executed for Q_1 can be reused for $Q_2 \ldots Q_n$. They are called common functions or common operations. Instead of repeating *n* times the same execution plan, a new global evaluation plan can be formulated and built. Figure 5.2(b) shows that some functions (black and white dots) can be executed only once while others (blue and red dots) are to be repeated for each query. The common functions that can be reused are in fact deleted from the global evaluation plan, hence, they are melted.

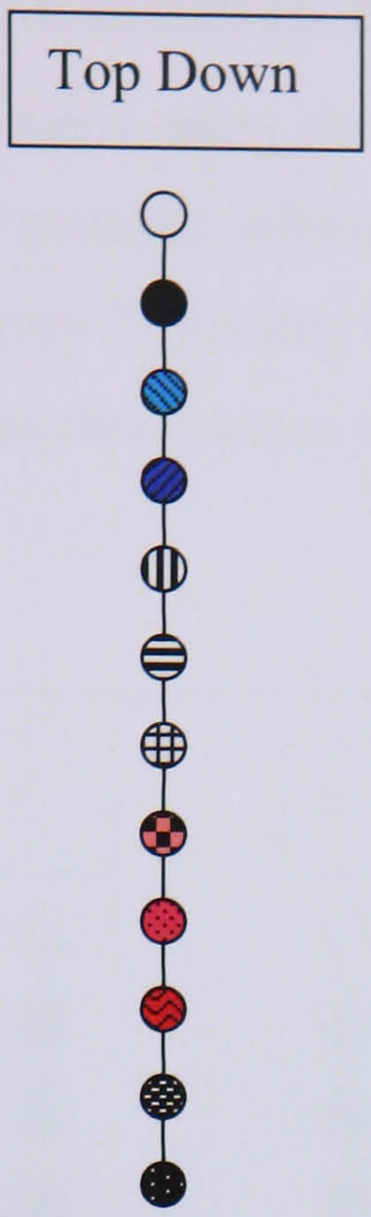
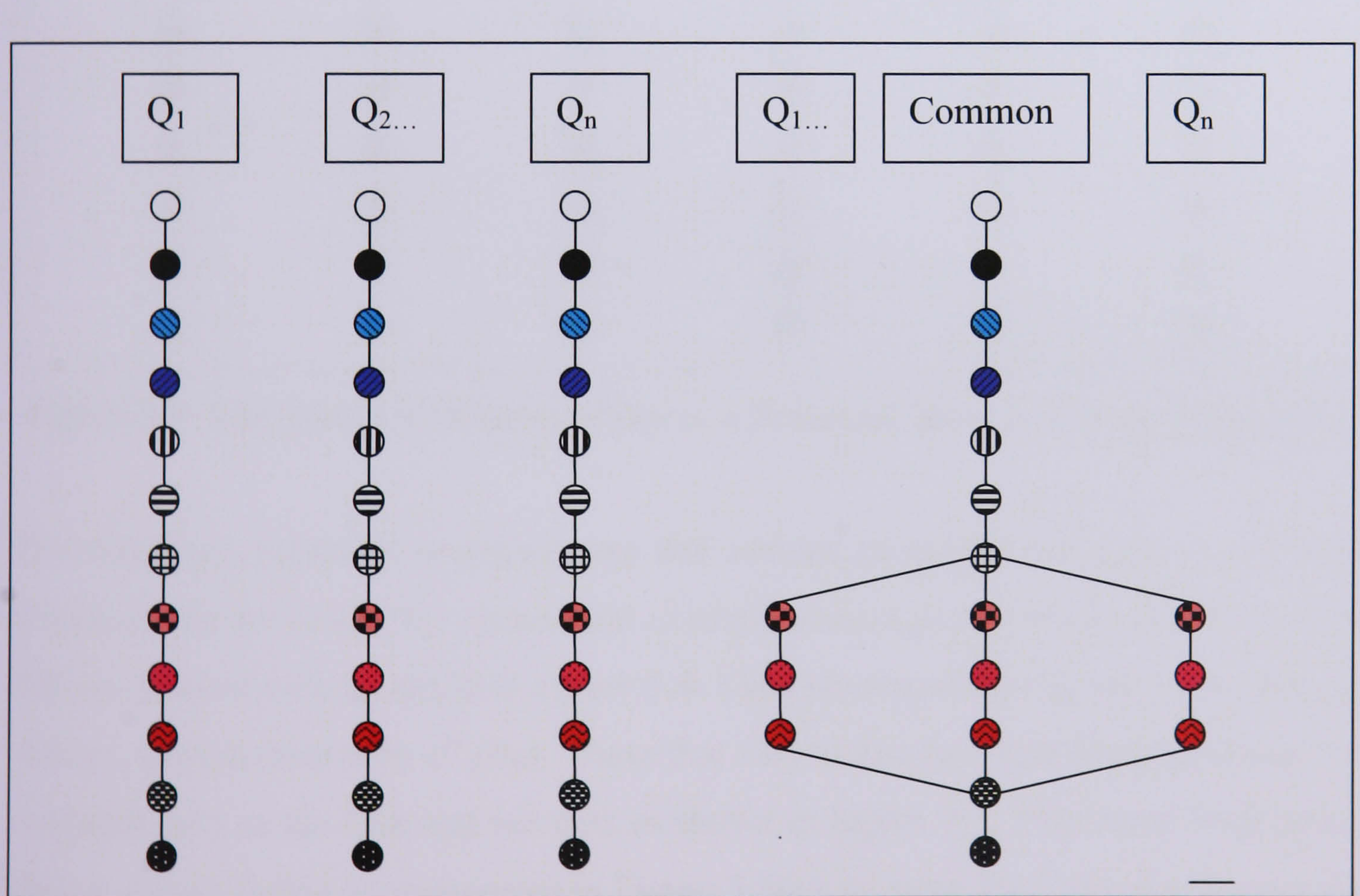


Figure 5.1: Query Evaluation Plan of a Static Query with One Operator.



(a) *n* Executions of Same Plan

(b) The Execution Plan Showing That The Common Operations Can Be Reused.

Figure 5.2: Query Evaluation Plan of Multiple Static Queries with One Operator.

5.2.2 Commonality in Query Execution Plans of Dynamic Operators

A dynamic query launches a continuous query that lives a certain period of time such as starting now and for the next 30 minutes, inform me about the time left to reach the nearest restaurant, updating me every 2 minutes based on my new locations. Figure 5.3 shows that at every new XY location update time instance t_i the same evaluation plan is repeated.

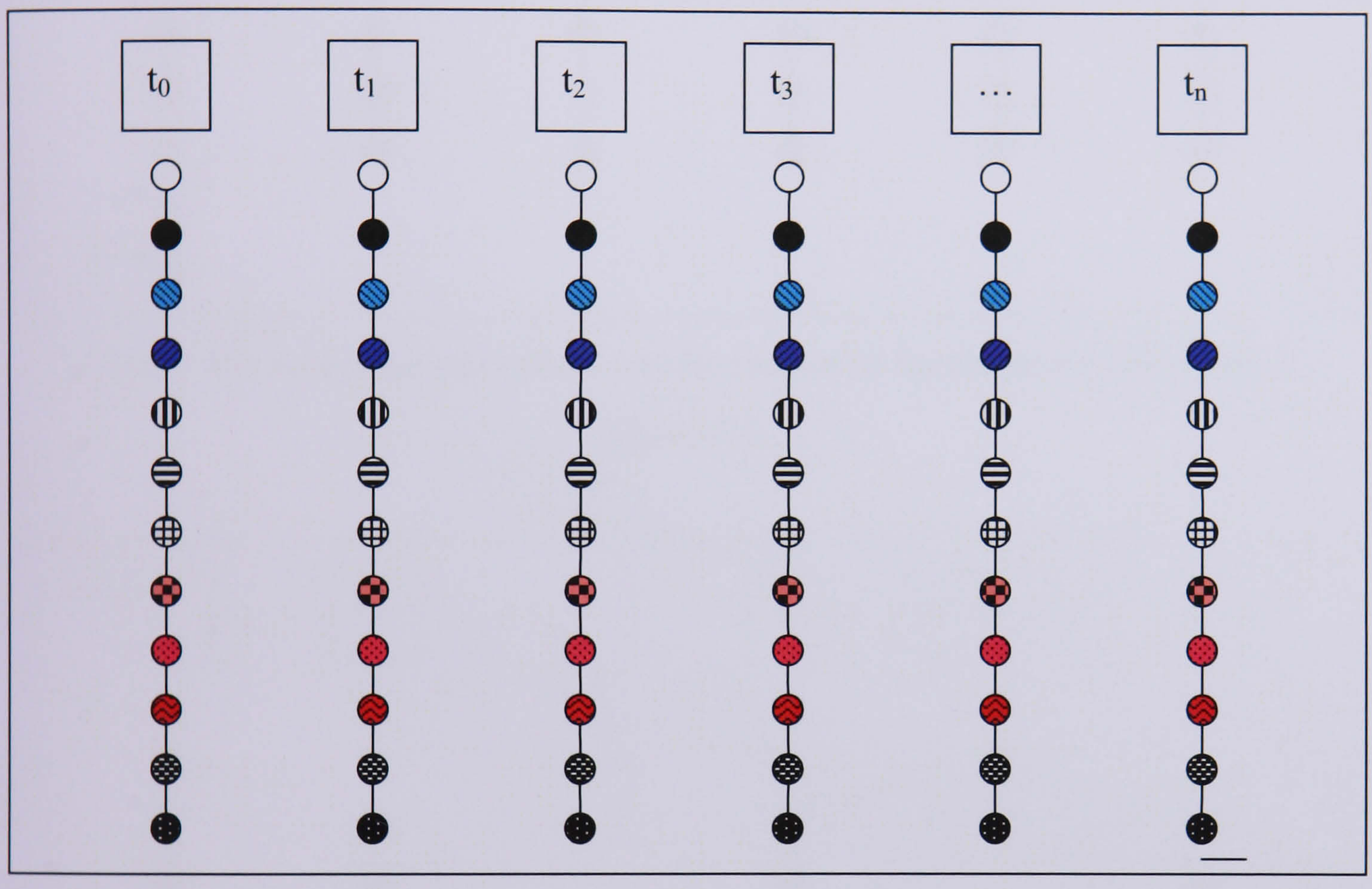


Figure 5.3: The Query Evaluation Plan of a Dynamic Query with One Operator.

The functions, methods, or parameters that remain invariable for some or all time instances can be reused by consequent executions such as the black and white dots that are marked with an arrow in Figure 5.4. They are executed only once at t_0 and are deleted /melted from their t_i^{th} plan. Those that vary at any new time instance should be executed such as the blue and red dots as shown in Figure 5.5. The Query Evaluation Plan in Figure 5.5(a) is represented in Figure 5.5(b) showing the Common Operations that can be melted.

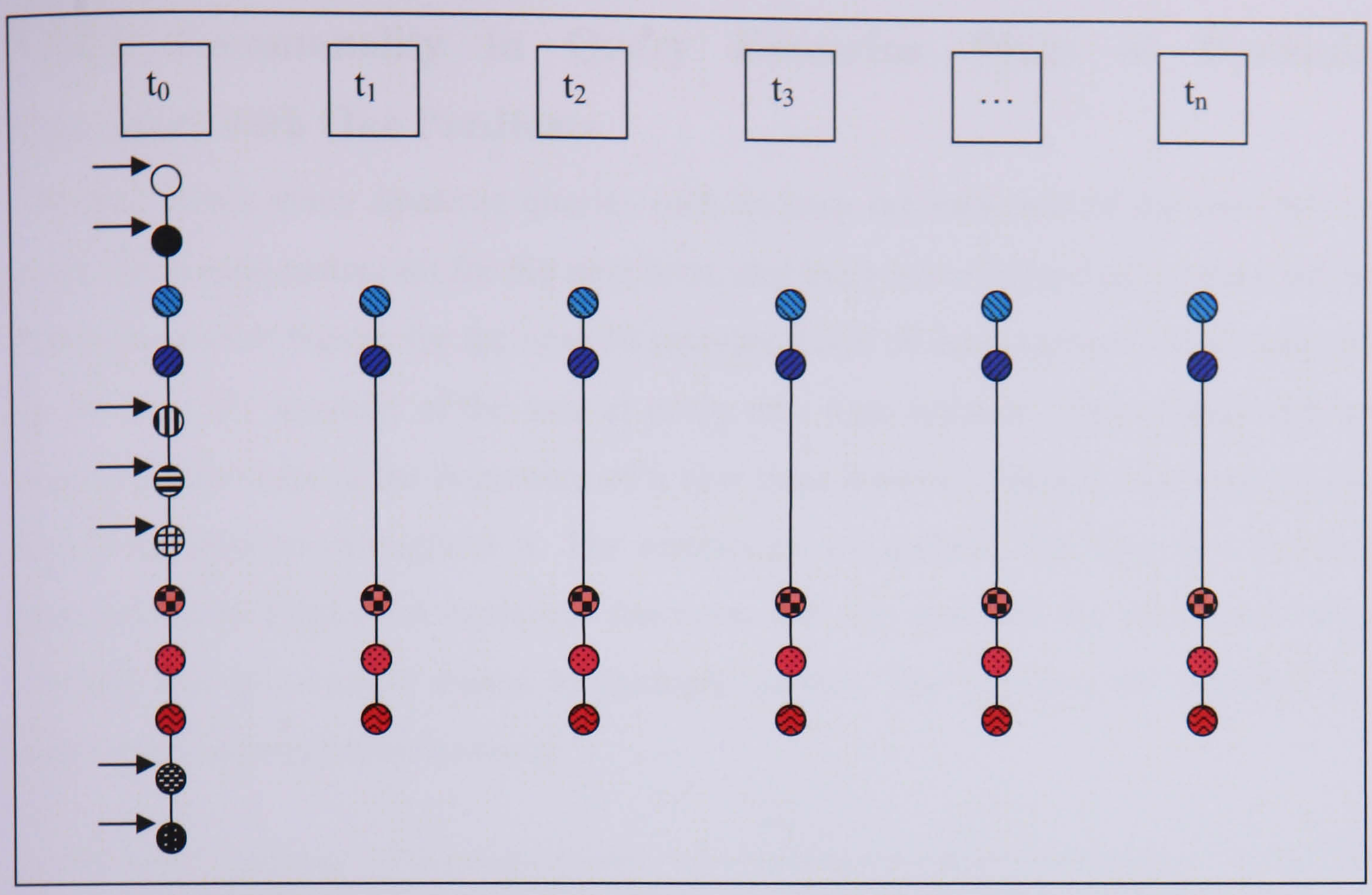
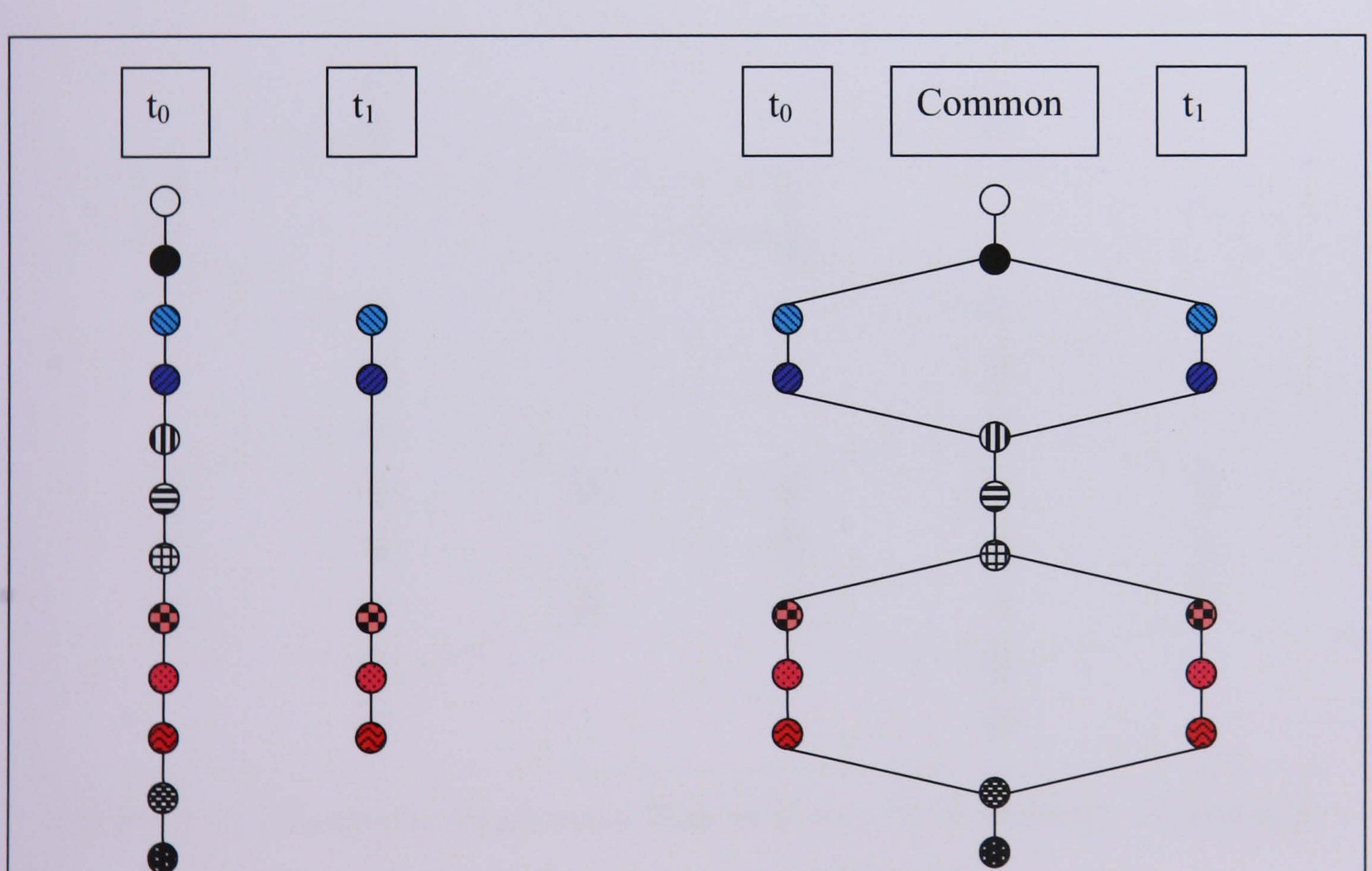


Figure 5.4: Common operators can be reused in Dynamic Queries with 1 Operator.



(a) (b)

Figure 5.5: Query Evaluation Plan of Multiple Dynamic Queries with 1 Operator.

5.2.2.1 Commonality in Query Execution Plans of Dynamic Operators with One Predicate

The user issues many dynamic queries such as keep me informed of the time left to reach the nearest restaurant for the next hour, and keep me informed of the time left to reach the nearest theatre for the next 30 minutes. Each of these two queries is updated by the new XY location of the user at every new time instance. This change can be executed only once at the beginning of a new time interval. Then, it is reused by the rest of the queries throughout it. The repetitions are melted. The blue dots marked with arrows in Figure 5.6 represent functions that are executed for every new time instance and then can be shared by multiple queries. The red ones are executed for every query at every time instance.

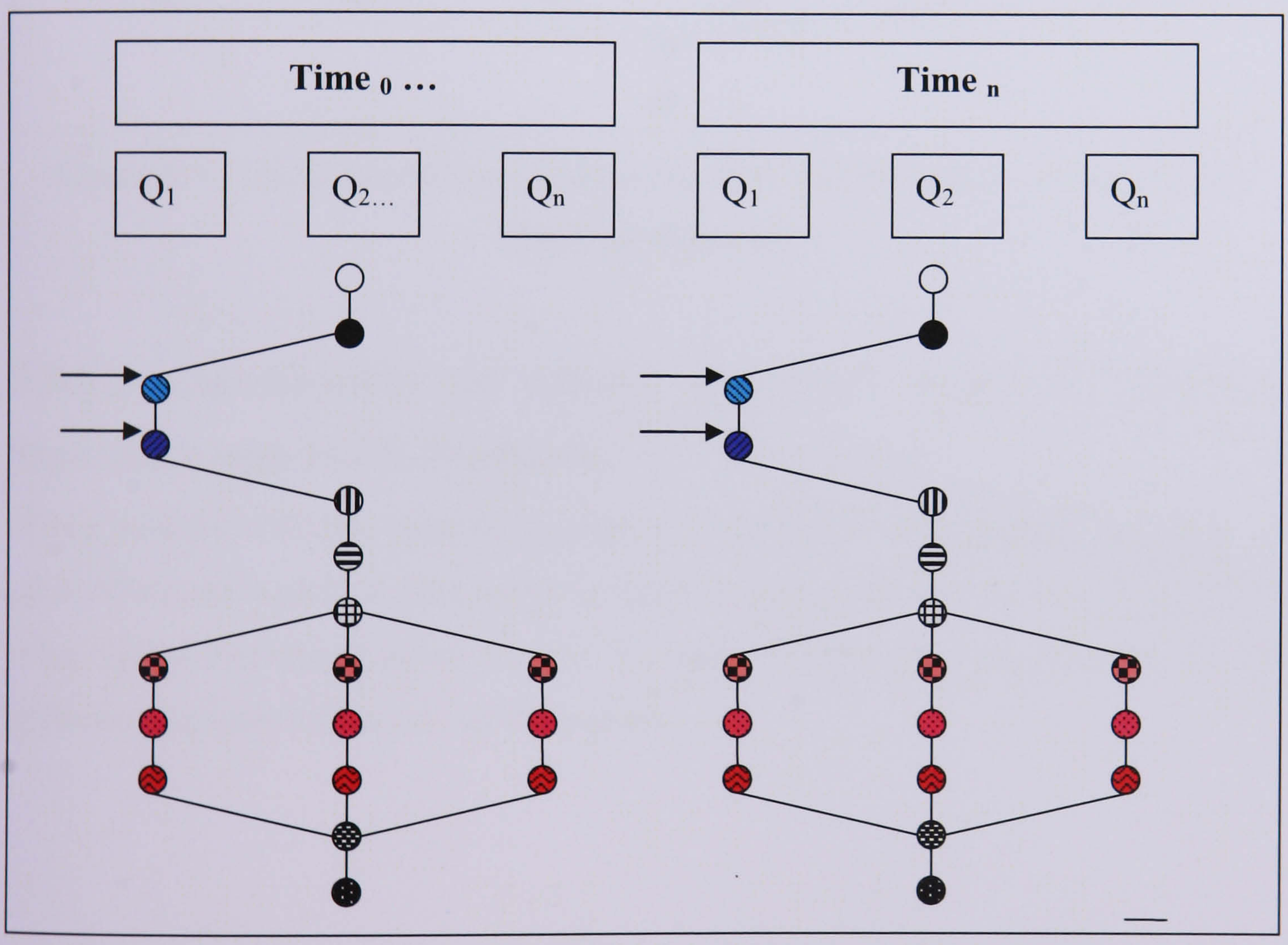


Figure 5.6: The Query Evaluation Plan of Every Time Instance of Multiple Dynamic Queries with One Operator.

Figure 5.7 shows how the black and white dots are melted to be executed only once for all the continuous queries and the blue ones at the beginning of each time instance. The red are not melted so executed at each time instance of each query.

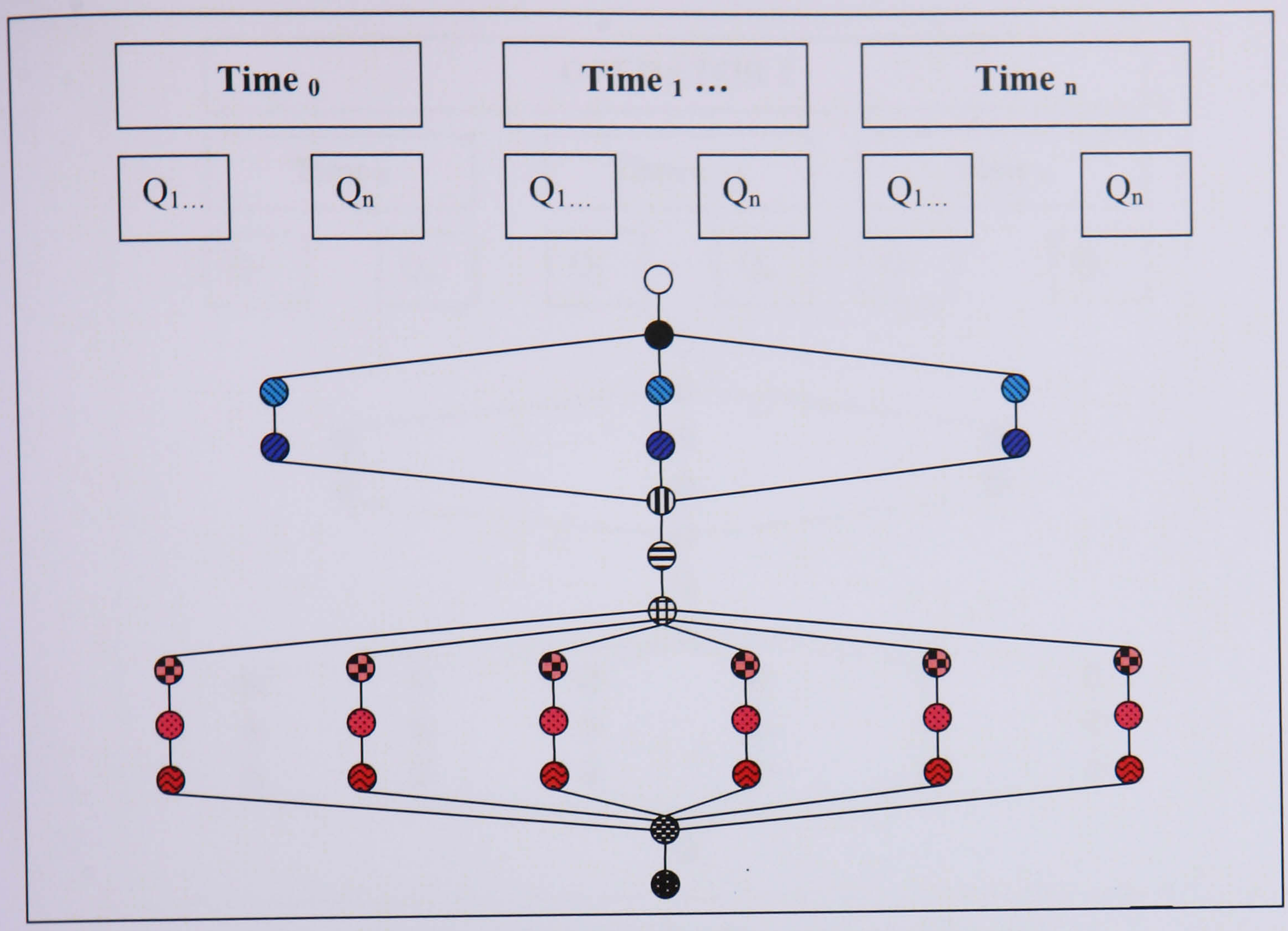


Figure 5.7: The Global Query Evaluation Plan of Multiple Dynamic Queries with One Operator.

5.2.2.2 Commonality in Query Execution Plans of Dynamic Operators with Multi-Predicates

When multiple dynamic complex queries are formulated with multiple operators the global evaluation plan of each operator might include functions that can be reused by other global evaluation plans. Figures 5.8 and 5.9 show the two plans of the two different operators Operator1 and Operator2.

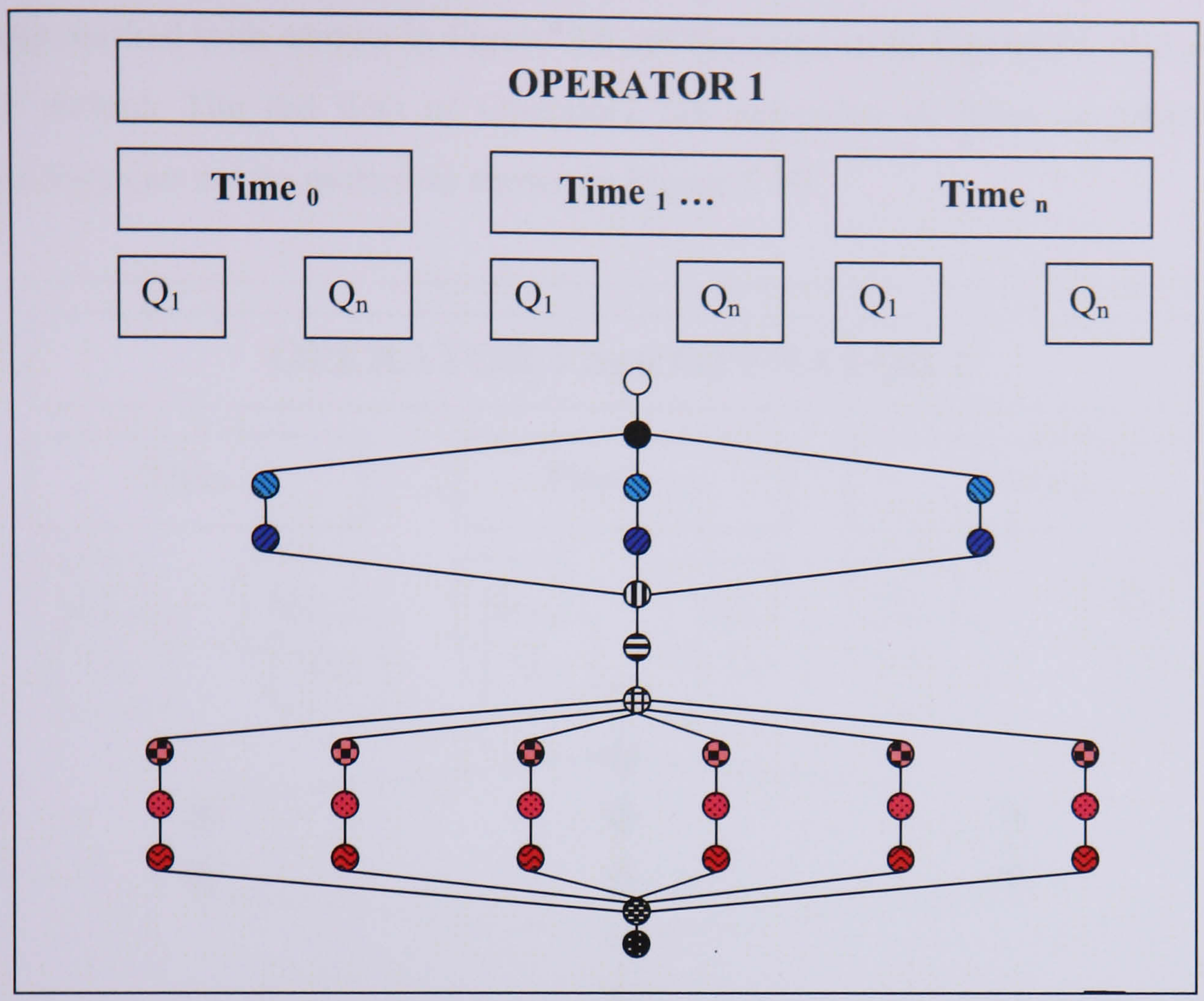


Figure 5.8: The Global Evaluation Plan of Operator1.

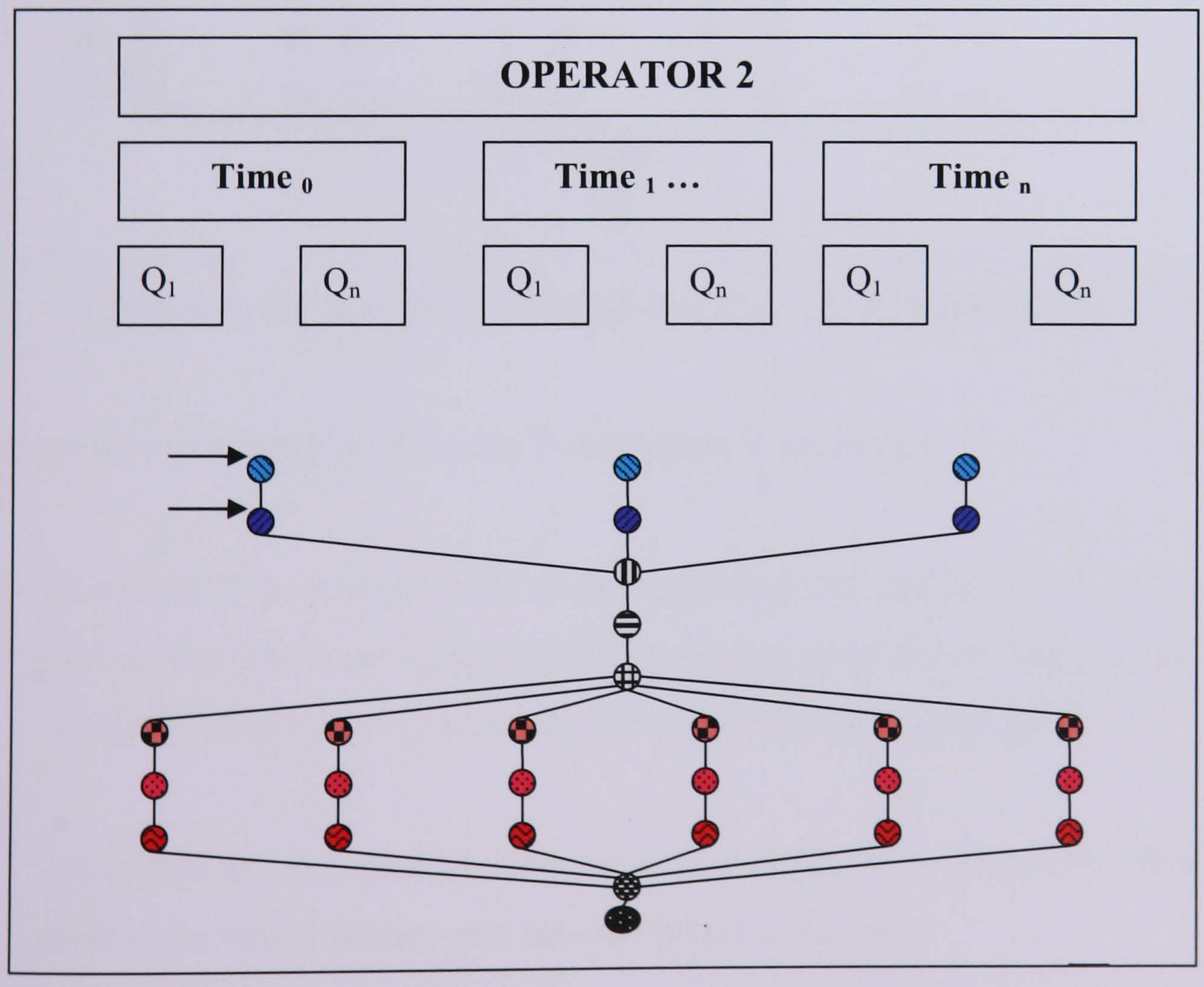


Figure 5.9: The Global Evaluation Plan of Operator2.

The dots marked with arrows in Figure 5.9 are the same as in Operator1 plan so they can be melted. The red dots of Operator2 are appended to those of Operator1's because they can not be melted as shown in Figure 5.10.

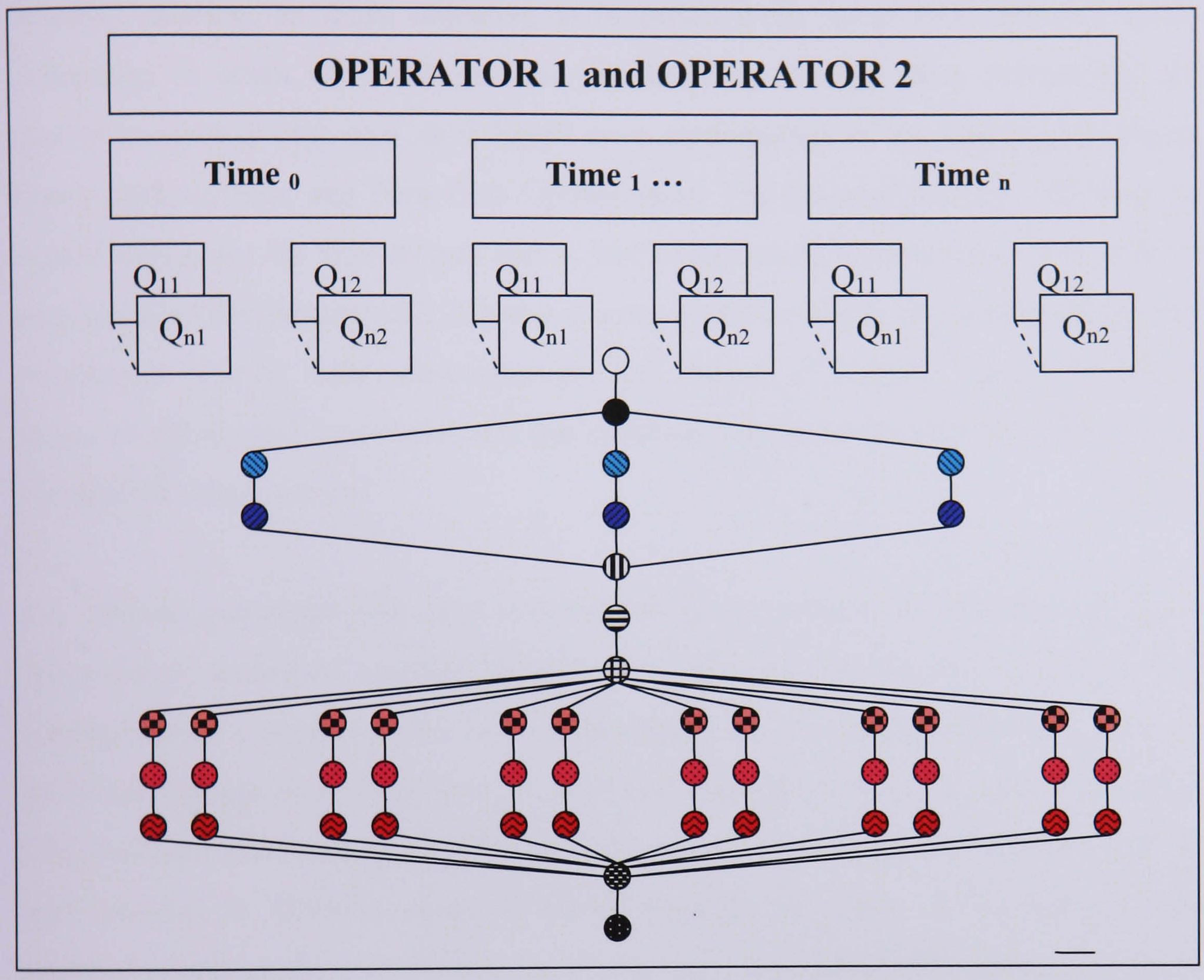


Figure 5.10: The Global Evaluation Plan of Both Operators.

In this global execution plan there are 3 categories of operators:

- Those that are executed once at the beginning and are later reused by all the consequent execution plans whether by operators (static or continuous), single or multiple predicates (operators), or within one continuous query.

- Those that are executed at every new time instant then reused by other plans until a new time instance gets into the picture and starts.

- Those that are executed for each query no matter what the type of query is.

5.3 Query Melting Paradigm

The Query Melting Paradigm (QMP) is a generic optimized framework for GIS. It aims at building the optimal global evaluation plan for GIS spatio-temporal dynamic complex queries. Its main objective is to produce the most cost effective query processing in terms of execution time and memory storage, thus, minimizing the queries execution time cost. It is based on a combination of the Sharing paradigm, Query Optimization, and Time Cost Optimization. The usage of the term "Melting" is coined especially for this project and is not a commonly known term. It is used to conceptualize and describe the fact that queries could be totally or partially eliminated resulting in sharing areas, time intervals, and objects of interest. Hence, to "Melt" means to eliminated repetitions and use common results of the operations that were executed by other queries.

The Sharing paradigm that was investigated in the review of literature of query optimization strategies [And06, Mok05a] is done by sharing the cache memory whereas here it is applied differently and is used at the low-level of processing, e.g., at the programming or coding level. It performs common sub-expression elimination CSE, that was discussed in [Kan94, Mok03], by eliminating all the repetitions of the same process in different plans. It allows sharing the object of interest between multiple simple queries by reading the object once and using it for many queries. In the work done by [And03] the object of interest was shared between multiple users each using one operator only in non dynamic queries, whereas it is applied here to share it between multiple simple queries with multiple operators that belong to the current dynamic complex query. When two queries share the same spatial area or when an area is included in another, the QMP uses the shared area for multiple queries. This is done by drawing the buffer of an area only once and using it multiple times to clip or find the objects of multiple simple queries similarly to the work done in [And02a, And02b] but for multiple simple queries that belong to the same dynamic complex query instead of multiple users queries using one operator only.

Moreover, the QMP allows sharing an interval of time if it is included or equal to the interval of another query. This is a newly introduced sharing that was not addressed by any previously reviewed paper. Also, sharing the intermediate results is done

through using the same underlying space (map) for all the queries of the same user. This is done through executing the plan of each user as a thread. The same was implemented in [And02c, And02d, and And02e] but for multiple users using one operator in non dynamic queries. When one or more operations of a query are melted, the query is partially melted. When all the operations of a query are melted, the whole query is fully melted. Query Melting results in a smaller number of operations to execute, hence, saving execution time, producing faster results, and leading to cost effectiveness.

The Query Optimization that was applied by [Kan94, And01, Elm06, and Mok06] is performed through converting the queries into a list of executable operations. The queries are decomposed, common sub-expressions are eliminated, the execution of queries is reordered, a global access plan is constructed, and the queries are translated into low-level programming. The Query Optimization is here extended to include "multi-user spatio-temporal multi-predicate dynamic complex queries". Finally, The Time Cost Optimization (TCOP) that is responsible for handling the multi-users dynamic complex queries at the GIS server side in order to manage the ones that have similar scenarios, aims at sharing a previously melted plan instead of generating a new one for each query. The work done in [ELM06] proposes sharing sub-plans whereas here TCOP which is the new decision making mechanism implements sharing all the plans. In order to minimize the execution time of the processor TCOP is implemented using two different methods of processing. The first is "With No Decider TCOP" where the complex queries templates are melted for each complex query. The second is "With Decider TCOP" where the templates are melted once per scenario and stored in memory for later retrieval by similar consequent scenarios. The decision making mechanism that proves to be cost effective is employed.

Query Melting is implemented using the Query Melting Processor QMP which plays the role of an interpreter, optimizer and processor. It is a middleware software system located on the server. Its major function is to input user queries, optimize them based on the Query Melting Paradigm, generate global execution plan and execute it, produce the resulting maps, and send the output to the user. It is mainly concerned with GIS and location-based services where mobile users issue spatio-temporal

dynamic complex queries to a mobile GIS server via wireless channels and GPS systems.

5.3.1 The Components of the Query Melting Processor

The Query Melting Processor consists of a number of components as shown in Figure 5.11. The architecture shows the three major steps that dynamic complex queries have to pass through in order to produce a query global evaluation plan. The Preprocessor takes as an input the dynamic complex query, parses it into multiple simple queries, groups them by category, and finally sorts them, in the aim to apply the first stage of Query Optimization as described in [And01, Elm06, and Mok03]. It reads the operators templates along with their costs. The cost of a function is its estimated or expected execution time, and the total cost of the functions represents the cost of the operator. The new Decision Making Mechanism TCOP, which implements the new paradigm "sharing previously generated plans" instead of sharing only sub-plans as suggested by [ELM06], checks if the queries scenario has already been previously generated, if it has the queries are routed to use it, otherwise the QMP proceeds as follows. Each simple query is decomposed according to its operator template. Two-dimensional arrays are used to store the decompositions and their relative costs.

The idea of using a Sliding ruler in [Mou00] that was used to navigate through a propagation hierarchy tree is applied here to melt the repetitions that exist in multiple plans. First, the Query Melting Ruler 1 melts the templates functions that are shared among multiple simple queries of the same operator and among different operators in the aim to implement common sub-expression elimination, sharing sub-plans, and sharing the underlying space (map). Sharing functions was implemented in [Mok05a, Mok05b] for multiple users using one operator only whereas it is implemented here for multiple users multiple operators multiple simple queries per operator. Sharing sub-plans was suggested by [Elm06] whereas here sharing the whole plan is being implemented. Sharing the underlying space (map) was implemented by [Mok04a, Mok04,b, and Mok04c] for multiple users dynamic queries with one operator whereas it is implemented here for multiple users dynamic queries with multiple operators and multiple simple queries per operator. The Query Melting Ruler 1 works on a two-

dimensional plane, and its output is the Initial Evaluation Plan for *Time 0* of the whole dynamic complex query.

Second, the Query Melting Ruler 2 is responsible for implementing sharing the space and areas, sharing the time intervals, and sharing the object of interest. Sharing the space and areas occurs when multiple queries share the same spatial area or when an area is included in another. The Query Melting Ruler 2 draws the buffer of an area only once and uses it multiple times to clip or find the objects of other multiple simple queries similarly to the work done in [And02a, And02b] but for multiple simple queries that belong to the same dynamic complex query instead of non dynamic multiple users queries using one operator only. Sharing the interval of time is a new sharing paradigm hereby introduced because it was not considered in the previous work that was reviewed in the literature of the query optimization strategies. The Query Melting Ruler 2 allows sharing an interval of time if it is included or equal to the interval of other multiple queries. Finally, sharing the object of interest allows sharing the object of interest between multiple simple queries by reading the object table once and using it for many queries. In the work done by [And03] the object of interest was shared between multiple users each using one operator only in non dynamic queries, whereas it is applied here to share it between multiple simple queries with multiple operators that belong to the current dynamic complex query. The Query Melting Ruler 2 works on the same two-dimensional plane as the Query Melting Ruler 1, and its output is the Final Global Evaluation Plan for *Times 1...n* of the whole dynamic complex query.

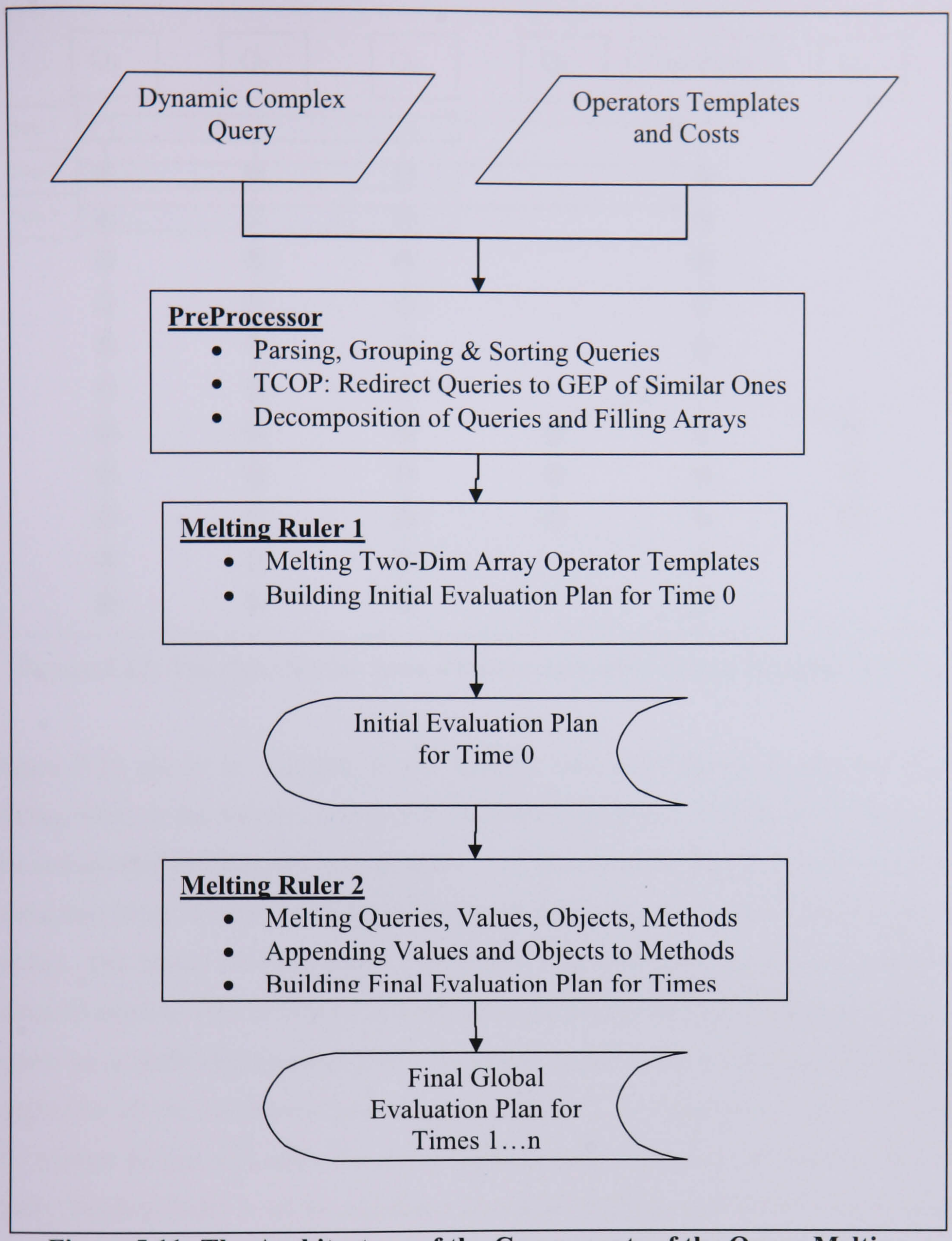


Figure 5.11: The Architecture of the Components of the Query Melting Processor.

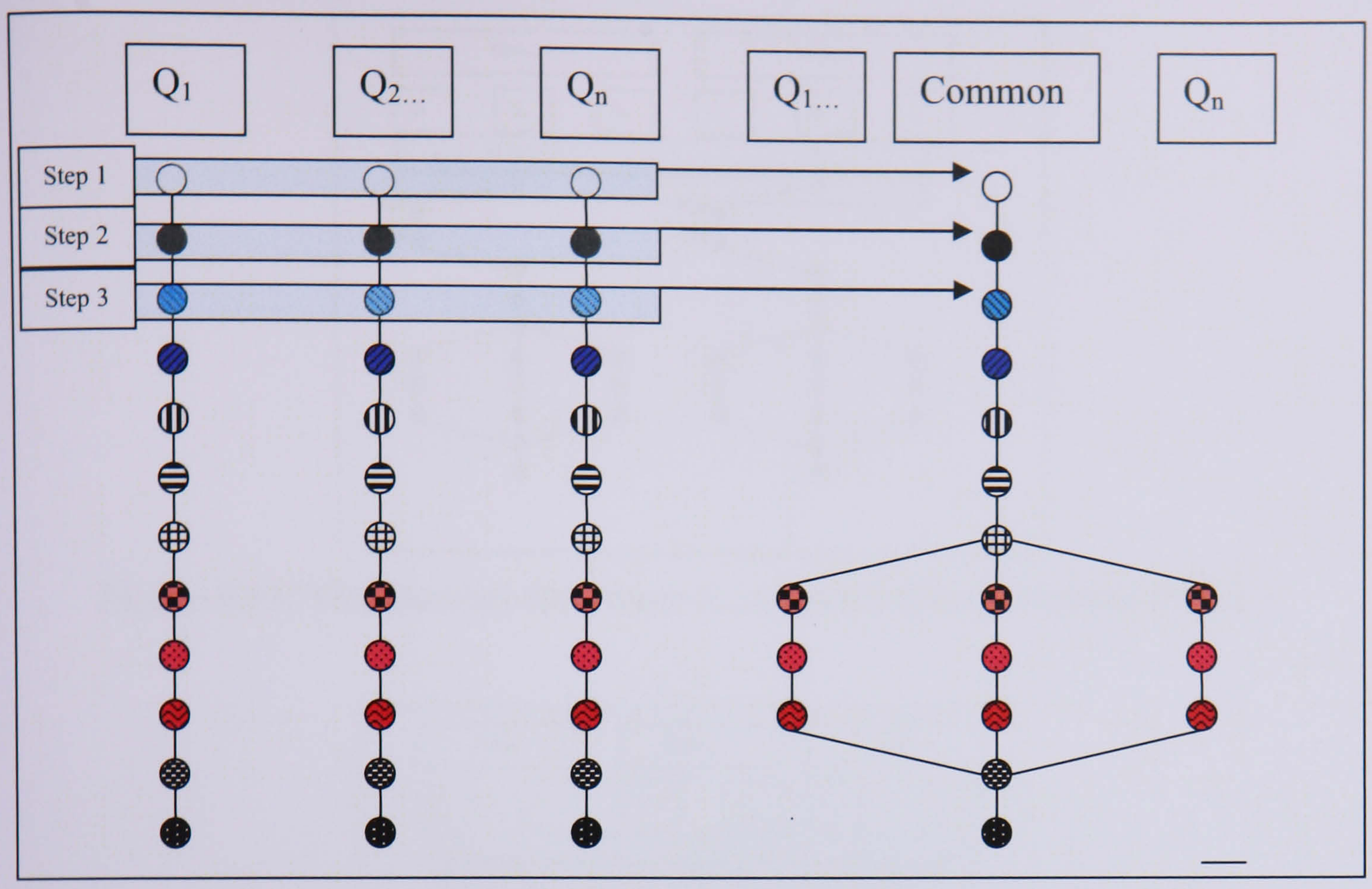


Figure 5.12: The Queries for *Time 0* before and after Query Melting Ruler 1.

Figure 5.12 shows an example of the melting ruler mechanism. It operates like a sliding ruler, in the top-down direction, melting each level's functions and it tackles the commonalities discussed in section 5.2. As shown in the figure, in step 1, it starts at the first level, in step 2 it tackles the second level, and so on until all the levels are melted. The Query Melting Ruler 2 melts the functions that are shared among the Initial Evaluation Plan of *Time 0* and the consequent time instances evaluation plan. It works on a multi-dimensional level. Its output is the Final Evaluation Plan that is applied for all the consequent time instances; *Times 1…n*. Figures 5.13 and 5.14 show the queries before and after the query melting ruler 2 process. It operates like the Query Melting Ruler 1, in the top-down direction, melting each level's functions but between the plan of *Time 0* and the plan of *Times 1…n*. As shown in the figure, in step 1, it starts at the first level, in step 2 it tackles the second level, and so on until all the levels are melted. Both the Initial Evaluation Plan and the Final Evaluation Plan are optimized during the Melting Rulers processes. Their result consists of a list of ordered and organized functions that are to be executed. When the execution of all the functions is done, the resulting map is the output of the dynamic complex query.

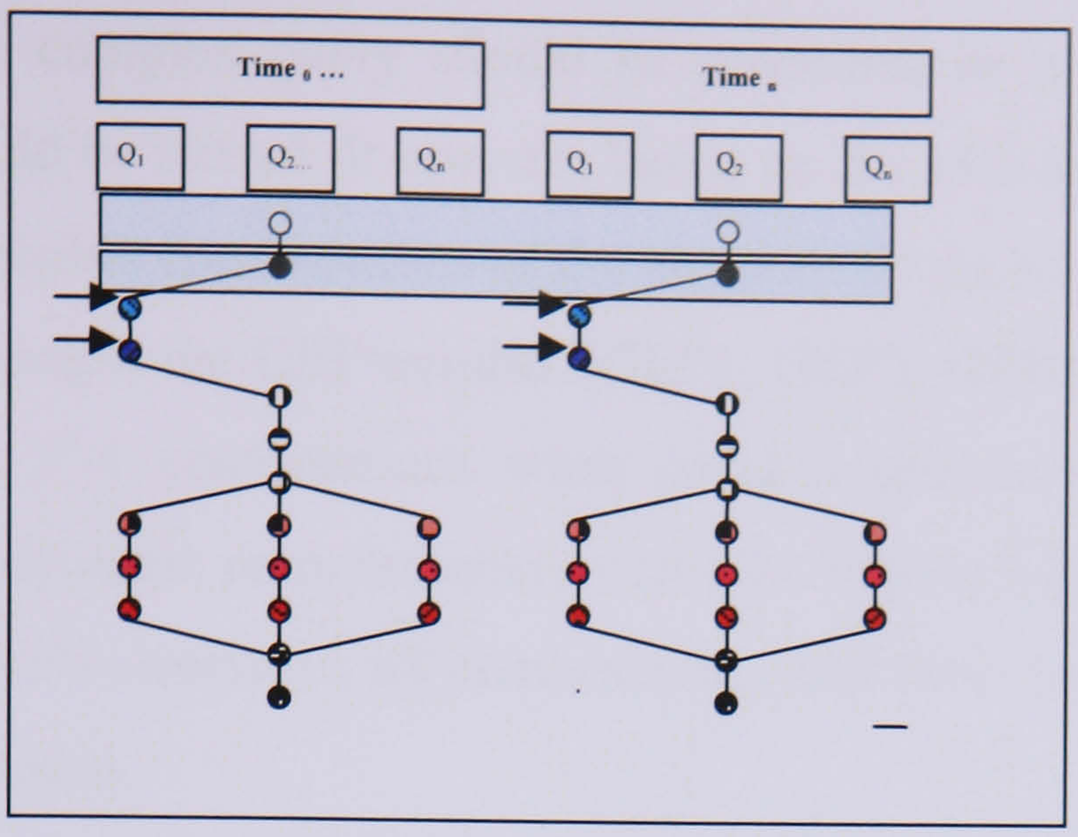


Figure 5.13: The Queries for *Times 1…n* before Query Melting Ruler 2.

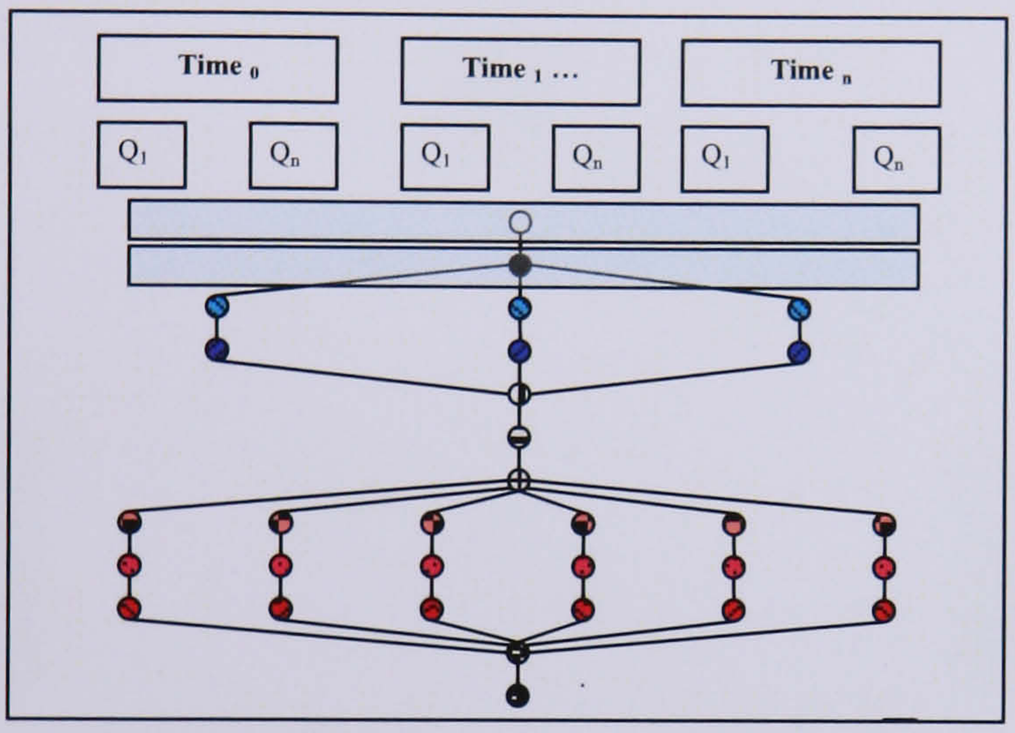


Figure 5.14: The Queries for *Times 1…n* after Query Melting Ruler 2.

5.3.2 TCOP: Decision Making Mechanism for Time Cost Optimization

In order to employ the new query optimization paradigm, Sharing GEP, which allows multiple users with similar query scenarios to share global execution plans of the melted templates of the operators, a decision making mechanism of time cost optimization TCOP is applied as part of the pre-processor. The major difference between TCOP and the work reviewed in the literature is that Elmongui in [Elm06] suggests sharing sub-plans only whereas TCOP allows sharing whole plans by multiple dynamic complex queries with similar scenarios. The main aim of the mechanism is to manage the similarities in the query scenarios, analyze them, and accordingly route the dynamic complex query either to a previously generated GEP of a similar query or to the query melting processor for processing and generating a new GEP for it. In other words, TCOP mechanism plays the role of a Decider, in terms of

whether a dynamic complex query should be processed or a previously generate templates GEP should be reused. It operates based on the idea of a decision tree that represents all the possible combinations of the elements of the set of operators and for each combination assigns the GEP number (GEP1, GEP2, GEP3, etc., and GEPn) to the leaf. There are 2^n-1 combinations when using n operators, each combination corresponding to a dynamic complex query scenario. Figure 5.15 shows an example of the decision tree of 4 operators, all combinations, and their corresponding 16 GEP represented as leaf boxes.

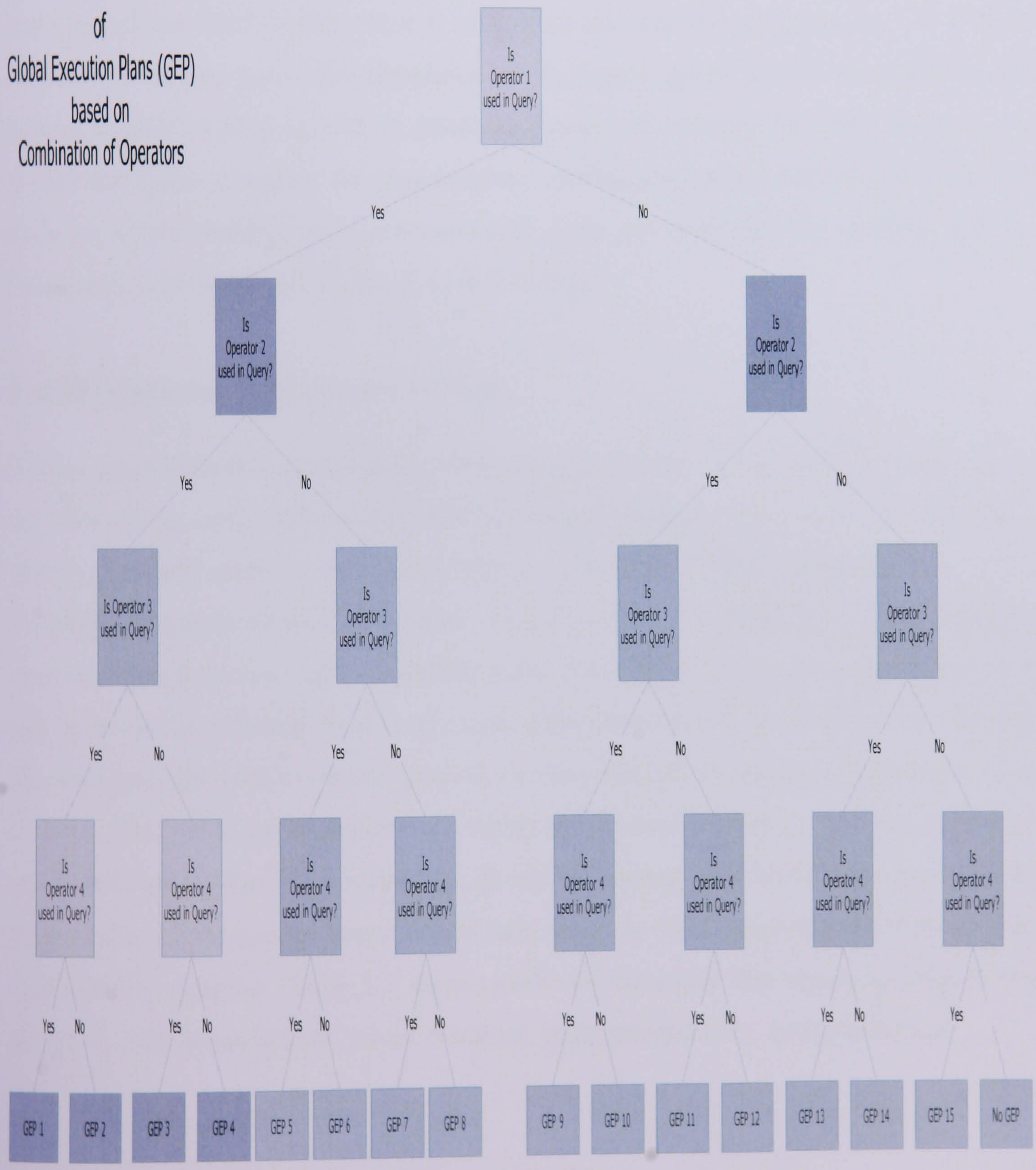


Figure 5.15: The Decision Tree of GEP based on Combination of Operators.

For each new dynamic complex query, the Decider starts by determining the list of operators used, proceeds with an examination of the similarities between the scenario of the current dynamic complex query and the scenario of the previously generated GEP, by checking if the first level operator of the decision tree is used in the current query, if yes it proceeds to the second level left side of the tree node, otherwise to the right side. This operation is repeated recursively until the leaf of the tree is reached which contains either a number (between 1 and 2^n-1) which means that a similar scenario of operators has been formulated earlier hence the current query should use their templates GEP or the value 0 which means that this scenario has never been formulated before, hence the templates of the current query should be melted, a new melted templates plan should be generated, stored in memory for later retrieval, and finally the leaf is assigned the plan number. At this point, the job of the Decider ends and the query melting processor proceeds with decomposing the queries, melting them, and generating the Global Evaluation Plan.

5.4 Mechanism of Execution Plan

Before describing the mechanisms of the execution plans, in the next few sections, we are describing some abbreviation for the representation of the icons of the IVQL which represent basically operators, values, and objects. Also we are showing some examples of simple as well as complex visual queries. The smiley icons that represent operators are listed in Table 5.1 along with their abbreviation and explanation. Any one icon can be selected first by the user when formulating a visual query. The first six operators are used for static queries, i.e. they ask about objects at the current *Time 0* only. The next six operators are used for dynamic queries, i.e. they launch a continuous query that lives *n* minutes. Every *m* minutes, they update the user with the new results of the queries based on his new location that is received through the GPS and satellite systems. Table 5.2 shows some of the icons that represent objects and facilities such as restaurant, motel, hospital, underground tube, and gymnasium.

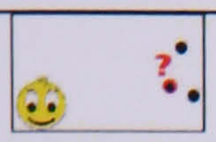
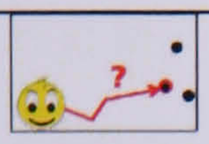
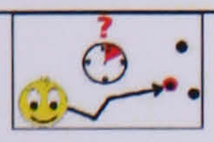
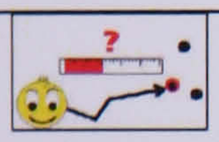
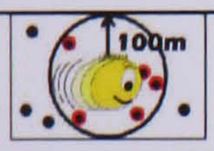
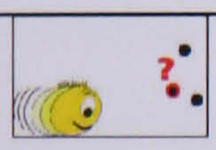
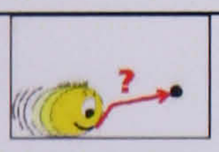
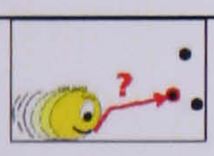
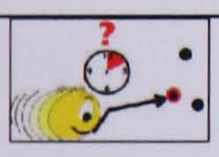
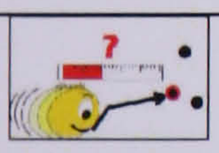
	Icon	Abbreviation	Explanation
1		SW	Find the facilities that are within a certain buffer
2		SN	Find the k nearest facilities kNN
3		SP	Find the shortest path
4		SNP	Find the k nearest facilities and their shortest paths
5		STL	Find the time left to reach the k nearest facilities
6		SDL	Find the distance left to reach the k nearest facilities
7		DW	Continuously find the facilities that are within a certain buffer on my way
8		DN	Continuously find the k nearest facilities kNN on my way
9		DP	Continuously find the shortest path on my way
10		DNP	Continuously find the k nearest facilities and their shortest paths on my way
11		DTL	For the next n minutes, keep on updating me continuously every m minutes with the time left to reach the k nearest facilities with t paths
12		DDL	For the next n minutes, keep on updating me continuously every m minutes with the distance left to reach the k nearest facilities

Table 5.1: The Icons Used as Operators to Formulate Dynamic Complex Queries.

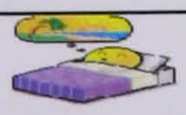
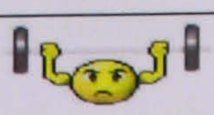
	Icon	Abbreviation	Explanation
1		R	Restaurant
2		M	Motel
3		H	Hospital
4		U	Tube / Metro
5		G	Gymnasium

Table 5.2: The Icons Used as Objects to Formulate Dynamic Complex Queries.

The AND operator  is used to combine multiple simple queries into a complex one.

Some examples of simple queries are:

[SN 1 M] find nearest 1 motel

[SNP 5 M] find 5 nearest motels with their shortest paths

[SW 500 R] find all restaurants that are within 500 meters from my location

[SW 1000 R] find all restaurants that are within 1000 meters from my location

[DTL 60 10G] continuously, find the Time Left to reach the nearest gymnasium for the next 60 minutes. Keep on supplying me with an updated result every 10 minutes. The result includes the shortest path.

An example of a dynamic complex query is shown in Table 5.3 summarizing a number of queries as follows. Find the nearest theatre and its shortest path. While I am on my way, keep on continuously supplying me with the 3 nearest motels and their shortest paths that are within 200 meters (or ahead of me, e.g., in half a circle buffer) until I reach the theatre or 60 minutes overlap. Update my map every 5 minutes.

[DTL 60 5 T]	[DNP 3 M]	[DW 200 M]

Table 5.3: A Dynamic Complex Query with 3 Predicates.

Table 5.4 shows another example of a complex query which is made up of six simple queries as follows. Find the Underground Stations that are within 300 meters from me, the restaurants that are within 100 meters, the 5 nearest hospitals, the 4 nearest schools with their paths, the 2 nearest Bus Stations, and the distance to reach the 7 nearest Police Office with their paths.

[SW 300 U]	[SW 100 R]	[SN 5 H]	[SNP 4 S]	[SN 2 B]	[SDL 7 P]

Table 5.4: A Static Complex Query with 4 Predicates.

5.4.1 Templates of Operators

Each operator is decomposed into a set of steps according to the operator's template. Figure 5.16 shows the actual template of the operator "Find the k Nearest Facilities".

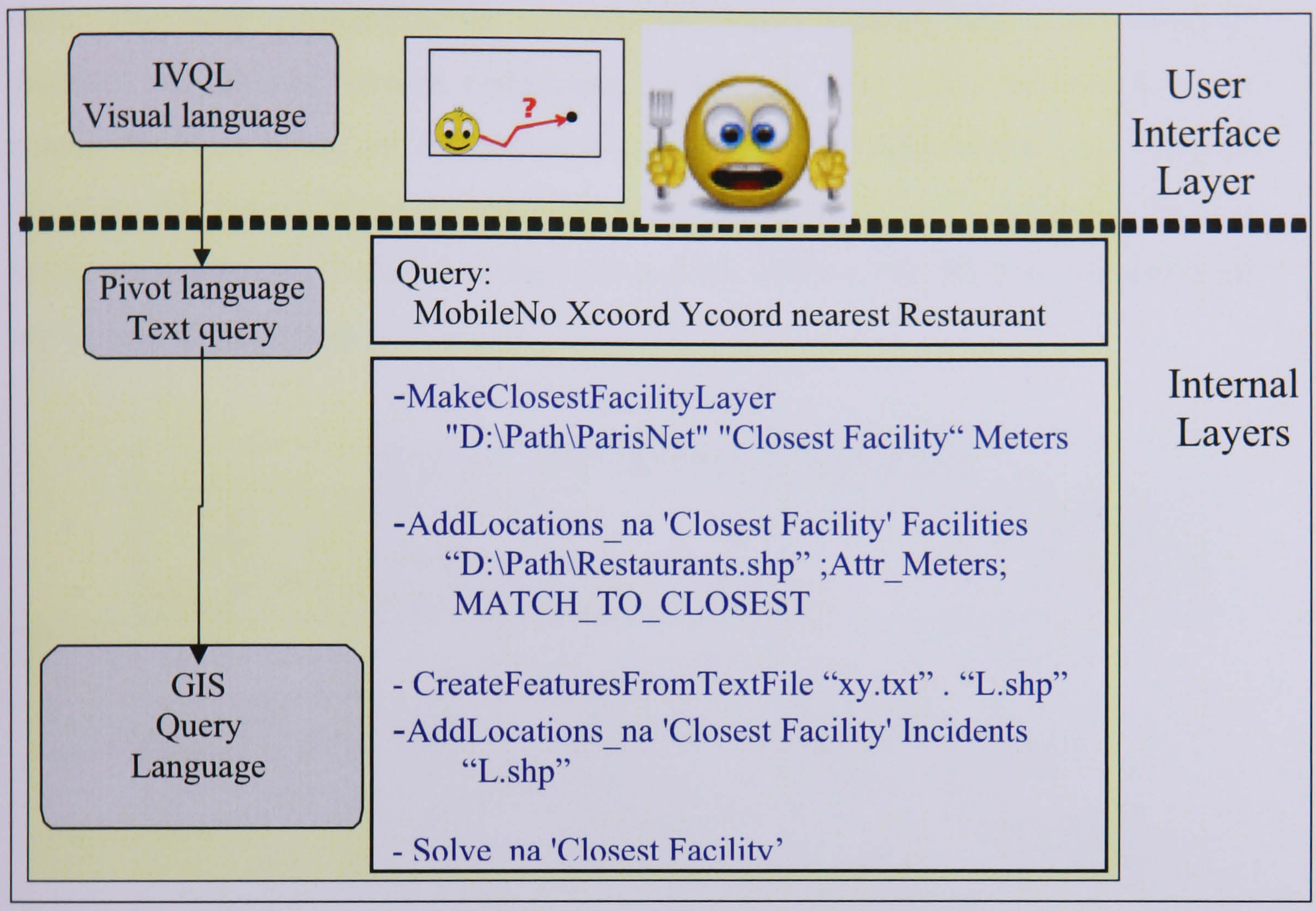


Figure 5.16: The Actual Template of the Operator "Find k Nearest Facilities".

The first step is to create a new layer used for finding the nearest objects. The second step is to add the facilities from the database table. The third is to add the XY location of the user as an incident. Finally, issue the command Solve that finds the nearest object. As can be seen in the Internal Layers of Figure 5.16, the Commands of each step are too long therefore, they will be abbreviated hereafter. The templates of some of the static operators are shown in Table 5.5 where each static operator is performed once only for *Time 0*. The steps of each operator are executed in the order in which they appear in the template following the top-down direction. The steps that appear empty mean that at this particular stage, there are no steps to do. Usually, the stages are reserved for the dynamic operator preparations such as create a new watcher and a new timer, etc. The MakeLayer function creates a new closest facility layer that is used to find the nearest objects. The ReadXY function reads the XY location of the

user. The AddXYIncident function adds the user's location as an incident to the layer. The Path is set to YES so as to produce the shortest path and set to No if no path is required. The NFacilities is the number of facilities needed to look for. The Impedance is set to Meters, Pedestrian-Time, or Drive-Time according to each operator which shows the result of the path either in meters, walking time, or drive time. The AddFacilities function specifies the object database table from which the facilities are selected such as restaurants, motels, etc. The Solve function finds the nearest facilities based on the previous parameters. The AddToMap and SendMap functions add the results to the map and send it to the user. The DrawBuffer draws a circle around the XY location of the user and the ClipBufFac finds the facilities that are located in this buffer.

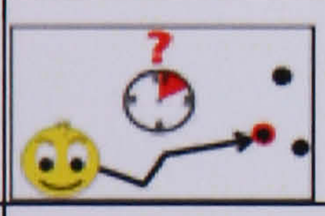
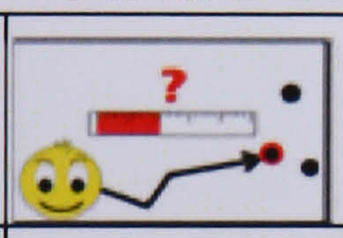
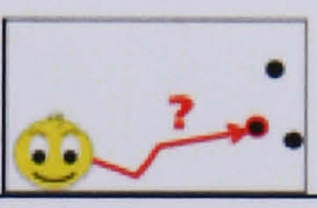
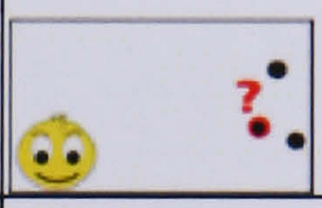
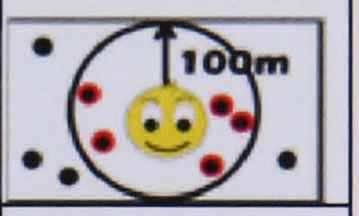
	STATIC OPERATORS TEMPLATES				
					
Step 1					
Step 2					
Step 3	MakeLayer	MakeLayer	MakeLayer	MakeLayer	
Step 4	ReadXY	ReadXY	ReadXY	ReadXY	ReadXY
Step 5	AddXYIncident	AddXYIncident	AddXYIncident	AddXYIncident	
Step 6	Path=Yes	Path=Yes	Path=Yes	Path=No	DrawBuffer
Step 7	NFacilities=1	NFacilities=1	NFacilities	NFacilities	ClipBufFac
Step 8	Impedance=T	Impedance=M	Impedance=M	Impedance=M	
Step 9	AddFacilities	AddFacilities	AddFacilities	AddFacilities	
Step 10	Solve	Solve	Solve	Solve	
Step 11	AddToMap	AddToMap	AddToMap	AddToMap	AddToMap
Step 12	SendMap	SendMap	SendMap	SendMap	SendMap
Step 13					

Table 5.5: The Templates of the Static Operators.

The templates of some of the dynamic operators are shown in Table 5.6. Each of the dynamic operators has two templates. The first is applied for *Time 0* when the query is launched and the second is applied for each of the consequent time instances *Times 1...n*. The NewWatcher waits for new XY Locations to arrive then launches triggers to read the data and continue execution accordingly. The NewTimer launches the query life time which triggers a timer that lasts as long as required by the user. The KillQueryIfX is used to terminate the query if it is expired, the user reaches his

destination, the user issues a cancellation order, or he gets disconnected. The rest of the functions operate the same as the static operators.

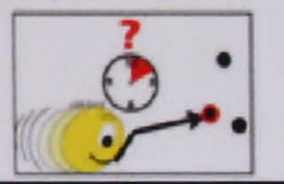
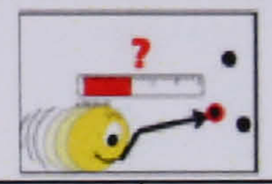
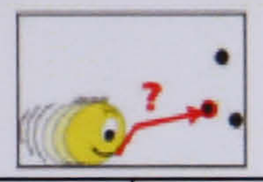
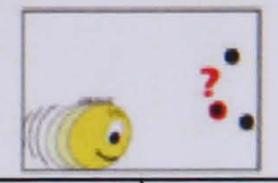
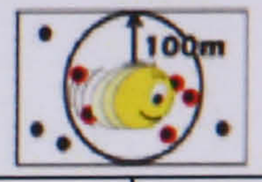
	DYNAMIC OPERATORS TEMPLATES									
										
	Time 0	Times 1...n	Time 0	Times 1...n	Time 0	Times 1...n	Time 0	Times 1...n	Time 0	Times 1...n
Step 1	NewWatcher		NewWatcher		NewWatcher		NewWatcher		NewWatcher	
Step 2	NewTimer		NewTimer		NewTimer		NewTimer		NewTimer	
Step 3	MakeLayer	MakeLayer	MakeLayer	MakeLayer	MakeLayer	MakeLayer	MakeLayer	MakeLayer		
Step 4	ReadXY	ReadXY	ReadXY	ReadXY	ReadXY	ReadXY	ReadXY	ReadXY	ReadXY	ReadXY
Step 5	AddXYIncident	AddXYIncident	AddXYIncident	AddXYIncident	AddXYIncident	AddXYIncident	AddXYIncident	AddXYIncident		
Step 6	Path=Yes	Path=Yes	Path=Yes	Path=Yes	Path=Yes	Path=Yes	Path=No	Path=No	DrawBuffer	DrawBuffer
Step 7	NFacilities=1	NFacilities=1	NFacilities=1	NFacilities=1	NFacilities	NFacilities	NFacilities	NFacilities	ClipBufFac	ClipBufFac
Step 8	Impedance=T	Impedance=T	Impedance=M	Impedance=M	Impedance=M	Impedance=M	Impedance=M	Impedance=M		
Step 9	AddFacilities	AddFacilitiesP	AddFacilities	AddFacilities	AddFacilities	AddFacilities	AddFacilities	AddFacilities		
Step 10	Solve	Solve	Solve	Solve	Solve	Solve	Solve	Solve		
Step 11	AddToMap	AddToMap	AddToMap	AddToMap	AddToMap	AddToMap	AddToMap	AddToMap	AddToMap	AddToMap
Step 12	SendMap	SendMap	SendMap	SendMap	SendMap	SendMap	SendMap	SendMap	SendMap	SendMap
Step 13	KillQueryIfX		KillQueryIfX		KillQueryIfX		KillQueryIfX		KillQueryIfX	

Table 5.6: The Templates of the Dynamic Operators.

5.4.2 Query Melting Process

The Query Melting process is performed by the preprocessor and the query melting rulers. The preprocessor groups the simple queries by category (operator), sorts them, creates an array for each query according to the template of its operator. If an element of the template ends with a star *, the corresponding elements in the arrays are filled with objects of the query such as AddFacilities* leads to AddRestaurant, AddHospital, etc. as shown in Step 9 of Table 5.7. If an element ends with two starts **, the corresponding elements are increasingly numbered starting with 1 such as Solve** leads to Solve1, Solve2, and so on as shown in Step10 of Table 5.7. If an element ends with ***, the corresponding elements are increased by 1 for every new time instance such as ReadXY*** leads ReadXY1 for *Time 0* and ReadXY2 for *Times 1...n* as shown in Step 4 of Table 5.12. Using ** at the end of an element means that this particular element is repeated and executed for each simple query whether static or dynamic, whether for *Time 0* or *Times 1...n*. Using *** means that this element is executed at every new time instance only. This reflects an update in the user XY location. Table 5.7 shows the arrays of four queries with the same static operator STL which is to find the time left to reach the nearest object. The objects of the query are respectively Restaurant, Hospital, Motel, and Gym. The preprocessor filled the array of each query according to the STL template shown in yellow.

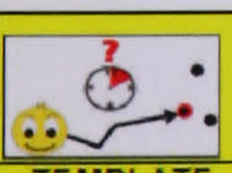
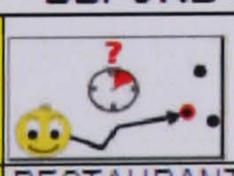
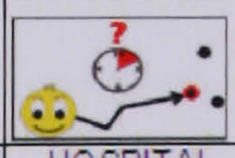
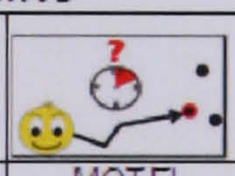
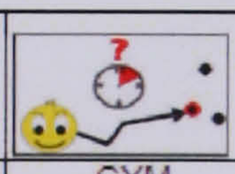
MULTIPLE QUERIES WITH ONE STATIC OPERATOR BEFORE QUERY MELTING					
STL					
	TEMPLATE	RESTAURANT	HOSPITAL	MOTEL	GYM
Step 1					
Step 2					
Step 3	MakeLayer	MakeLayer	MakeLayer	MakeLayer	MakeLayer
Step 4	ReadXY	ReadXY	ReadXY	ReadXY	ReadXY
Step 5	AddXYIncident	AddXYIncident	AddXYIncident	AddXYIncident	AddXYIncident
Step 6	Path=Yes	Path=Yes	Path=Yes	Path=Yes	Path=Yes
Step 7	NFacilities=1	NFacilities=1	NFacilities=1	NFacilities=1	NFacilities=1
Step 8	Impedance=T	Impedance=T	Impedance=T	Impedance=T	Impedance=T
Step 9	AddFacilities*	AddRestaurant	AddHospital	AddMotel	AddGym
Step 10	Solve**	Solve1	Solve2	Solve3	Solve4
Step 11	AddToMap**	AddToMap1	AddToMap2	AddToMap3	AddToMap4
Step 12	SendMap	SendMap	SendMap	SendMap	SendMap
Step 13					

Table 5.7: Multiple Queries with One Static Operator before Query Melting.

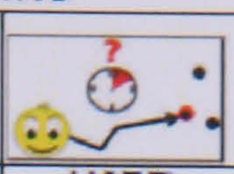
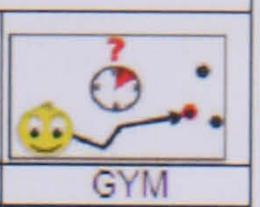
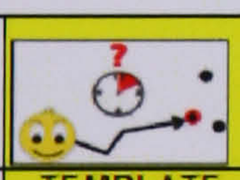
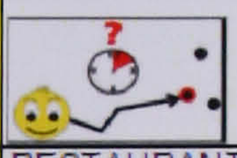
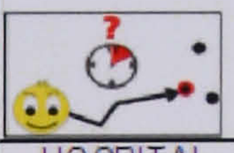
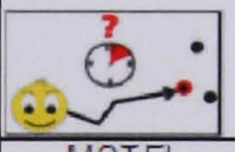
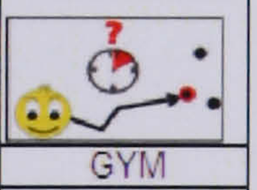
MULTIPLE QUERIES WITH ONE STATIC OPERATOR BEFORE QUERY MELTING					
STL					
	TEMPLATE	RESTAURANT	HOSPITAL	MOTEL	GYM
Step 1					
Step 2					
Step 3	MakeLayer	MakeLayer	MakeLayer	MakeLayer	MakeLayer
Step 4	ReadXY	ReadXY	ReadXY	ReadXY	ReadXY
Step 5	AddXYIncident	AddXYIncident	AddXYIncident	AddXYIncident	AddXYIncident
Step 6	Path=Yes	Path=Yes	Path=Yes	Path=Yes	Path=Yes
Step 7	NFacilities=1	NFacilities=1	NFacilities=1	NFacilities=1	NFacilities=1
Step 8	Impedance=T	Impedance=T	Impedance=T	Impedance=T	Impedance=T
Step 9	AddFacilities*	AddRestaurant	AddHospital	AddMotel	AddGym
Step 10	Solve**	Solve1	Solve2	Solve3	Solve4
Step 11	AddToMap**	AddToMap1	AddToMap2	AddToMap3	AddToMap4
Step 12	SendMap	SendMap	SendMap	SendMap	SendMap
Step 13					

Table 5.8: Multiple Queries with One Static Operator during Query Melting.

The Melting Ruler follows the top down direction starting with Step 1 then Step 2, and so on. It eliminates all the repetitions of an element. Table 5.8 shows the melting ruler in Step 3 where MakeLayer is repeated four times. The ruler keeps the first occurrence and eliminates the rest as shown in Table 5.9, thus melting the unnecessary functions. The ruler slides down to each of the next rows in turn and melts the repetitions. The result of the Melting Ruler is a list of functions that are to be executed. This list is in fact the Global Evaluation Plan as shown in Table 5.9.

MULTIPLE QUERIES WITH ONE STATIC OPERATOR AFTER QUERY MELTING					
STL					
	TEMPLATE	RESTAURANT	HOSPITAL	MOTEL	GYM
Step 1					
Step 2					
Step 3	MakeLayer	MakeLayer			
Step 4	ReadXY	ReadXY			
Step 5	AddXYIncident	AddXYIncident			
Step 6	Path=Yes	Path=Yes			
Step 7	NFacilities=1	NFacilities=1			
Step 8	Impedance=T	Impedance=T			
Step 9	AddFacilities*	AddRestaurant	AddHospital	AddMotel	AddGym
Step 10	Solve**	Solve	Solve	Solve	Solve
Step 11	AddToMap**	AddToMap	AddToMap	AddToMap	AddToMap
Step 12	SendMap	SendMap			
Step 13					

GLOBAL EVALUATION PLAN
MakeLayer
ReadXY
AddXYIncident
Path=Yes
NFacilities=1
Impedance=T
AddRestaurant
Solve
AddToMap
AddHospital
Solve
AddToMap
AddMotel
Solve
AddToMap
AddGym
Solve
AddToMap
SendMap

Table 5.9: Multiple Queries with One Static Operator after Query Melting.

The query melting process is applied on all combinations of multiple queries such as with one static operator, multiple static operators, one dynamic operator, multiple dynamic operators, multiple static and multiple dynamic operators. Table 5.10 shows two static operators STL and SDL before query melting and Table 5.11 shows them after query melting with their corresponding Global Evaluation Plan. The repetitions that are melted by the ruler belong to the SDL MOTEL query. They are MakeLayer, ReadXY, AddXYIncident, Path=Yes, NFacilities=1, and SendMap.

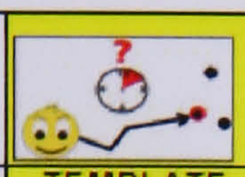
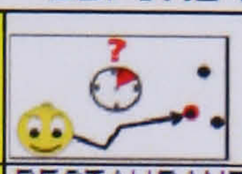
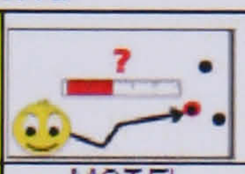
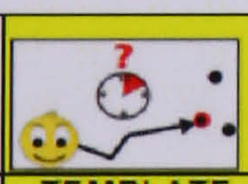
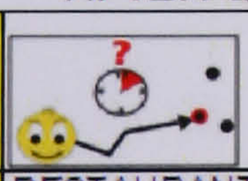
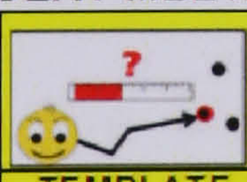
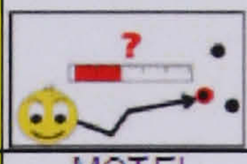
MULTIPLE QUERIES WITH MULTIPLE STATIC OPERATORS BEFORE QUERY MELTING					
STL AND SDL					
	TEMPLATE	RESTAURANT	TEMPLATE	MOTEL	
Step 1					
Step 2					
Step 3	MakeLayer	MakeLayer	MakeLayer	MakeLayer	
Step 4	ReadXY	ReadXY	ReadXY	ReadXY	
Step 5	AddXYIncident	AddXYIncident	AddXYIncident	AddXYIncident	
Step 6	Path=Yes	Path=Yes	Path=Yes	Path=Yes	
Step 7	NFacilities=1	NFacilities=1	NFacilities=1	NFacilities=1	
Step 8	Impedance=T	Impedance=T	Impedance=M	Impedance=M	
Step 9	AddFacilities*	AddRestaurant	AddFacilities*	AddMotel	
Step 10	Solve**	Solve1	Solve**	Solve2	
Step 11	AddToMap**	AddToMap1	AddToMap**	AddToMap2	
Step 12	SendMap	SendMap	SendMap	SendMap	
Step 13					

Table 5.10: Multiple Queries with Multiple Static Operators before Query Melting.

MULTIPLE QUERIES WITH MULTIPLE STATIC OPERATOR AFTER QUERY MELTING					
STL AND SDL					
	TEMPLATE	RESTAURANT	TEMPLATE	MOTEL	
Step 1					
Step 2					
Step 3	MakeLayer	MakeLayer	MakeLayer		
Step 4	ReadXY	ReadXY	ReadXY		
Step 5	AddXYIncident	AddXYIncident	AddXYIncident		
Step 6	Path=Yes	Path=Yes	Path=Yes		
Step 7	NFacilities=1	NFacilities=1	NFacilities=1		
Step 8	Impedance=T	Impedance=T	Impedance=M	Impedance=M	
Step 9	AddFacilities*	AddRestaurant	AddFacilities*	AddMotel	
Step 10	Solve**	Solve	Solve**	Solve	
Step 11	AddToMap**	AddToMap	AddToMap**	AddToMap	
Step 12	SendMap	SendMap	SendMap		
Step 13					

GLOBAL EVALUATION PLAN
MakeLayer
ReadXY
AddXYIncident
Path=Yes
NFacilities=1
Impedance=T
AddRestaurant
Solve
AddToMap
Impedance=M
AddMotel
Solve
AddToMap
SendMap

Table 5.11: Multiple Queries with Multiple Static Operators after Query Melting.

Table 5.12 shows multiple dynamic operators before query melting. The query DTL RESTAURANT is to continuously find the time left to reach the nearest restaurant. The query DNP 3 RESTAURANT is to continuously find the 3 nearest restaurants with their corresponding shortest paths while the user is moving. Both queries live and run for a period of time and watch for each new XY location updates of the user in order to reproduce accordingly the new results for him. The template of each dynamic query contains one plan for *Time 0* and another one for *Times 1...n*. The preprocessor fills that array in the same way discussed earlier.

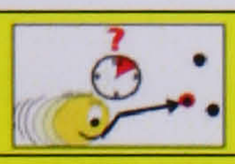
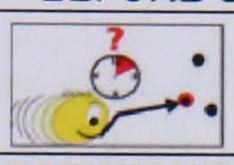
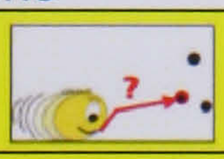
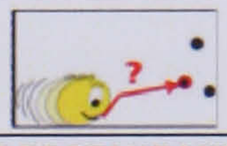
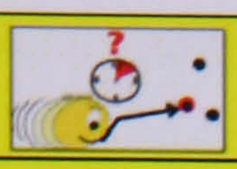
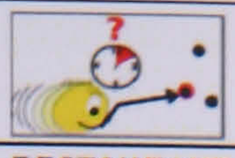
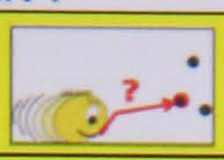
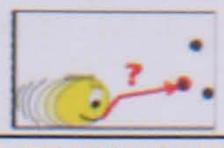
MULTIPLE QUERIES WITH MULTIPLE DYNAMIC OPERATORS BEFORE QUERY MELTING								
DTL AND DNP								
	TEMPLATE		RESTAURANT		TEMPLATE		3 RESTAURANT	
	Time 0	Times 1...n	Time 0	Times 1...n	Time 0	Times 1...n	Time 0	Times 1...n
Step 1	NewWatcher		NewWatcher		NewWatcher		NewWatcher	
Step 2	NewTimer		NewTimer		NewTimer		NewTimer	
Step 3	MakeLayer		MakeLayer		MakeLayer		MakeLayer	
Step 4	ReadXY***	ReadXY***	ReadXY1	ReadXY2	ReadXY***	ReadXY***	ReadXY1	ReadXY2
Step 5	AddXYIncid***	AddXYIncid***	AddXYIncid1	AddXYIncid2	AddXYIncid***	AddXYIncid***	AddXYIncid1	AddXYIncid2
Step 6	Path=Yes	Path=Yes	Path=Yes	Path=Yes	Path=Yes	Path=Yes	Path=Yes	Path=Yes
Step 7	NFacilities=1	NFacilities=1	NFacilities=1	NFacilities=1	Nfacilities=*	Nfacilities=*	Nfacilities=3	Nfacilities=3
Step 8	Impedance=T	Impedance=T	Impedance=T	Impedance=T	Impedance=M	Impedance=M	Impedance=M	Impedance=M
Step 9	AddFacilities*	AddFacilities*	AddRestaurant	AddRestaurant	AddFacilities*	AddFacilities*	AddRestaurant	AddRestaurant
Step 10	Solve**	Solve**	Solve1	Solve2	Solve**	Solve**	Solve3	Solve4
Step 11	AddToMap**	AddToMap**	AddToMap1	AddToMap2	AddToMap	AddToMap	AddToMap3	AddToMap4
Step 12	SendMap	SendMap	SendMap	SendMap	SendMap	SendMap	SendMap	SendMap
Step 13	KillQueryIfX	KillQueryIfX	KillQueryIfX	KillQueryIfX	KillQueryIfX	KillQueryIfX	KillQueryIfX	KillQueryIfX

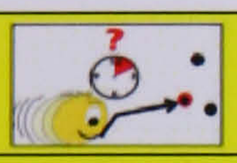
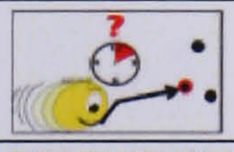
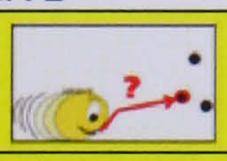
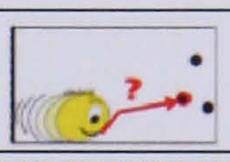
Table 5.12: Multiple Queries with Multiple Dynamic Operators before Query Melting.

The Melting Ruler 1 tackles *Plans 0* of all the queries. *Plan 0* of DTL RESTAURANT is melted with *Plan 0* of DNP 3 RESTAURANT. The melted elements of the query DNP 3 RESTAURANT are NewWatcher, NewTimer, MakeLayer, ReadXY1, AddXYIncident, Path=Yes, AddRestaurant, SendMap, and KillQueryIfX. Figure 5.13 shows the melted functions in red under the column entitled *Time 0* of the DNP operator. The Melting Ruler 2 tackles *Plans 1...n* of all the queries. *Plan 1...n* of DTL RESTAURANT is melted with *Plan 1...n* of DNP 3 RESTAURANT. The melted elements of the query DNP 3 RESTAURANT are ReadXY2, AddXYIncid, Path=Yes, AddRestaurant, SendMap, and KillQueryIfX. Figure 5.14 shows the melted functions in blue under the column entitled *Times 1...n* of the DNP operator.

MULTIPLE QUERIES WITH MULTIPLE DYNAMIC OPERATORS QUERY MELTING RULER 1								
DTL AND DNP								
	TEMPLATE		RESTAURANT		TEMPLATE		3 RESTAURANT	
	Time 0	Times 1...n	Time 0	Times 1...n	Time 0	Times 1...n	Time 0	Times 1...n
Step 1	NewWatcher		NewWatcher		NewWatcher		NewWatcher	
Step 2	NewTimer		NewTimer		NewTimer		NewTimer	
Step 3	MakeLayer		MakeLayer		MakeLayer		MakeLayer	
Step 4	ReadXY***	ReadXY***	ReadXY1	ReadXY2	ReadXY***	ReadXY***	ReadXY1	ReadXY2
Step 5	AddXYIncid***	AddXYIncid***	AddXYIncid1	AddXYIncid2	AddXYIncid***	AddXYIncid***	AddXYIncid1	AddXYIncid2
Step 6	Path=Yes	Path=Yes	Path=Yes	Path=Yes	Path=Yes	Path=Yes	Path=Yes	Path=Yes
Step 7	NFacilities=1	NFacilities=1	NFacilities=1	NFacilities=1	Nfacilities=*	Nfacilities=*	Nfacilities=3	Nfacilities=3
Step 8	Impedance=T	Impedance=T	Impedance=T	Impedance=T	Impedance=M	Impedance=M	Impedance=M	Impedance=M
Step 9	AddFacilities*	AddFacilities*	AddRestaurant	AddRestaurant	AddFacilities*	AddFacilities*	AddRestaurant	AddRestaurant
Step 10	Solve**	Solve**	Solve1	Solve2	Solve**	Solve**	Solve3	Solve4
Step 11	AddToMap**	AddToMap**	AddToMap1	AddToMap2	AddToMap	AddToMap	AddToMap3	AddToMap4
Step 12	SendMap	SendMap	SendMap	SendMap	SendMap	SendMap	SendMap	SendMap
Step 13	KillQueryIfX	KillQueryIfX	KillQueryIfX	KillQueryIfX	KillQueryIfX	KillQueryIfX	KillQueryIfX	KillQueryIfX

Ruler 1: Melting Time 0 Plans of all the Queries

Table 5.13: Multiple Queries with Multiple Dynamic Operators during Query Melting Ruler 1.

MULTIPLE QUERIES WITH MULTIPLE DYNAMIC OPERATORS QUERY MELTING RULER 2								
DTL AND DNP								
	TEMPLATE		RESTAURANT		TEMPLATE		3 RESTAURANT	
	Time 0	Times 1...n	Time 0	Times 1...n	Time 0	Times 1...n	Time 0	Times 1...n
Step 1	NewWatcher		NewWatcher		NewWatcher			
Step 2	NewTimer		NewTimer		NewTimer			
Step 3	MakeLayer		MakeLayer		MakeLayer			
Step 4	ReadXY***	ReadXY***	ReadXY1	ReadXY2	ReadXY***	ReadXY***		ReadXY2
Step 5	AddXYIncid***	AddXYIncid***	AddXYIncid1	AddXYIncid2	AddXYIncid***	AddXYIncid***		AddXYIncid2
Step 6	Path=Yes	Path=Yes	Path=Yes	Path=Yes	Path=Yes	Path=Yes		Path=Yes
Step 7	NFacilities=1	NFacilities=1	NFacilities=1	NFacilities=1	Nfacilities=*	Nfacilities=*	Nfacilities=3	Nfacilities=3
Step 8	Impedance=T	Impedance=T	Impedance=T	Impedance=T	Impedance=M	Impedance=M	Impedance=M	Impedance=M
Step 9	AddFacilities*	AddFacilities*	AddRestaurant	AddRestaurant	AddFacilities*	AddFacilities*		AddRestaurant
Step 10	Solve**	Solve**	Solve1	Solve2	Solve**	Solve**	Solve3	Solve4
Step 11	AddToMap**	AddToMap**	AddToMap1	AddToMap2	AddToMap	AddToMap	AddToMap3	AddToMap4
Step 12	SendMap	SendMap	SendMap	SendMap	SendMap	SendMap		SendMap
Step 13	KillQueryIfX	KillQueryIfX	KillQueryIfX	KillQueryIfX	KillQueryIfX	KillQueryIfX		KillQueryIfX

Ruler 2: Melting Times 1...n Plans of all the Queries

Table 5.14: Multiple Queries with Multiple Dynamic Operators during Query Melting Ruler 2.

Figure 5.15 shows the multiple queries with multiple operators after being melted. All the melted elements have been removed from their corresponding simple query plans. The execution of the remaining ones is enough to answer correctly the dynamic complex query. The remaining functions are cleaned from the temporary digits that were appended at their end during the query melting process. Then, they are filled in two new arrays, the first one consists of the *Plan 0* of all the melted simple queries

and the second the *Plan 1…n*. The resulting arrays are in fact the Global Evaluation Plan of *Time 0* and the Global Evaluation Plan of *Times 1*…n as shown in Table 5.16.

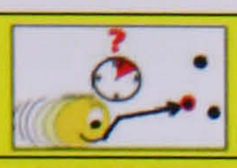
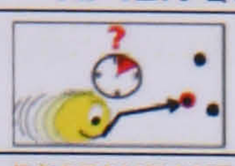
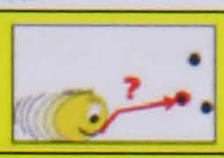
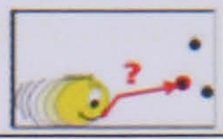
MULTIPLE QUERIES WITH MULTIPLE DYNAMIC OPERATORS AFTER QUERY MELTING								
DTL AND DNP								
	TEMPLATE		RESTAURANT		TEMPLATE		3 RESTAURANT	
	Time 0	Times 1…n	Time 0	Times 1…n	Time 0	Times 1…n	Time 0	Times 1…n
Step 1	NewWatcher		NewWatcher		NewWatcher			
Step 2	NewTimer		NewTimer		NewTimer			
Step 3	MakeLayer		MakeLayer		MakeLayer			
Step 4	ReadXY***	ReadXY***	ReadXY1	ReadXY2	ReadXY***	ReadXY***		
Step 5	AddXYIncid***	AddXYIncid***	AddXYIncid1	AddXYIncid2	AddXYIncid***	AddXYIncid***		
Step 6	Path=Yes	Path=Yes	Path=Yes	Path=Yes	Path=Yes	Path=Yes		
Step 7	NFacilities=1	NFacilities=1	NFacilities=1	NFacilities=1	Nfacilities=*	Nfacilities=*	Nfacilities=3	Nfacilities=3
Step 8	Impedance=T	Impedance=T	Impedance=T	Impedance=T	Impedance=M	Impedance=M	Impedance=M	Impedance=M
Step 9	AddFacilities*	AddFacilities*	AddRestaurant	AddRestaurant	AddFacilities*	AddFacilities*		
Step 10	Solve**	Solve**	Solve1	Solve2	Solve**	Solve**	Solve3	Solve4
Step 11	AddToMap**	AddToMap**	AddToMap1	AddToMap2	AddToMap	AddToMap	AddToMap3	AddToMap4
Step 12	SendMap	SendMap	SendMap	SendMap	SendMap	SendMap		
Step 13	KillQueryIfX	KillQueryIfX	KillQueryIfX	KillQueryIfX	KillQueryIfX	KillQueryIfX		

Table 5.15: Multiple Queries with Multiple Dynamic Operators after Query Melting.

GLOBAL EVALUATION PLAN T0
NewWatcher
NewTimer
MakeLayer
ReadXY
AddXYIncid
Path=Yes
NFacilities=1
Impedance=T
AddRestaurant
Solve
AddToMap
Nfacilities=3
Impedance=M
Solve
AddToMap
SendMap
KillQueryIfX

GLOBAL EVALUATION PLAN T1…Tn
ReadXY
AddXYIncid
Path=Yes
NFacilities=1
Impedance=T
AddRestaurant
Solve
AddToMap
Nfacilities=3
Impedance=M
Solve
AddToMap
SendMap
KillQueryIfX

Table 5.16: The Global Evaluation Plan for *Time 0* and Global Evaluation Plan for *Times 1…n*.

5.5 Conclusion

In this chapter we presented a thorough investigation into GIS operators that are related to proximity analysis and examined the commonalities between their execution plans and determined which common parts can be shared and which ones re-ordered. The investigation covered all types of operators namely the static operators, dynamic operators with one-predicate, and dynamic operators with multiple predicates. The investigated execution plans have proved to have significant

commonalities hence the sharing paradigm was employed in processing them. We introduced the new Query Melting paradigm which includes the sharing paradigm that was implemented in [And06, Mok05a] for non dynamic multiple users simple operator queries but extended it in this work to include dynamic multiple users multiple operators with multiple simple queries per operator queries, push-down approach that was implemented by [Elm06] to swap the Select and join operators but was extended in this work to reorganize all the operations in a global execution plan, traditional query optimization as described in [Kan94], and finally the new paradigm sharing the global execution plans (GEP) instead of sharing only sub-plans as suggested in [Elm06].

However, in order to employ query melting efficiently, the Query Melting Processor (QMP) for dynamic complex queries has been presented, its components described, and its process elaborated. QMP employs the new melting ruler mechanism that acts like a sliding ruler to allow sharing the functions of various templates, input data, intermediate results, underlying space maps, spatial areas, time intervals, objects of interest, and query results. The mechanism of the melting ruler was detailed using the templates of the operators as examples.

Finally, a new query optimization paradigm, called Sharing Global Execution Plans (GEP), has been introduced, which aims at sharing totally a previously generated GEP by multiple users who formulate similar scenarios of queries. However, in order to manage properly the similarities in the query scenarios, a new Decision Making Mechanism called time cost optimizer (TCOP) has also been introduced with the aim to optimize the time cost. The function of TCOP was explained as being to examine the operators used in the formulation of the dynamic complex queries, check for similarities between the current scenario of queries and previously formulated ones, and either redirect the query to a previously generated GEP or send it to the query processor that will construct for it a new GEP and store it in memory for later retrieval by other consequent similar scenario queries.

Chapter 6 - Design and Implementation

6.1 Introduction

The design and implementation of the Query Melting Processor play a major role in the life cycle of the system especially since QMP is implemented as a middleware on a GIS server and is responsible for processing a large-scale of dynamic complex queries in a multi-user environment. At the design stage, the system architecture is presented using a variety of diagrams such as the Use Case Diagram and the Time Sequence Table. The Use case Diagram is used to visualize the activities and tasks of a system showing at the same time the actor/actors associated with each activity. Each actor could be a person or a component in a system. The Time Sequence Table shows the life time of each activity as well as the time sequence in which the processes are executed. The implementation stage involves the actual development of the system. It shows and discusses the programming language with its environment, the algorithm of each process, and the schema of each table. It also describes the relation and association between the elements of the system. Moreover, in order to evaluate and prove the cost effectiveness of the Query Melting Processor a theoretical evaluation is conducted in order to determine 1) the cost and profit of each phase which reflects either a time saving meaning that it is cost effective or a time loss meaning that it is not cost effective and 2) the memory space that is occupied by each process of the Query Melting Processor (QMP) as well as by the Decision Making Mechanism (TCOP). Moreover, an experimental evaluation using a case study based on the map of Paris will be carried out (Chapter 7), in order to evaluate and prove that significant saving in time can be achieved by employing the newly developed strategies.

This chapter proceeds with presenting the design of the Query Melting Processor using a number of diagrams, elaborates the implementation of the Processor by showing the user interface, the algorithm of each process that is concerned with the melting process, the evaluation of the speed of each algorithm using the Big-Oh notation to determine the running time, and concludes with a summary of the query process. Section 6.2 is concerned with the Design of the Query Melting Processor using a number of diagrams such as the Time Sequence Diagram and the Use Case

Diagram in order to show which actor is associated with which activity. They also present the Architecture of the Query Melting Processor showing the input tables with their attributes, the variables, fields, or arrays that are stored in the memory, the processes or procedures that are executed, and the output of the processor. Section 6.3 presents the implementation of the Query Melting Processor starting with a description of the environment of the implementation and the User Interface of the Query Melting Processor. It is mainly concerned with the implementation algorithms of the processes of the Query Melting Processor where each process algorithm is presented and its computational cost is estimated using the Big-Oh notation. In section 6.4 a theoretical evaluation is done in to order to quantify the computation cost and the memory space of the Query Melting Processor as well as the time cost optimizer in terms of profit and loss. Finally, section 6.5 concludes with a summary of the design and implementation of the query melting processor that is based on the query melting paradigm, query optimization, and time cost optimization. The query melting paradigm includes sharing the functions, objects, spatial areas, time intervals. The query optimization includes decomposition of the queries, common sub-expression elimination, generation of global execution plan, and evaluation of the plan. The time cost optimization is performed through sharing the global execution plans.

6.2 Design of the Query Melting Processor

The design of the Query Melting Processor is done using an object-oriented visual modelling tool (SmartDraw). Each diagram visualizes a specific model that is related to a specific process of the query processing.

6.2.1 The Use Case of the Query Melting Processor

The Use Case Diagram is used to visualize the activities and tasks of a system showing at the same time the actor/actors of each activity. Each actor could be a person or a component in a system. The person could be a user, client, employee, officer, manager, etc., and the component could be a processor, a ruler, etc. The actors that are going to use the system appear always to the left side of the activities whereas the internal actors of the company or system appear to the right side. Each of the activities is represented by an oval and they are all listed sequentially in a rectangle.

To show that an actor is associated with a task, a link connects the actor to that particular task. The Query Melting Processor consists of a number of activities or tasks as shown in Figure 6.1. The first activity Read Templates is executed by the pre-processor that reads the templates of the operators from a data file and assigns them in arrays. The next activity Formulate Dynamic Query is performed by the mobile user when he launched a query. In the next one, the pre-processor takes as an input the dynamic complex query, parses it into multiple simple queries, groups them by category, and finally sorts them, in the aim to apply the first stage of Query Optimization as described in [And01, Elm06, and Mok03]. In the next task, the new Decision Making Mechanism TCOP, which implements the new paradigm "sharing previously generated plans" instead of sharing only sub-plans as suggested by [ELM06], checks if the queries scenario has already been previously generated, if it has the queries are routed to use it, otherwise the QMP proceeds with the next task. The main aim of the mechanism is to manage the similarities in the query scenarios, analyze them, and accordingly route the dynamic complex query either to a previously generated GEP of a similar query or to the query melting processor for processing and generating a new GEP for it.

In the next three tasks, the idea of using a Sliding ruler in [Mou00] that was used to navigate through a propagation hierarchy tree is applied here to melt the repetitions that exist in multiple plans. First, the Query Melting Ruler 1 melts the templates functions that are shared among multiple simple queries of the same operator and among different operators in the aim to implement common sub-expression elimination, sharing sub-plans, and sharing the underlying space (map). Sharing functions was implemented in [Mok05a, Mok05b] for multiple users using one operator only whereas it is implemented here for multiple users multiple operators multiple simple queries per operator. Sharing sub-plans was suggested by [Elm06] whereas here sharing the whole plan is being implemented. Sharing the underlying space (map) was implemented by [Mok04a, Mok04,b, and Mok04c] for multiple users dynamic queries with one operator whereas it is implemented here for multiple users dynamic queries with multiple operators and multiple simple queries per operator.

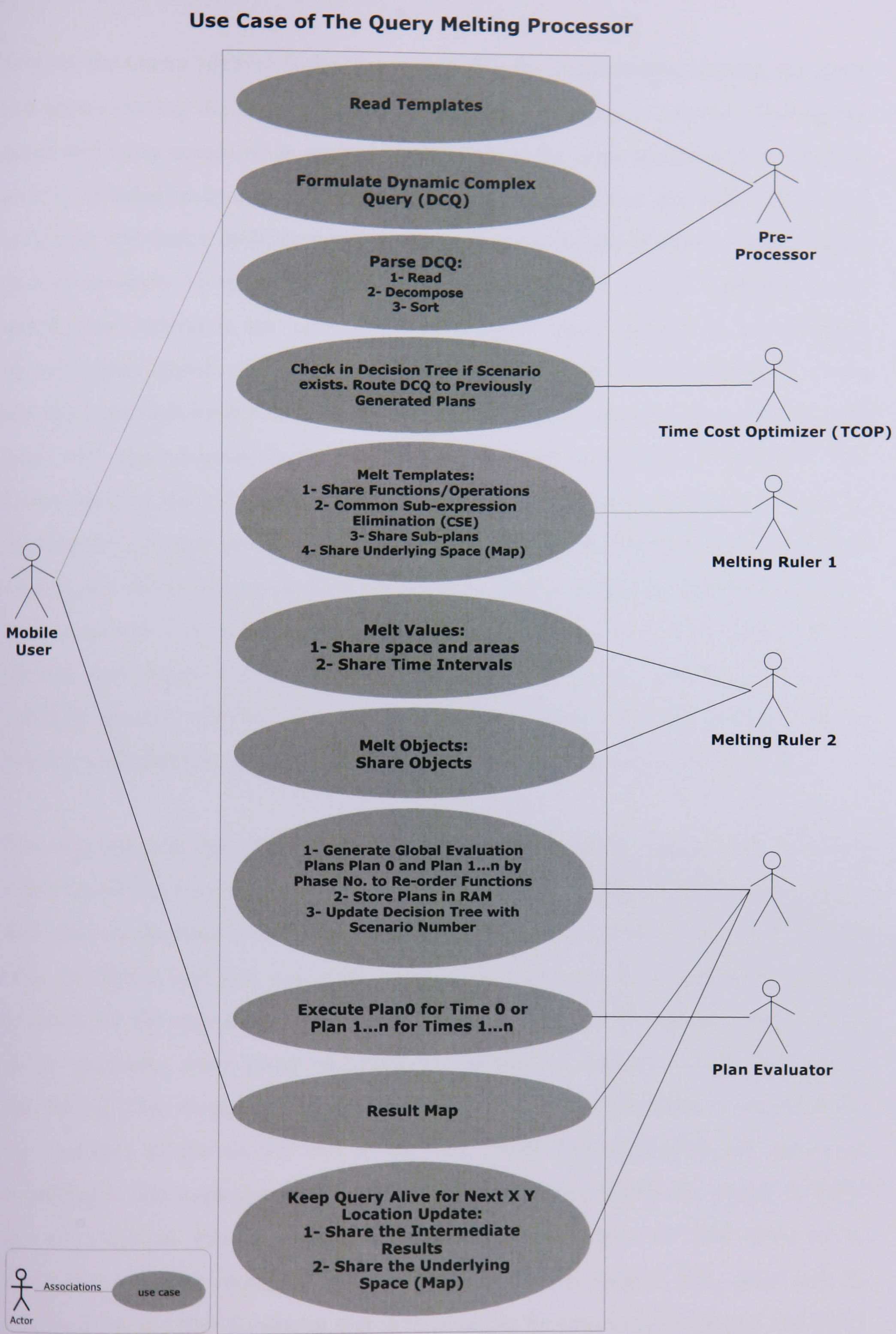


Figure 6.1: The Use Case of the Query Melting Processor.

Second, the Query Melting Ruler 2 is responsible for implementing sharing the space and areas, sharing the time intervals, and sharing the object of interest. Sharing the space and areas occurs when multiple queries share the same spatial area or when an area is included in another. The Query Melting Ruler 2 draws the buffer of an area only once and uses it multiple times to clip or find the objects of other multiple simple queries similarly to the work done in [And02a, And02b] but for multiple simple queries that belong to the same dynamic complex query instead of non dynamic multiple users queries using one operator only. Sharing the interval of time is a new sharing paradigm hereby introduced because it was not considered in the previous work that was reviewed in the literature of the query optimization strategies. The Query Melting Ruler 2 allows sharing an interval of time if it is included or equal to the interval of other multiple queries. Finally, sharing the object of interest allows sharing the object of interest between multiple simple queries by reading the object table once and using it for many queries. In the work done by [And03] the object of interest was shared between multiple users each using one operator only in non dynamic queries, whereas it is applied here to share it between multiple simple queries with multiple operators that belong to the current dynamic complex query.

The next task is responsible for Generating the Plans, both the Global Execution Plan 0 and the Global Execution Plan 1…n. During this task, the push-down strategy that is discussed in the review of literature in [ELM06] is employed. It was used to swap only the Select and Join operators, whereas in this work it is applied to re-order properly all the operations after Melting them in the aim to organize them in the proper sequence. Each group of operations belongs to a phase such as first phase, second, etc. This means that the operators of the first phase should be executed before the operators of the second and so on. The global execution plans are generated according to the sequence of these phases. The generated plans are stored in RAM and the Decision Tree is updated with the scenario number for later retrieval by successive dynamic complex queries with similar scenarios. The next task is performed by the Plan Evaluator that is responsible for executing the global execution plan functions sequentially by following the order in which they appear. In the next task, the query melting processor sends the map that contains the results of the query to the mobile user.

Finally, in the last task, the query melting processor keeps the query of each user alive as a thread and waits for the next mobile user X Y location update. Here the processor is actually implementing sharing the intermediate results that was presented in the review of literature. However, the implementation in [And06] was limited to multiple-users non dynamic one operator queries whereas it is implemented in this work for multiple-users dynamic complex queries. The processor also employs sharing the underlying space (map) by keeping the map in the thread to be re-used by new time instances of the mobile user. The main difference is that it was used in [Mok05a, Mok05b] for multiple-users dynamic one operator queries whereas it is implemented in this work for multiple users dynamic multiple operators multiple simple queries per operator queries.

6.2.2 The Timeline Sequence Diagram of a Dynamic Complex Query

The Time Sequence Diagram shows the order in time of each task or activity including the user. Each task is shown in a box in the first row. The diagram visualizes how long each task lasts by specifying the beginning and the end of its time interval using vertical bars. Moreover, it indicates the input and output of each task. The right arrows could represent the output of a task or launching the next task. The left arrows show when the result is returned to the user.

The diagram is read for each user time interval from left to right following the right arrows and down following the blue bars. The left arrows indicate that the result map is sent back to the user. A user might have many labelled bars, *Time 0*, *Time 1*, and *Time n*, each of which specifying the time interval when the user is active. A user is considered active when he launches a dynamic complex query or his X Y location is updated.

Figure 6.2 shows the Time Sequence Diagram of a Dynamic Complex Query. At *Time 0*, the user launches a dynamic complex query which is received by the Query Watcher and sent to the Query Parser. The Query Parser decomposes the dynamic complex query into multiple simple queries. The Query melting task reads the simple queries and fetches their corresponding Operators Templates. It melts the simple

queries and their functions. Then it generates the Global Evaluation *Plan 0* that is to be executed only once for *Time 0* of the user, and the Global Evaluation *Plan 1...n*, that is to be executed at every new time instance that reports a change in the user X Y location. It submits the current X Y location of the user at *Time 0* to the Global Evaluation *Plan 0* and evaluates it. The resulting map is returned to the user at the end of the GEP0 execution.

The activities of the processor are executed only once when the user launches the dynamic complex query at *Time 0*. Afterwards, for each new Time Instance or when a change in the user's X Y location is reported to the server and hence to the processor, only the GEP 1...n is evaluated. At *Time 1*, when the X Y Watcher receives a new user X Y location, it submits the new location coordinates to the Global Evaluation *Plan 1...n*, evaluates it, and returns a new map with the new results to the mobile user. The same is repeated for new time instances until the user gets disconnected, stops the query or the life time of the query expires.

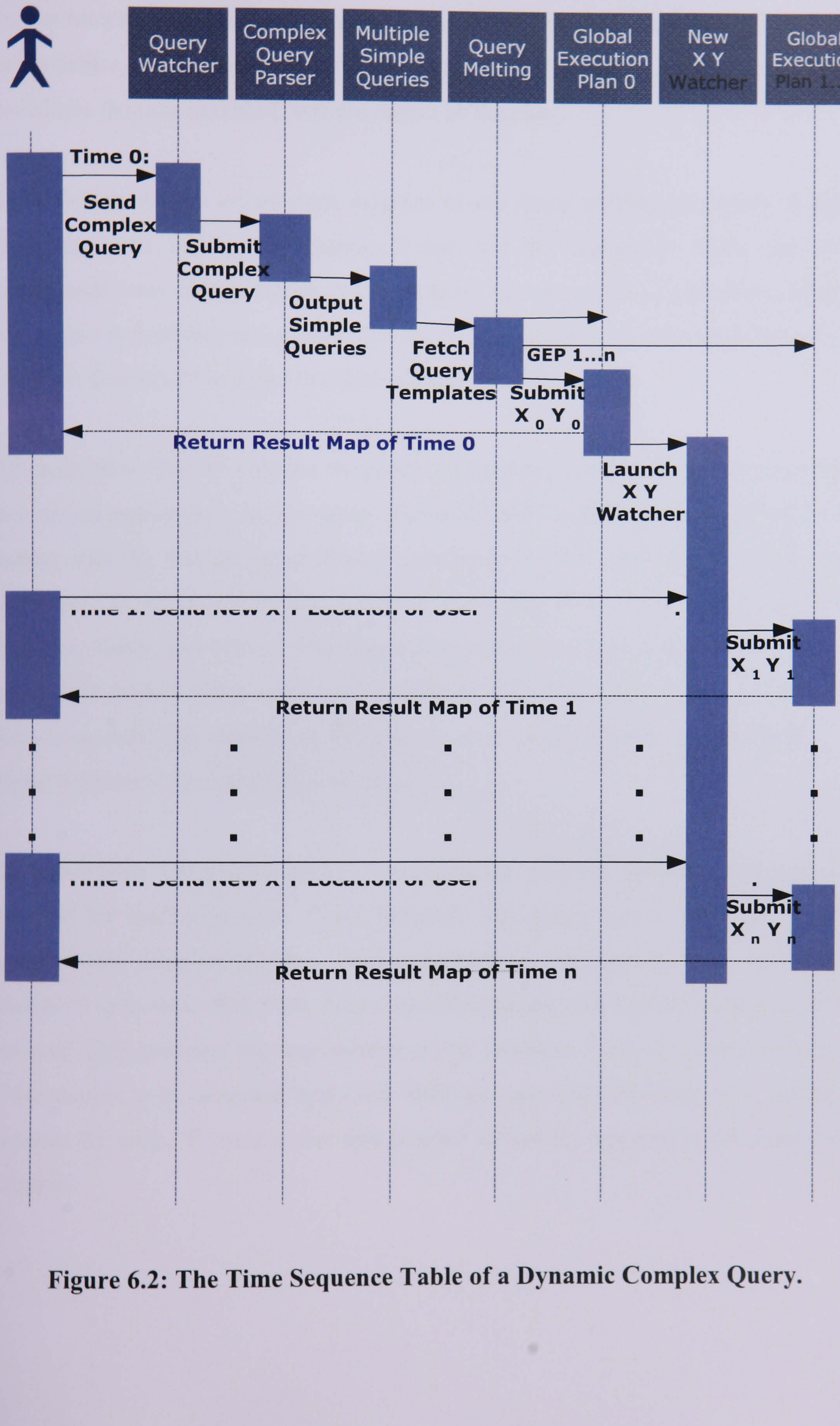


Figure 6.2: The Time Sequence Table of a Dynamic Complex Query.

6.2.3 The Architecture of the Query Melting Processor.

The Architecture Diagram of a processor shows the input tables with their attributes, the variables, fields, or arrays that are stored in the memory, the processes or procedures that are executed, and the output of the class.

Figure 6.3 shows the architecture diagram of the query melting processor. It shows the input tables namely the Queries Table and the Templates Table, the multi-dimensional array that is stored in the memory, the processes or procedures that are responsible for melting the queries, and the two generated execution plans namely the Global Evaluation *Plan 0* and the Global Evaluation *Plan 1…n*.

The input table Queries supplies the processor with the multiple simple queries where each record represents a simple query. The attribute *ID* represents the user mobile unit number and the rest of the attributes correspond to the components of the Query Language. As discussed earlier, a simple query has three components namely the *Operator*, *Value*, and *Object*. The Operator specifies the type of query such as to find the nearest facility (SN) and to find facilities within a certain buffer (SWB). The *Value* represents the number of facilities to find or the distance of the buffer. The *Object* represents which facilities to find.

The input table Templates supplies the processor with the functions that are to be executed for each *Operator*. Each Template has many tuples. The ID1 attribute represents the template number, *Operator* represents the operator that the template belongs to or is associated with, *Phase* specifies during which phase a method is to be executed, *Step* contains the step number of the function, *Function* contains the name of the method to be executed, and *Code* attribute represents the status of a method for example the code "#" means that this method should be repeated at every new time instance.

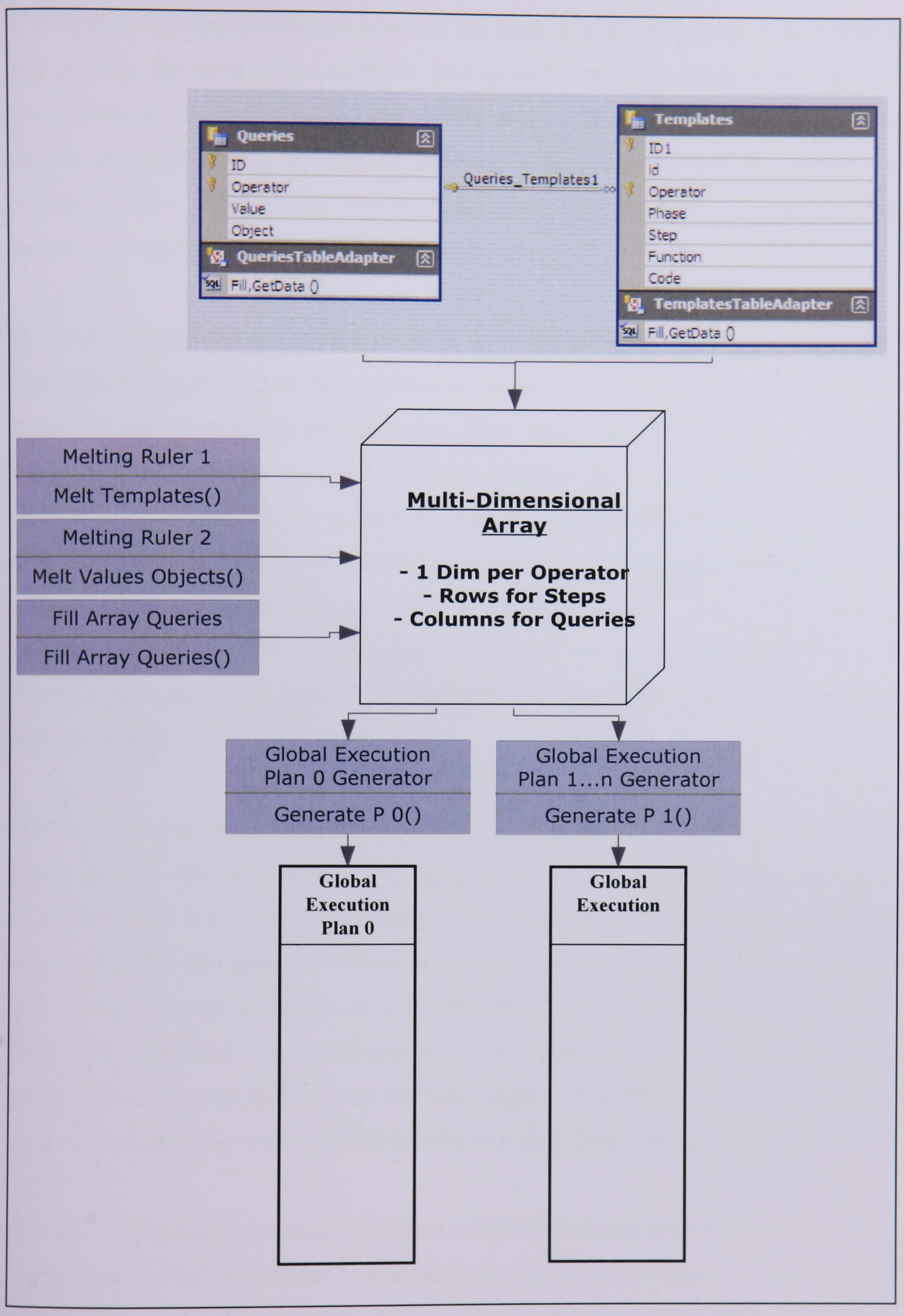


Figure 6.3: The Architecture of the Query Melting Processor.

The Multi-Dimensional Array that is located in the memory is made up of three dimensions. The first dimension which is the rows corresponds to the Step numbers and contains the name of the methods. The second dimension which is the columns corresponds to the templates and the simple queries. The templates are assigned in column number 0 and the queries are assigned in the rest of the columns starting by column number 1 and so on. The third dimension represents the Operators. Each operator has one plan or dimension of its own.

The Melting Ruler 1 *MeltTemplates()* process is responsible for melting the templates. The methods that belong to the templates are scanned through from the first template to the last one considering each step at a time. When the same method is found in many consecutive templates, it is enough to execute it once only for the first one. In such a case the method is kept in the first template and deleted from all the subsequent ones. This procedure eliminates common methods between templates which results in melted templates that are repetition free. The Melting Ruler 2 *MeltValuesObjects()* process is responsible for melting whole queries if they are totally included in each others, static queries in their corresponding dynamic ones, melting values, and melting objects.

Simple queries are considered as totally included in other queries when they have the same operator, the same object, but different values. For example, the first query could be to find the nearest 3 restaurants and the second query to find the nearest 5 restaurants. The first query is totally included in the second one, hence, should be fully melted. Another example would be when the first query is to find the restaurants within 500 meters and the second one to find the restaurants within 300 meters. Both queries have the same operator and the same object, but different values. The second query is considered as totally included in the first one, hence, should be fully melted.

A static query produces a result to the user related to his current position at the current time such as to find the nearest 3 restaurants. The query is executed only once at *Time 0*. On the other hand, a dynamic query is launched for a certain period of time. It supplies the user with the result related to his current position and time and keeps on supplying him with updated results based on his new location at every new time instance. If both the static query and the dynamic query have the same operator and

the same object, the static query is considered as a particular case of the dynamic one. Hence, the static one should be fully melted because it is totally included in the dynamic one. For example, the static query could be to find the nearest 4 hotels now and the dynamic one to find the nearest 4 hotels for the next hour supplying the result every 2 minutes. Finding the 4 hotels now is in fact the same as the first answer of the dynamic query at *Time 0* related to the current location of the user.

Melting values occurs when many consecutive simple queries have the same value. For example, three consecutive simple queries could be respectively to find the nearest 3 hotels, to find the nearest 3 restaurants, and to find the nearest 3 universities. The value 3 could be set once only for the first query and reused by the subsequent ones. In such a case, the values of the second and the third simple queries are melted. Thus, the repetitions of the same value are removed. Melting objects occurs when many consecutive simple queries have the same object even when they have different operators. For example, two consecutive simple queries could be respectively to find the nearest 3 hotels with their corresponding shortest paths and to launch a dynamic query for one hour to find the time left to reach the 5 nearest hotels. The object hotels could be set once only for the first query and reused by the second query. In such a case, the object of the second query is melted. Thus, the repetition of the same object is removed.

The Fill Array Queries process, *FillArrayQueries*(), is responsible for building the plan by filling the steps of each query with the methods that remain not melted. This procedure takes into consideration the code of each method that was input from the Templates table. Each code represents the type of the method in terms of execution rule. Some methods should be executed only once for all the queries, others at every new time instance, etc. The codes are shown in Table 6.1.

Code	Execution Rule
*	Should be repeated for every simple query
%	Should be filled by the *Value* of the simple query
?	Should be filled by the *Object* of the simple query
No code	Should be changed for every new operator
/	Should be repeated only once for all the queries
#	Should be repeated at every new Time Instance

Table 6.1: The Code of the Execution Rule of the Methods.

The code "*" means that this particular method is supposed to be executed for every simple query. Thus, for every simple query, the element in the array that corresponds with the step number is assigned and filled with the method name. The code "%" means that the name of this particular method is supposed to be appended with the *Value* of the query and then assigned to the array element that corresponds to the step number of the method. The code "?" means that the name of this particular method is supposed to be appended with the Object of the query and then assigned to the array element that corresponds to the step number of the method. The empty code "" means that this particular function is supposed to be executed for every new operator. Thus, the first query of every operator is assigned and filled with the method name. The code "/" means that this particular method is supposed to be executed only once for all the queries. Thus, only the first query of the first operator template is assigned and filled with the method name. The code "#" means that this particular method is supposed to be executed for every new time instance. Thus, the first query of the first operator of both plans *Plan 0* and *Plan 1...n* are assigned and filled with the method name.

While filling the queries array, the process *FillArrayQueries*() is actually filling only the non melted methods, non melted values, non melted objects, and non melted queries. After filling them, the multi dimensional array ends up having for each simple query a list of the methods that are necessary to be executed in order to produce the output of the query.

The Generate Plans process, *Generateplans(),* is responsible for generating the Global Evaluation Plans for the complex query. It generates the Global Evaluation *Plan* 0 only when all the simple queries are of the static type. However, it generates two plans namely the Global Evaluation *Plan 0* and the Global Evaluation *Plan 1...n*, when the complex query is formed of either dynamic simple queries or a combination of static and dynamic queries.

The attribute *Phase* in the operator template table specifies for each method the phase number during which it is supposed to be executed. For each phase, the columns of each query methods are appended one after the other at the end of a new list. The same is repeated for *Time 0* and *Time 1...n*. The resulting two lists of methods are the Global Execution *Plan 0* and the Global Evaluation *Plan 1...n* each of which is made up of a list of functions or methods that are to be executed.

6.3 Implementation of the QMP

The QMP is implemented using the Microsoft Visual Basic .NET running under the Microsoft Windows XP. The VB.NET programming language is a component of the Microsoft Visual Studio 2005 which is an object-oriented programming environment. The underlying Geographic Information System is the ESRI ArcMap and ArcView version 9.2 which are products of the ESRI ArcGIS Software that offers software and technology for desktop mapping, online maps, GIS data, and products [ESR]. The ArcMap allows users to access its objects, controls, and toolboxes. The underlying experimental environment consists of a 1.8 GHz Intel Centrino Duo PC with 1 GB RAM main memory running Microsoft Windows XP Home Edition Version 2002 Service Pack 2.

6.3.1 User Interface

The User Interface of QMP is divided into sections as shown in Figures 6.4, 6.5, and 6.6. Each Operator such as "Time Left" and "Distance Left" has its own section that is made up of two boxes, the first for the dynamic queries and the second for static queries. Each box contains six columns, the first for the template and the next for queries. Each row corresponds to the method/function. The two long list boxes to the right are the global evaluation plans. A START/STOP is used to start and stop the processor. After processing a complex query, the execution time is displayed in the box called Cost of QMP that is located under the Start/Stop Button. The execution time is expressed in nanoseconds and seconds. During execution, the queries are displayed in their boxes after each procedure.

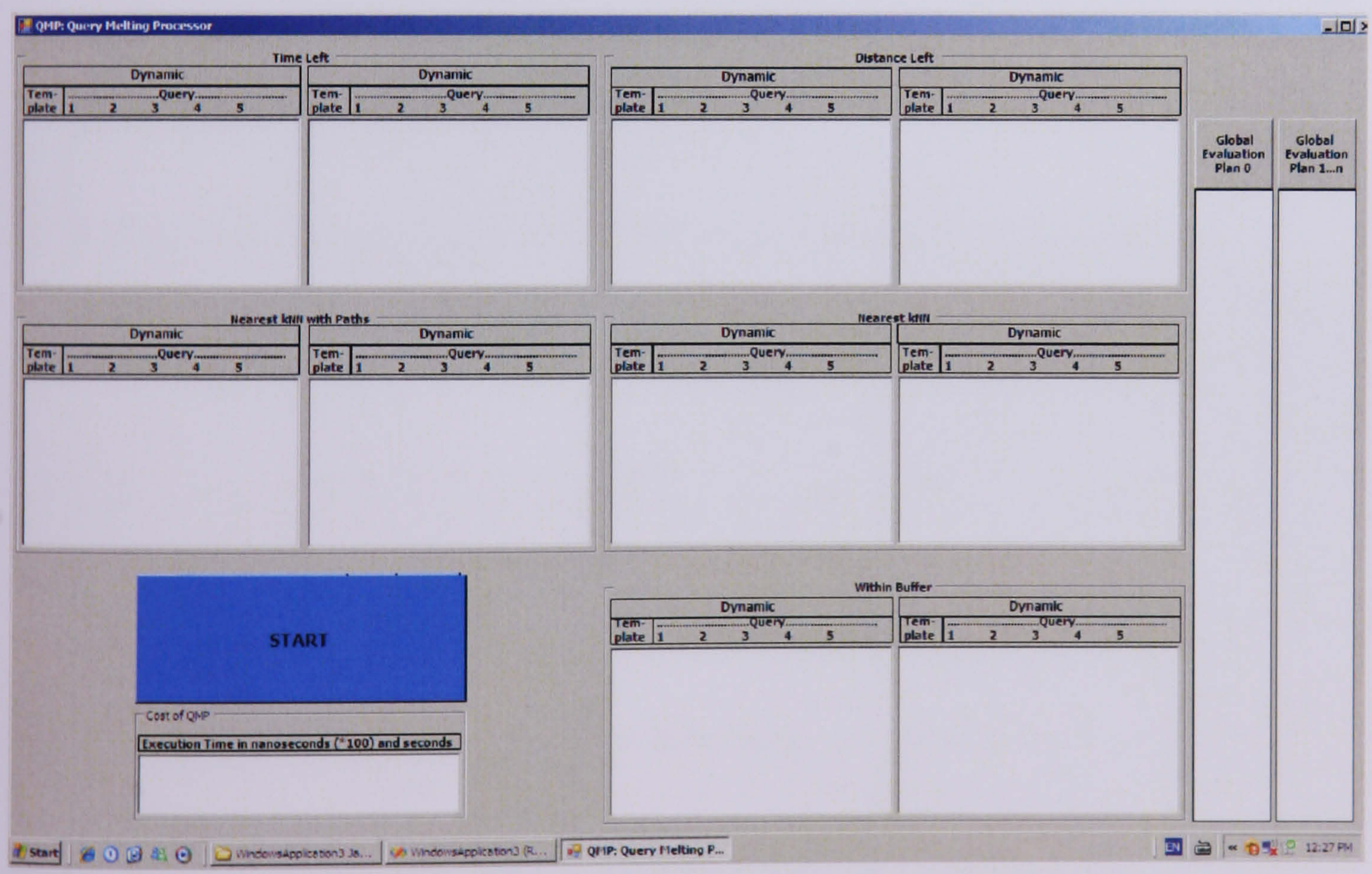


Figure 6.4: The User Interface of the Query Melting Processor.

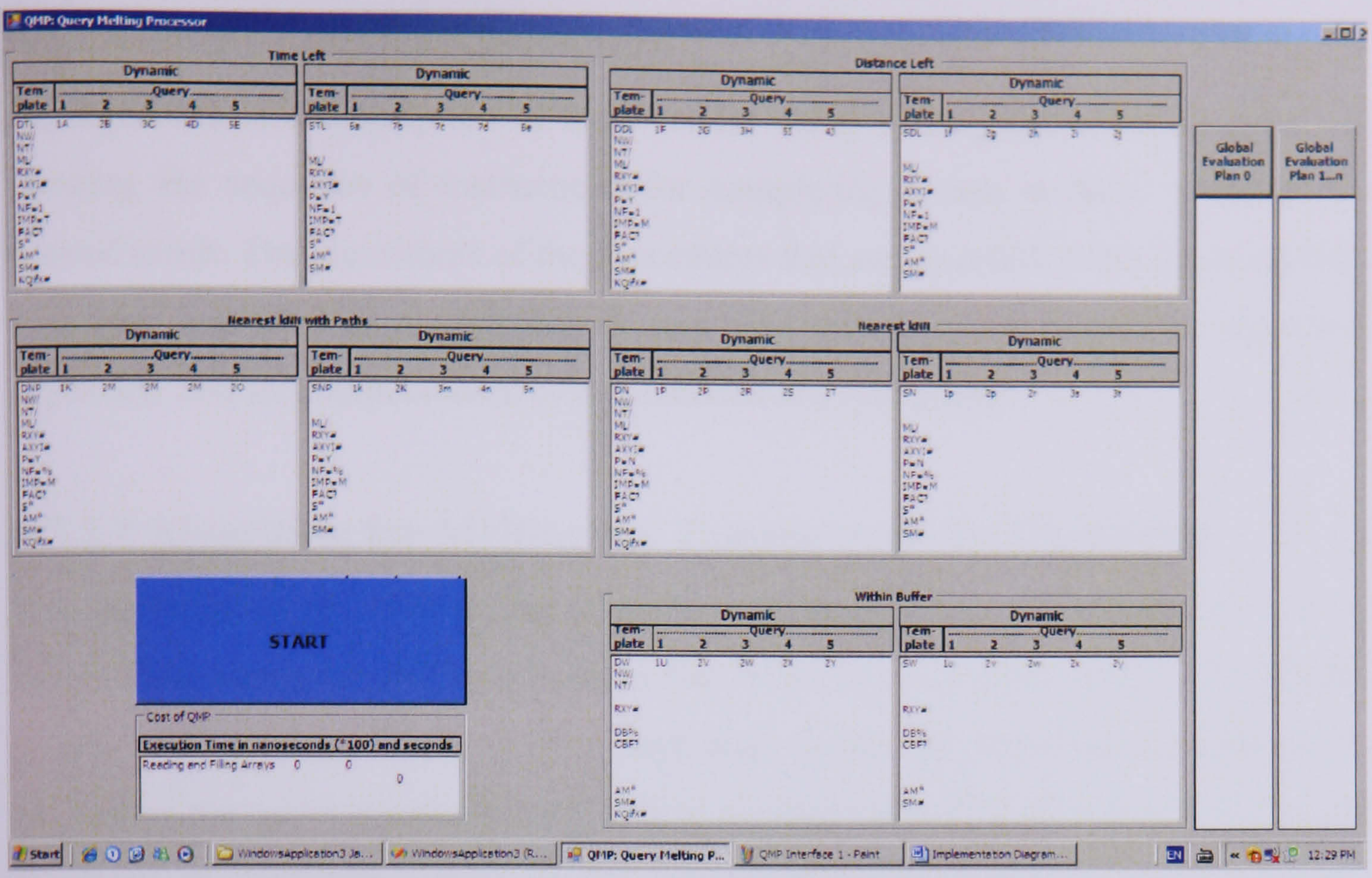


Figure 6.5: The User Interface of the Query Melting Processor before Melting.

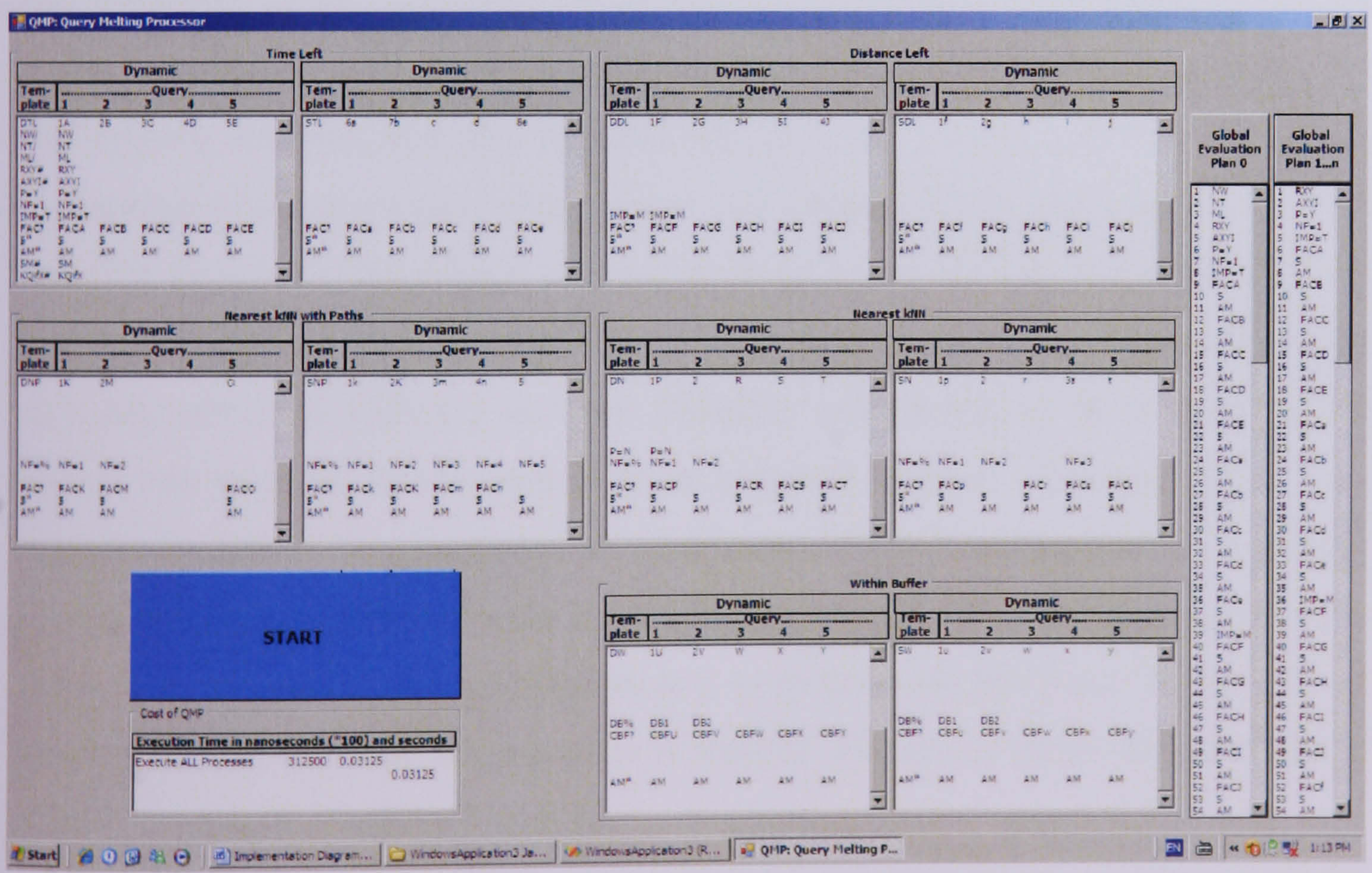


Figure 6.6: The User Interface of the Query Melting Processor after Melting.

6.3.2 Melting Queries Algorithms

An algorithm is a description of a procedure using a computable set of steps and showing the sequence of instructions for completing a task in order to achieve a desired result. The algorithms of the procedures that are included in the query melting processor are detailed below where each procedure is presented, its algorithm explained, and its computational cost calculated and quantified.

6.3.2.1 Algorithm for Melting the Templates of the Operators

The algorithm of the Melting the Templates of the Operators is presented to show how it implements the sharing paradigm that is investigated in the review of literature of query optimization strategies. The work done in [And06, Mok05a] is implemented by sharing the cache memory whereas here it is applied differently and is used at the low-level of processing, e.g., at the programming or coding level. It performs common sub-expression elimination CSE, that is discussed in [Kan94, Mok03], by eliminating all the repetitions of the same process in different plans.

The templates of the operators are melted by eliminating the functions that are repeated more than once in consecutive templates. The process tackles each step at a time starting from the first step until the last one. The functions of each row which corresponds to a step are tackled in reverse order starting by the last template until the first. The function of the current template is compared to the function of the previous one, and if they are equal the function of the current template is eliminated then the procedure proceeds with the previous template considering its function to be the current one in order to compare it to its previous function. This process is done recursively until all the repetitions are eliminated. If the functions that are compared are not equal the procedure leaves the second function as it is, considers the previous function as the current one, and compares it to its previous functions. This process is done recursively until the first template is reached. Figure 6.7 shows the algorithm of melting the templates of the operators.

```
//Algorithm of MeltTemplates()
Maximum = maximum number of steps in templates
For (each step s in first operator template steps running s from 1 to Maximum)
    For (each template t in templates processed in reverse order from last template to second template)
        For (each previous template tp in templates processed in reverse order from template -1 to first template)
            If (function in step s of template t equals function in step s of template tp)
                {   Remove function in step s of template t
                    Proceed with next template t                }
```

Figure 6.7: The Algorithm for Melting the Templates of the Operators.

The Computational complexity of an algorithm is used to measure how the cost of computation grows as a function of the size of the input. Theoretical analysis considers the computation cost to be the running time in terms of the input size. It allows us to evaluate the speed of an algorithm independently of hardware and software environment. The analysis of an algorithm determines the running time in Big-Oh notation. The Big-Oh notation indicates an approximation to the number of steps taken by the algorithm. It gives or indicates an upper bound on the growth rate of a function, and it is also used to rank multiple functions according to their growth rate.

For example, an algorithm that scans through all the elements of a matrix of size Columns × Rows, has the computational complexity *O(Columns × Rows)*. This means that the number of steps taken to traverse a matrix of size Columns × Rows grows proportionately to the product of the number of columns and the number of rows. In other words, traversing a matrix of size 4×5 takes about twice as much as a matrix of size 2×5.

The computational cost of the above algorithm is *O(Ns × (Nt-1))*, where *Ns* is the number of steps and *Nt* is the number of templates. The steps correspond to the row index numbers of each template. The templates occupy the first column of each plan that is columns with index 0. All the steps of the first template are processed. The

computational cost of processing *Ns* steps is *O(Ns)*. For each step, the rest *Nt-1* templates that are numbered from two to the last one are processed in reverse order. The computational cost of processing *Nt-1* templates is *O(Nt-1)*. Thus, the computational cost of both these dimensions is *O(Ns × (Nt-1))*.

6.3.2.2 Algorithm for Melting Full Queries

The queries that are totally included in other queries are fully melted by eliminating their values and their objects as well. For each template, the queries that are associated with it are by turn compared to their next queries. If the queries have the same object, their values are compared. The query that has a value less than or equal to the value of the next one is completely eliminated. This process is repeated until all the templates are scanned. Figure 6.8 shows the algorithm of fully melting the included queries.

```
// Algorithm to Melt Full Queries

For (each template t in templates)
        For (each query q that is associated with template t of templates)
                Current = current query
                For (each next query q processed from query + 1 to last query)
                        If (object of current query equals object of next query)
                                {
                                        If (value of current query is less than or
                                        equals to value of next query)
                                                {
                                                // fully melt current query
                                                Remove value of current query
                                                Remove object of current query
                                                }
                                }
```

Figure 6.8: The Algorithm for Melting Full Queries (Values and Objects).

The computational cost of the above algorithm is $O(Nt \times Nqti \times \sum_{n=1}^{qk-1} n)$, where Nt is the number of templates, $Nqti$ is the number of queries of template i, and qk is the query number (the k^{th} query). The queries correspond to the columns index numbers that are associated with each of the templates. They occupy the columns numbered from one to q of each plan that represents a template. The sum of the number of queries is in fact equal to the total number of queries. Each query q that is associated with template t is compared to all its subsequent queries that are also associated with the same template t. Hence, the computation cost of the inner loop in the algorithm is *(qk-1)+...+3+2+1*.

6.3.2.3 Algorithm for Melting the Values of the Queries

The algorithm of the Melting the Values of the Queries is presented to show how it implements the sharing paradigm that is investigated in the review of literature of query optimization strategies. The work done in [And06, Mok05a] is implemented by sharing the space and areas. In this work, when two queries share the same spatial area or when an area is included in another one, Melting their Values uses the shared area for multiple queries. This is done by drawing the buffer of an area only once and using it multiple times to clip or find the objects of multiple simple queries similarly to the work done in [And02a, And02b] but for multiple simple queries that belong to the same dynamic complex query instead of multiple users queries using one operator only. Moreover, the algorithm allows sharing an interval of time if it is included or equal to the interval of another query. This is a newly introduced sharing that is not addressed by any previously reviewed paper.

The values of the queries are melted if they are the same for consecutive queries. In this case, the first occurrence of the value is kept unchanged and the next occurrences are melted by being eliminated. This process tackles all the values of all the queries of all the templates. However, the queries are processed independently from the template number. The queries are scanned in order from the first query to the last one. The procedure considers the first value of the first query as the previous value. It starts comparing the values from the second and so on considering the second value as the current value. If the value of the current query is equal to that of the previous one, it is

eliminated and the next value of the next query is considered as the current value. If the value of the current query is not equal to that of the previous one, the previous value is kept unchanged and the current value is considered as the new previous value. The process continues until all the values of all the queries are examined. Figure 6.9 shows the algorithm of melting the values of the queries.

```
//Algorithm to Melt Values

Previous = value of first query q of queries
For (each query q of queries processed from second query to last query)
        If (value in query q of queries equals previous)
                {
                        Remove value of query q of queries
                }
        Else
                {
                        Previous = value of query q
                }
```

Figure 6.9: The Algorithm for Melting the Values of the Queries.

The computational cost of the above algorithm is *O(Nq-1),* where *Nq* is the total number of queries. All the queries are processed starting from the first one until the last one.

6.3.2.4 Algorithm for Melting the Objects of the Queries

The algorithm of the Melting the Templates of the Operator is presented to show how it implements the sharing paradigm that is investigated in the review of literature of query optimization strategies. The work done in [And06, Mok05a] is implemented by sharing the object of interest. The algorithm allows sharing the object of interest between multiple simple queries by reading the object once and using it for many queries. In the work done by [And03] the object of interest is shared by multiple users each using one operator only in non dynamic queries, whereas it is applied here

to share it between multiple simple queries with multiple operators that belong to the current dynamic complex query.

The objects of the queries are melted if they are the same for consecutive queries. In this case, the first occurrence of the object is kept unchanged and the next occurrences are melted by being eliminated. This process tackles all the objects of all the queries of all the templates. However, the queries are processed independently from the template number. The queries are scanned in order from the first query to the last one. The procedure considers the first object of the first query as the previous object. It starts comparing the objects from the second and so on considering the second object as the current object. If the object of the current query is equal to that of the previous one, it is eliminated and the next object of the next query is considered as the current object. If the object of the current query is not equal to that of the previous one, the previous object is kept unchanged and the current object is considered as the new previous object. The process continues until all the objects of all the queries are examined. Figure 6.10 shows the algorithm of melting the objects of the queries.

```
//Algorithm to Melt Objects

Previous = object of first query q of queries
For (each query q of queries processed from second query to last query)
        If (object in query q of queries equals previous)
                {
                        Remove object of query q of queries
                }
        Else
                {
                        Previous = object of query q
                }
```

Figure 6.10: The Algorithm for Melting the Objects of the Queries.

The computational cost of the above algorithm is *O(Nq-1)*, where *Nq* is the total number of queries. All the queries are processed starting from the first one until the last one.

6.3.2.5 Algorithm for Filling the Methods of the Remaining Queries

The functions that remain not melted are assigned in the cells of their corresponding remaining queries. The procedure tackles each step at a time from the first step until the last one. For each step, the code associated with the functions determines whether the function is supposed to be repeated for each query, for each new time instance, for each new operator, or once only for all the queries. It also specifies whether the function's name is supposed to be appended with the value or the object of the query. The procedure checks the code of each template step and accordingly fills the cell that corresponds to the query cell in the array. When the code specifies that a function is supposed to be repeated for all the queries, the procedure fills all the queries cells with the name of the function. When it specifies that a function is supposed to be repeated for every new time instance, the procedure fills the method name in two cells only. The first cell is the one that corresponds to the cell of the first query of the template. This cell is related to the *plan 0*. The second cell is the element numbered 0 in the array and is related to the *plan 1...n*.

When the code specifies that a function is supposed to be repeated only once for all the queries, the cell of the first query of the first template only is filled with method name. If it specifies that a function is supposed to be repeated for each new template, the procedure fills the cells of the first queries of each template with its corresponding method name. In the case where the method name is supposed to be appended with the value of the query, the process takes the value from the query and appends it to method name before filling it in the corresponding cell. Finally, if the code specifies that the method name is supposed to be appended with the object of the query, the process takes the object from the query and appends it to method name before filling it in the corresponding cell.

This process is repeated until all the remaining functions of all the steps of all the templates are properly filled in the cells of all their corresponding queries. Figure 6.11 shows the algorithm of filling the functions in their corresponding query cells.

```
//Algorithm of FillArrayQueries()

Maximum = maximum number of steps in templates
For (each step s in first operator template steps running s from 1 to Maximum)
        For (each query q in queries)
                If (code of template t of query q equals "*") // repeat the same function
                for all queries of all templates
                {
                        Function of query q of Plan0 = function of template t
                        Function of query q of Plan1toN = function of
                        template t
                }
                If (code of template t of query q equals "%") // append the function
                name with the value of the query q
                {
                        Function of query q of Plan0 = function of template t
                        + value of query q
                        Function of query q of Plan1toN = function of
                        template t + value of query q
                }
                If (code of template t of query q equals "?") // append the function
                name with the object of the query q
                {
                        Function of query q of Plan0 = function of template t
                        + object of query q
                        Function of query q of Plan1toN = function of
                        template t + object of query q
                }
```

If (**code** of template **t** of query **q** equals “”) and (first **query** of template **t**) // for first query of template t for Plan0 and Plan 1toN

// to change for every new operator

{

Function of query **q** of **Plan0** = **function** of template **t**

Function of query **q** of **Plan1toN** = **function** of template **t**

}

If (**code** of template **t** of query **q** equals “/”) and (first **query** of first **template**) // for Plan0 of first query of first operator only

{

Function of query q of **Plan0** = **function** of template **t**

}

If (**code** of template **t** of query **q** equals “#”) and (first **query** of first **template**)// for Plan0 & Plan1toN first query of first operator

// to repeat for every new time instance

{

Function of query **q** of **Plan0** = **function** of template **t**

Function of query **q** of **Plan1toN** = **function** of template **t**

}

Figure 6.11: The Algorithm for Filling the Functions that Remain Non-Melted in the Queries.

The computational cost of the above algorithm is $O(Ns \times Nq)$, where Ns is the number of steps and Nq is the number of queries. The steps correspond to the row index numbers of each template. The queries correspond to the columns of each plan. All the queries are processed for all the steps. The computational cost of processing Ns steps is $O(Ns)$ and processing Nq queries is $O(Nq)$. Thus, the computational cost of both these dimensions is $O(Ns \times Nq)$.

6.3.2.6 Algorithm for Generating the Global Evaluation Plans

The algorithm of Generating the Plans is used to show both Global Execution Plan 0 and Global Execution Plan 1…n are generated and show how it implements the push-down strategy that is discussed in the review of literature in [ELM06]. The push-down strategy was used to swap only the Select and Join operators, whereas in this work it is applied to re-order properly all the operations after Melting them in the aim to organize them in the proper sequence. Each group of operations belongs to a phase such as first phase, second, etc. This means that the operators of the first phase should be executed before the operators of the second and so on. The algorithm generates the plans according to the sequence of these phases.

The procedure that generates the two global evaluation plans collects all the remaining non empty methods/functions that are in the array. Then, it generates a new single dimensional array for each global evaluation plan made up of the list of functions that are to be executed. Each function is supposed to be executed during a particular phase. The methods that belong to each phase are collected from the array according to the increasing order of their step number that is related to that specific phase. In other words, the methods included in the columns of the queries are scanned sequentially by phase and by step. Then they are appended at the end of their corresponding Global Evaluation Plan. Figure 6.12 shows the algorithm of generating the Global Evaluation *Plan 0* and the Global Evaluation *Plan 1…n*.

```
//Algorithm of Generateplans()

Plan0 = empty list for Plan 0
Plan1toN = empty list for Plan1toN
For (each phase p of phases)
        For (each template t of templates)
                For (each query q in queries associated with template t)
                        For (each step s in steps of phase p)
                                // Plan 0
                                Add function of step s of query q to the end of Plan0
                                // Plan 1…n
                                If (first template and first query)
                                        {
                                                                        Add function of step s of query q0 to the
                                                                        end of Plan1toN
                                        }
                                Else
                                        {
                                                                        Add function of step s of query q to the
                                                                        end of Plan1toN
                                        }
```

Figure 6.12: The Algorithm for Generating the Global Evaluation *Plan 0* and the Global Evaluation *Plan 1…n*.

The computational cost of the above algorithm is $O(Nq \times Ns)$. All the multi-dimensional array elements are traversed. For each template all the queries are processed. For each query all the steps are processed. The computational cost of processing all the queries Nq is $O(Nq)$ and Ns steps is $O(Ns)$. Thus, the computational cost of processing all the dimensions is $O(Nq \times Ns)$. The phase number does not affect the running time of the algorithm because it is used only to control the sequence in which the global evaluation plans are generated.

6.4 Theoretical Evaluation of the Query Melting Processor

In order to evaluate the efficiency and cost effectiveness of the Query Melting Processor, the major factors that are taken into consideration are the execution time cost, memory space used to store the necessary arrays, and the extra I/O operations of the templates along with their respective execution time costs. The Execution Time of multiple static queries with multiple operators before query melting is defined as follows:

$$ETC(N,S) = \sum_{op \in OP_{sent}} \left(\sum_{s \in S_{op}} t_s(op) \times \left| Q_{op} \right| \right)$$

where

- *ETC function is the Execution Time Cost of the strategy N (No Melting) on the type S which is Static queries.*
- OP_{sent} *is the set of operators that are included in the queries.*
- Q_{op} *is the set of queries with a particular operator.*
- S_{op} *is the set of steps or functions of a particular operator.*
- $t_s(op)$ *is the execution time of a step or function of a particular operator.*

The total cost is equal to the summation of all the costs of the operators that are sent where the cost of each operator is the summation of all its functions costs multiplied by the number of queries of that particular operator. The Execution Time of multiple dynamic queries with multiple operators before query melting is defined as follows:

$$ETC(N,D) = \sum_{op \in OP_{sent}} \left(\sum_{s \in S_{plan0}} t_s(op) \times \left| Q_{op} \right| \right) + \left| t_i \right| \times \sum_{op \in OP_{sent}} \left(\sum_{s \in S_{plan1 \ldots n}} t_s(op) \times \left| Q_{op} \right| \right)$$

where

- *ETC function is the Execution Time Cost of the strategy N (No Melting) on the type D which is Dynamic queries.*
- S_{plan0} *is the set of steps or functions of a particular operator at Time 0.*
- $S_{plan1 \ldots n}$ *is the set of steps or functions of a particular operator at Times 1…n.*
- t_i *is a time instance.*

Since this is a continuous query, two plans are considered namely *Plan 0* for *Time 0* and *Plan 1...n* for *Times 1...n*. The total cost is equal to the sum of *Plan 0*'s cost and the *Plan 1...n* cost multiplied by the number of instances. The number of instances represents the number of XY Location updates of the user. The cost of each plan is calculated the same way as for static queries. The Execution Time of multiple static queries with multiple operators after query melting is defined as follows:

$$ETC(M,S) = \sum_{s \in GEP} t_s$$

where

- *ETC function is the Execution Time Cost of the strategy M (Melting) on the type S which is Static queries.*
- *GEP is the set of steps or functions of the Global Execution Plan.*

The total cost is equal to the summation of all the steps functions costs that belong to the Global Evaluation Plan. The Execution Time of multiple dynamic queries with multiple operators after query melting is defined as follows:

$$ETC(M,D) = \sum_{s \in GEP_0} t_s + |t_i| \times \sum_{s \in GEP_{1...n}} t_s$$

where

- *ETC function is the Execution Time Cost of the strategy M (Melting) on the type D which is Dynamic queries.*
- *GEP_0 is the set of steps or functions of the Global Execution Plan for Time 0.*
- *$GEP_{1...n}$ is the set of steps or functions of the Global Execution Plan for Times 1...n.*

The total cost is equal to the sum of GEP_0 cost and $GEP_{1...n}$ cost multiplied by the number of instances. The cost of each GEP plan is calculated the same way as for static queries. The profit of applying the strategy QM is equal to the total execution time saved or decreased. It is measured as the difference between the execution time before melting and the execution time after melting. The profit of Query Melting Static queries is:

$$PROFIT(M,S) = ETC(N,S) - ETC(M,S)$$

The profit of Query Melting Dynamic queries is:

$$PROFIT(M,D) = ETC(N,D) - ETC(M,D)$$

The time to execute the PreProcessor, Query Melting Ruler 1, and Query Melting Ruler2 is considered as a loss and is deducted from the profit of the strategy. The I/O cost is also deducted from the profit. It is the time to read the Functions Cost table once only before Query Melting plus the time to write the updated Functions Cost table once after Query Melting. The net profit of the strategy is calculated as follows:

$$NETPROFIT(M,D) = PROFIT(M,D) - t_{\Pr eP} - t_{Ruler1} - t_{Ruler2} - t_{\mathrm{Re}\,ad} - t_{Write}$$

where

- *NETPROFIT function is the Net Execution Time Cost of the strategy M (Melting) on the type D which is Dynamic queries.*
- t_{PreP} *is the execution time of the PreProcessor.*
- T_{Ruler1} *is the execution time of the Query Melting Ruler 1.*
- t_{Ruler2} *is the execution time of the Query Melting Ruler 2.*
- t_{Read} *is the execution time to read the Functions Cost table before Query Melting.*
- T_{Write} *is the execution time to update the Functions Cost table after Query Melting.*

A positive value of NETPROFIT means that Query Melting is cost effective. A negative value means that the cost of processing Query Melting is greater than the cost saved. Thus, Query Melting is not cost effective. The greater is the value of NETPROFIT, the lower is the total execution time, and the more cost effective is the Melting Process.

For the evaluation of the Query Melting Processor (QMP) and the Decision making Mechanism (TCOP), the 'time' quality characteristic is quantified and cost is estimated using the Big-Oh notation which reflects how the cost of computation

grows as a function the input size by considering the computation cost to be the running time in terms of the size, thus indicating an approximation to the number of steps taken by a process. Therefore, the computational cost of Melting the Templates is $O(Ns \times (Nt\text{-}1))$ where Ns is the number of steps, Nt is the number of templates and the steps correspond to the row index numbers of each template. Since the computational cost of processing Ns steps is $O(Ns)$ where for each step the rest $Nt\text{-}1$ templates that are numbered from two to the last one are processed in reverse order and since the computational cost of processing $Nt\text{-}1$ templates is $O(Nt\text{-}1)$, thus the computational cost of both these dimensions is $O(Ns \times (Nt\text{-}1))$. The same method of calculation applies to the other processes, so the computational cost of Melting Full Queries is $O(Nt \times Nqti \times \sum_{n=1}^{qk-1} n)$, where $Nqti$ is the number of queries of template i, and qk is the query number (the k^{th} query). The queries correspond to the columns index numbers that are associated with each of the templates, occupy the columns numbered from one to q of each plan that represents a template, and each query q that is associated with template t is compared to all its subsequent queries that are also associated with the same template t. Hence, the computation cost of the inner loop in the algorithm is $(qk\text{-}1)+\ldots+3+2+1$.

Moreover, the computational cost of Melting the Values is $O(Nq\text{-}1)$ where all the queries are processed starting from the first one until the last one, Melting the Objects is $O(Nq\text{-}1)$ where all the queries are processed starting from the first one until the last one, Filling and Appending the Methods with Values and Objects is $O(Ns \times Nq)$ where all the queries are processed for all the steps, and Generating the Global Execution Plans is $O(Nq \times Ns)$ where all the multi-dimensional array elements are traversed where for each template all the queries are processed and for each query all the steps are processed. Hence, it can be concluded that the total time cost of the Query Melting Processor is the sum of the time costs of all its processes and can be calculated as follows:

$$O(QMP) = O(Ns \times (Nt\text{-}1)) + O(Nt \times Nqti \times \sum_{n=1}^{qk-1} n) + O(Nq\text{-}1) + O(Nq\text{-}1) + O(Ns \times Nq) + O(Ns \times Nq) + O(Ns \times Nq)$$

The computational cost of the Decision Making Mechanism (TCOP) is *O(Nt)* because the number of levels of the decision tree is in fact equal to the number of templates, which means that *t* comparisons are to be executed in order to determine which scenario is used by the complex query. In the case where a new plan is to be stored while TCOP is activated, the computational cost is equal to *O(Nm)* where *Nm* is the number of functions/methods in the new plan, whereas in the case where an old plan is to be accessed, the computational cost is equal to *O(Om)* where *Om* is the number of functions/methods in the old plan. Hence the total cost of processing a new plan is equal to *O(Nt)* + *O(Nm)* and the total cost of processing an old plan is equal to *O(Nt)* + *O(Om)*. The second quality characteristic that is quantified in order to evaluation the efficiency of the Query Melting Processor is the cost of the memory space that is occupied. Two types of arrays are used during the melting process. The first one is the Cost Array that contains for each function its name and cost. The name might be maximum 20 Bytes and the cost which is an integer number occupies 2 or 4 Bytes depending on the system. The space needed for each function is 24 Bytes which is considered minimal and not significant at all. The second one is the arrays that store the templates of the operators and the functions of the queries. Since the name of each function does not exceed 20 characters and each character occupies 2 Bytes in RAM (some systems 1 Byte) the name occupies maximum 40 Bytes. The total space occupied in the RAM is 40B/Function/Operator. In other words, it is equal to the product of the number of operators, the number of functions of each operator, and 40 Bytes, which is quantified as Space Cost = 40 Bytes × Number of Functions × Number of Operators. Assuming that there are 10 templates of operators, 17 functions per operator, and 10 queries per operator, the space occupied is 40 B × 17 × 10 = 68000 Bytes. Hence, the space cost 68 KB is considered minimal.

On top of the memory space that is occupied by the Query Melting Processor (QMP), the Decision Making Mechanism (TCOP) occupies space for the decision tree and the structure array that is used to store the old plans. First, the tree is made up of *t* levels each corresponding to an operator template and including 2^t elements making a total number of elements $\sum_{i=1}^{t} 2^i$. Since each element stores the operator`s name which is made up of 4 characters and each character occupies 2 Bytes, therefore, the size of the

decision tree can be estimated as $\sum_{i=1}^{t} 2^i \times 8$ Bytes. Second, the structure array is made up of $(2^t\text{-}1)$ rows each corresponding to a plan that contains only the functions/methods that remain non-melted, where each function name is made up of maximum 20 characters and occupies 40 Bytes. Hence the size of the structure array is equal to $((2^t\text{-}1) \times 40$ Bytes $\times$ Number of Functions left $\times$ Number of Operators).

6.5 Summary and Conclusion

The design and implementation of the Query Melting Processor were presented in this chapter. The main job of the processor is to receive dynamic complex queries from the mobile user, decompose them, melt queries either fully or partially, and build an optimized Global Evaluation Plan. The design of the Query Melting Processor is done using a number of visual diagrams each of which visualizing a specific model that is related to a specific activity of query processing. The Use case Diagram is used to visualize the activities and tasks of a system showing at the same time the actor/actors associated with each activity. Each actor could be a person or a component in a system. The Time Sequence Table Diagram shows the order in time of each task or activity including the user. The diagram visualizes how long each task lasts by specifying the beginning and the end of its time interval. It also shows when the result map is returned to the user. The Architecture of the Query Melting Processor Diagram shows the input tables with their attributes, the variables, fields, or arrays that are stored in the memory, the processes or procedures that are executed, and the output of the processor.

The algorithms of each activity have been presented to show how they implement their related purpose. The Melting Ruler 1 algorithm show the Sharing paradigm that was investigated in the review of literature of query optimization strategies [And06, Mok05a] was done by sharing the cache memory whereas here it was applied differently and was used at the low-level of processing, e.g., at the programming or coding level. It performed common sub-expression elimination CSE, that was discussed in [Kan94, Mok03], by eliminating all the repetitions of the same process in different plans. The algorithm of the Query Melting Ruler 2 showed how it allowed sharing the object of interest between multiple simple queries by reading the object

once and using it for many queries. In the work done by [And03] the object of interest was shared between multiple users each using one operator only in non dynamic queries, whereas it was applied here to share it between multiple simple queries with multiple operators that belong to the current dynamic complex query. The algorithm showed also how sharing areas was implemented. When two queries share the same spatial area or when an area is included in another, the QMP uses the shared area for multiple queries. This was done by drawing the buffer of an area only once and using it multiple times to clip or find the objects of multiple simple queries similarly to the work done in [And02a, And02b] but for multiple simple queries that belong to the same dynamic complex query instead of multiple users queries using one operator only.

Moreover, the algorithm showed how it allowed sharing an interval of time if it is included or equal to the interval of another query. This is a newly introduced sharing that was not addressed by any previously reviewed paper. Finally, it showed how sharing the intermediate result was done through using the same underlying space (map) for all the queries of the same user. This was done through executing the plan of each user as a thread. The same was implemented in [And02c, And02d, and And02e] but for multiple users using one operator in non dynamic queries. Moreover, the Query Optimization was here extended to include "multi-user spatio-temporal multi-predicate dynamic complex queries". The algorithm of Generating the Plans is used to show both Global Execution Plan 0 and Global Execution Plan 1...n are generated and show how it implements the push-down strategy that is discussed in the review of literature in [ELM06]. The push-down strategy was used to swap only the Select and Join operators, whereas in this work it is applied to re-order properly all the operations after Melting them in the aim to organize them in the proper sequence. Each group of operations belongs to a phase such as first phase, second, etc. This means that the operators of the first phase should be executed before the operators of the second and so on. The algorithm generates the plans according to the sequence of these phases. On the other hand, the Time Cost Optimization (TCOP) that was responsible for handling the multi-users dynamic complex queries at the GIS server side in order to manage the ones that had similar scenarios, aimed at sharing a previously melted plan instead of generating a new one for each query. The work

done in [ELM06] proposed sharing sub-plans whereas here TCOP which was the new decision making mechanism implemented sharing all the plans.

The QMP is implemented using the Microsoft Visual Basic .NET running under the Microsoft Windows XP. The underlying Geographic Information System is the ESRI ArcMap and ArcView version 9.1 which are products of the ESRI ArcGIS Software. The implementation algorithms of the processes of the Query Melting Processor were presented and the computational cost estimated using the Big-Oh notation. The templates of the operators are melted by eliminating the functions that are repeated more than once in consecutive templates. The computational cost of the algorithm is $O(Ns \times (Nt-1))$, where Ns is the number of steps and Nt is the number of templates. The queries that are totally included in other queries are fully melted by eliminating their values and their objects as well in order to delete them. The computational cost of the algorithm is $O(Nt \times Nqti \times \sum_{n=1}^{qk-1} n)$ where Nt is the number of templates, $Nqti$ is the number of queries of template i, and qk is the query number (the k^{th} query). The sum of the number of queries is in fact equal to the total number of queries. The values of the queries are melted if they are the same for consecutive queries. In this case, the first occurrence of the value is kept unchanged and the next occurrences are melted by being eliminated. The computational cost of the algorithm is $O(Nq-1)$, where Nq is the total number of queries. Similarly, the objects of the queries are melted if they are the same for consecutive queries. The computational cost is also $O(Nq-1)$.

The functions that remain not melted are assigned in the cells of their corresponding remaining queries. For each step, the code associated with the functions determines whether the function is supposed to be repeated for each query, for each new time instance, for each new operator, or once only for all the queries. It also specifies whether the function's name is supposed to be appended with the value or the object of the query. The computational cost of the algorithm is $O(Ns \times Nq)$. The procedure that generates the two global evaluation plans collects all the remaining non empty methods/functions that are in the array. Then, it generates a new single dimensional array for each global evaluation plan made up of the list of functions that are to be executed. The computational cost of the algorithm is $O(Nq \times Ns)$. Finally, in order to

evaluate and prove the cost effectiveness of the Query Melting Processor a theoretical evaluation has been conducted and was able to quantify the cost and profit of each phase. The results of the evaluation concluded with a net profit value which reflects either a time saving meaning that it is cost effective or a time loss meaning that it is not.

As a conclusion, the query process is designed and implemented based on the query melting paradigm, query optimization, time cost optimization. The query melting paradigm includes sharing the functions, objects, spatial areas, time intervals. The query optimization includes decomposition of the queries, common sub-expression elimination, generation of global execution plan, and evaluation of the plan. The time cost optimization is performed through sharing the global execution plans.

Chapter 7 - Case Study: Proximity Analysis for Multiple Dynamic Complex Queries Using Paris Map

7.1 Introduction

This chapter presents the integration of the proposed Iconic Visual Query Language (IVQL) together with the new optimization strategies namely, Query Melting paradigm, Query Optimization, and Cost Optimization. Its emphasis is to explore the Query Melting Processor on the k-Nearest-Neighbours (kNN) as well as Buffer queries in the context of dynamic complex queries. Its main aim is to give an experimental evaluation of the system, using a case study which aimed at a Tourist Mobile GIS application using the map of Paris. The user formulates the complex query which is sent to the geographic information system ArcGIS server. The query is processed and its result map is sent back to the user.

The major objectives are:

- To apply the IVQL user interface to formulate dynamic complex queries

- To apply the Query Melting paradigm through the following tasks:
  - CSE: Common Sub-expressions Elimination
  - Sharing objects of interest
  - Sharing spatial areas
  - Sharing Time
  - Sharing the underlying space (Map) as an intermediate result

- To apply Query Optimization

- To optimize the time cost of the processor

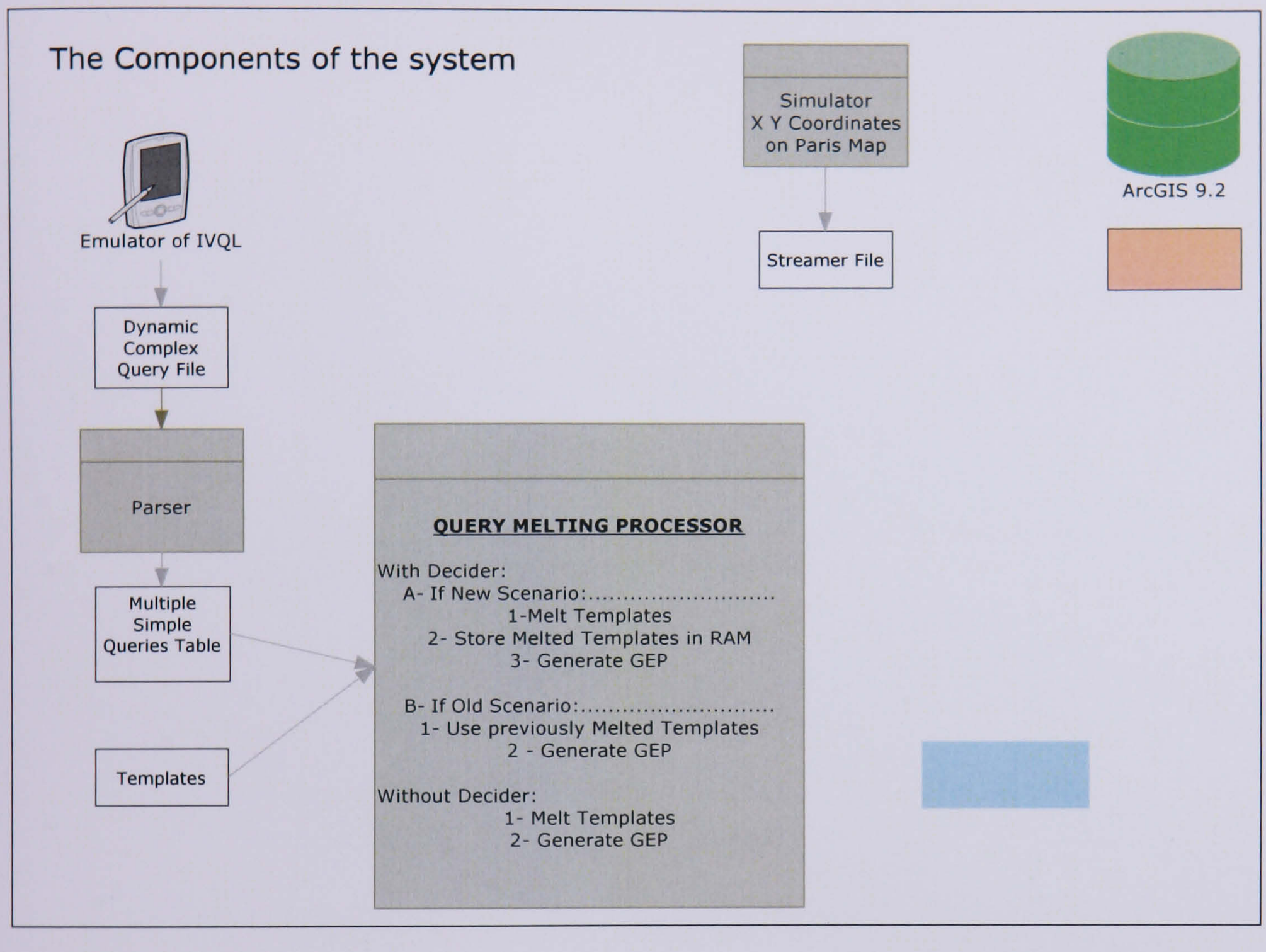
The IVQL is implemented using a mobile phone emulator in order to allow the visual query formulation of multiple dynamic complex queries. The Query Melting Processor which is responsible for the Query Melting Paradigm and the Query Optimization is implemented in VB.NET using the Microsoft Visual Studio 2005 API and the Desktop GIS ArcGIS version 9.2 components that are called ArcObjects. It performs common sub-expression elimination CSE by eliminating all the repetitions of the same process in different plans. It allows sharing object of interest between multiple queries by setting the object once and using it for many queries. When two queries share the same spatial area or when an area is included in another, the QMP uses the shared area for multiple queries. It allows sharing an interval of time if it is included or equal to the interval of another query with the same object of interest. Sharing the intermediate results is done through using the same underlying space (map) for all the queries of the same user. This is done through executing the plan of each user as a thread.

The Query Optimization is also achieved by the system through decomposing the dynamic complex queries, eliminating the sub-expressions, generating the Global Evaluation Plan, and evaluating the plan. The last objective is Cost Optimization which is to optimize the time cost of the processor. This is achieved through an experimental evaluation of the time cost of the Query Melting Processor. Two methods of processing are compared. The first is "With No Decider" where the complex queries templates are melted for each complex query. The second is "With Decider" where the templates are melted once per scenario and stored in memory for later retrieval by similar consequent scenarios. A comparison between the time costs of the two processing methods is enough to conclude which of them is cost effective.

The chapter is divided into four major sections. The components of the system are explained in section 7.2, an explicit scenario for three users only is presented in order to show how the system works, the experimental evaluation is detailed in section 7.3, and the summary with the conclusion are presented in section 7.4.

7.2 Components of the System

All the components of the case study are shown in Figure 7.1. The Emulator of IVQL is used to formulate dynamic complex queries. The parser receives the complex queries and converts them into multiple simple queries.



PAGE NUMBERING AS ORIGINAL

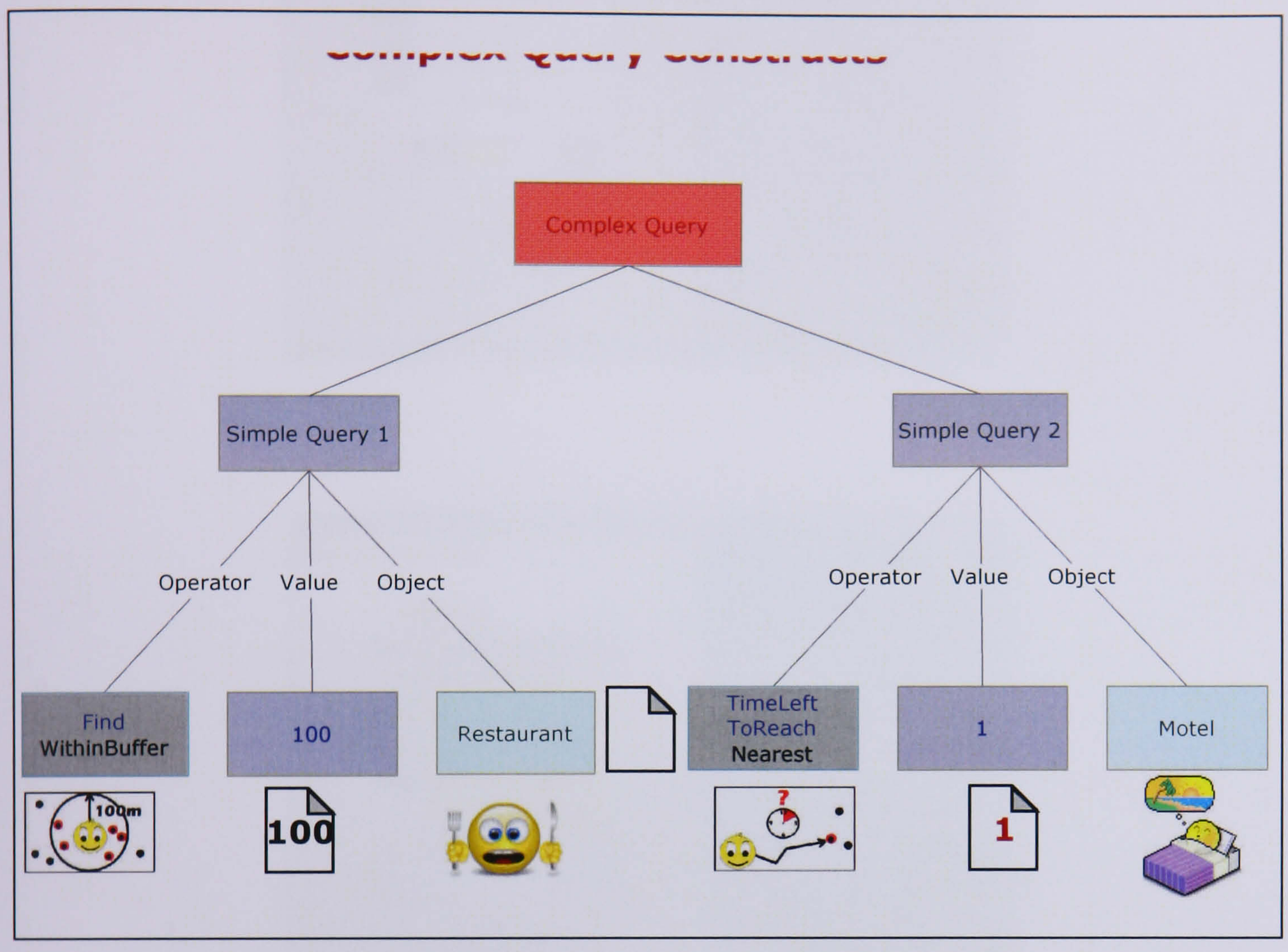
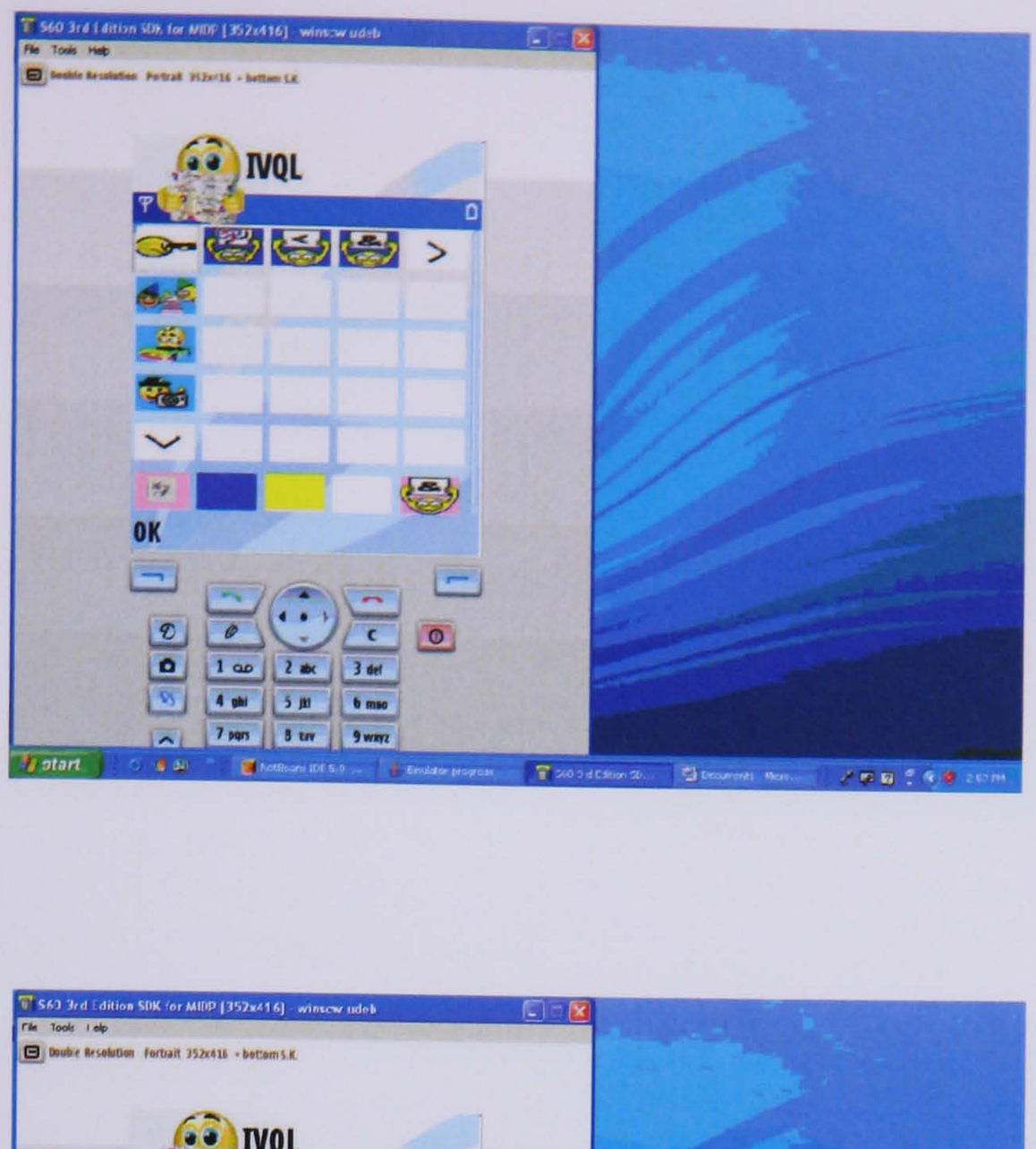
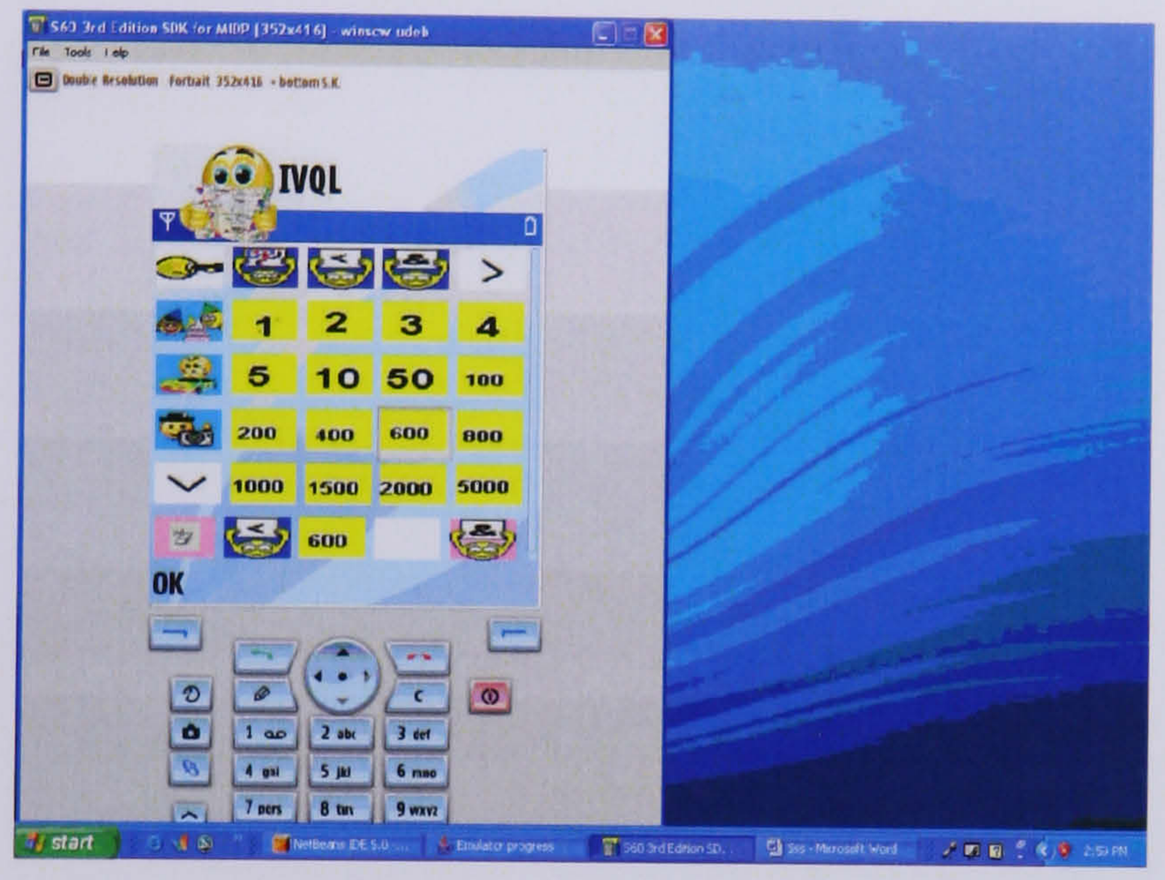


Figure 7.3: The Constructs of a Complex Query.

The IVQL graphical user interface is used to formulate simple and complex visual queries using the emulator as shown in Figures 7.4(a) and 7.4(b). They are automatically translated to regular complex queries. The user mobile number and his current X Y Location are concatenated with the complex query. The resulting text is sent as a tuple of a single-tuple table to a file folder for processing. A Watcher informs the Parser about the arrival of any file in due time.





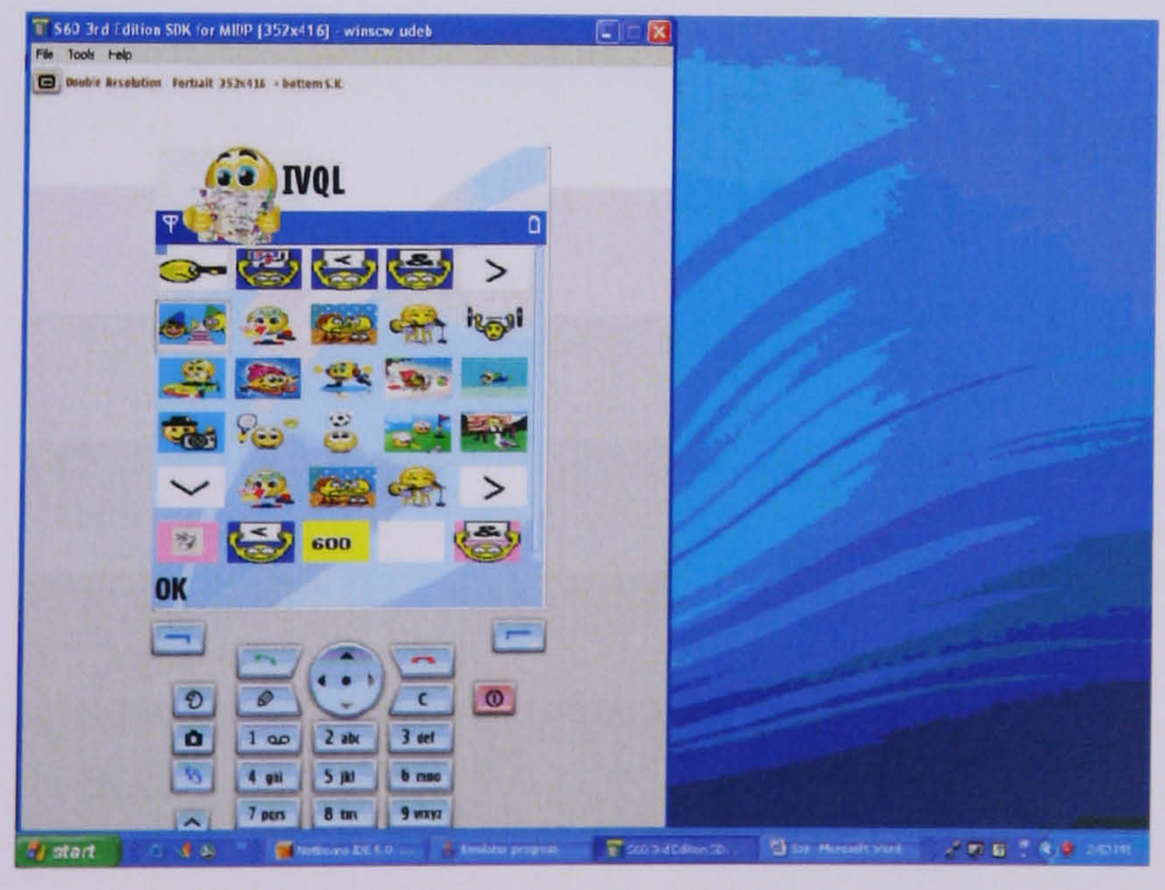


Figure 7.4(a): Some Snapshots of the IVQL Graphical User Interface.

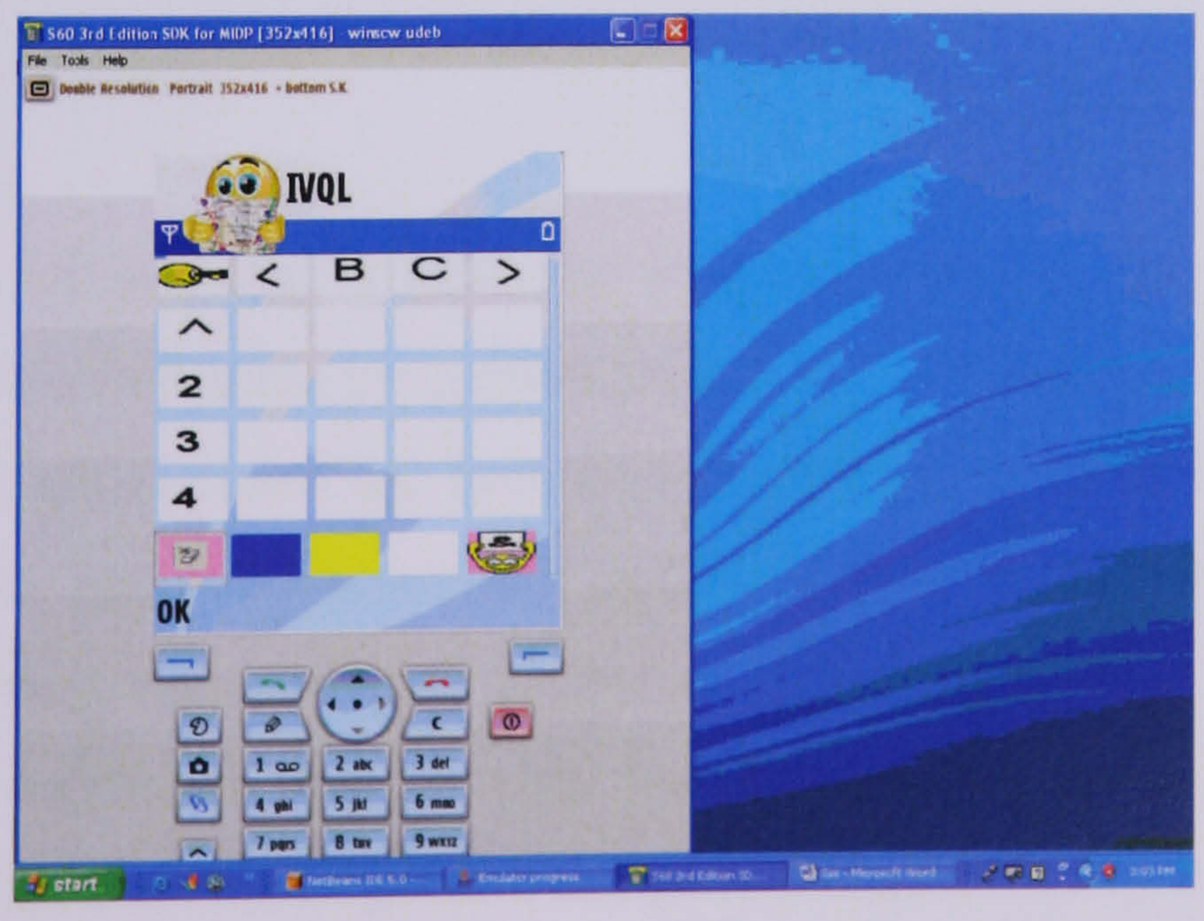


Figure 7.4(b): Some Snapshots of the IVQL Graphical User Interface.

All the operators and objects that were explained in chapter 5 are used to formulate complex queries called data files hereafter. The data files are generated varying the number of complex queries respectively from 1, 2, 5, 10, 20, 50, to 100. For each data file, other data files are also generated varying the number of operators respectively from 2, 5, to 10.

7.2.2 The Parser

The Parser is invoked by the Watcher once a new complex query is created in the file folder. It converts the complex query into multiple simple queries and assigns their value in special arrays in the memory.

7.2.3 The Query Melting Processor

At the beginning of its execution, the Query Melting Processor reads the Templates of the operators that are included in the system. Each template has an evaluation plan for one query based on its operator. Moreover, the number of templates determines the number of possible scenarios a complex query could have. If t is the number of templates then the number of possible scenarios is $2^t - 1$. For example, suppose that the templates of Nearest and Buffer operators are included. The possible scenarios that can be produced are Nearest alone, Buffer alone, and both Nearest and Buffer, with each scenario being assigned a code. The Query Melting Processor reads the simple queries of each user at a time. The combination of their operators is converted to the code of the scenario.

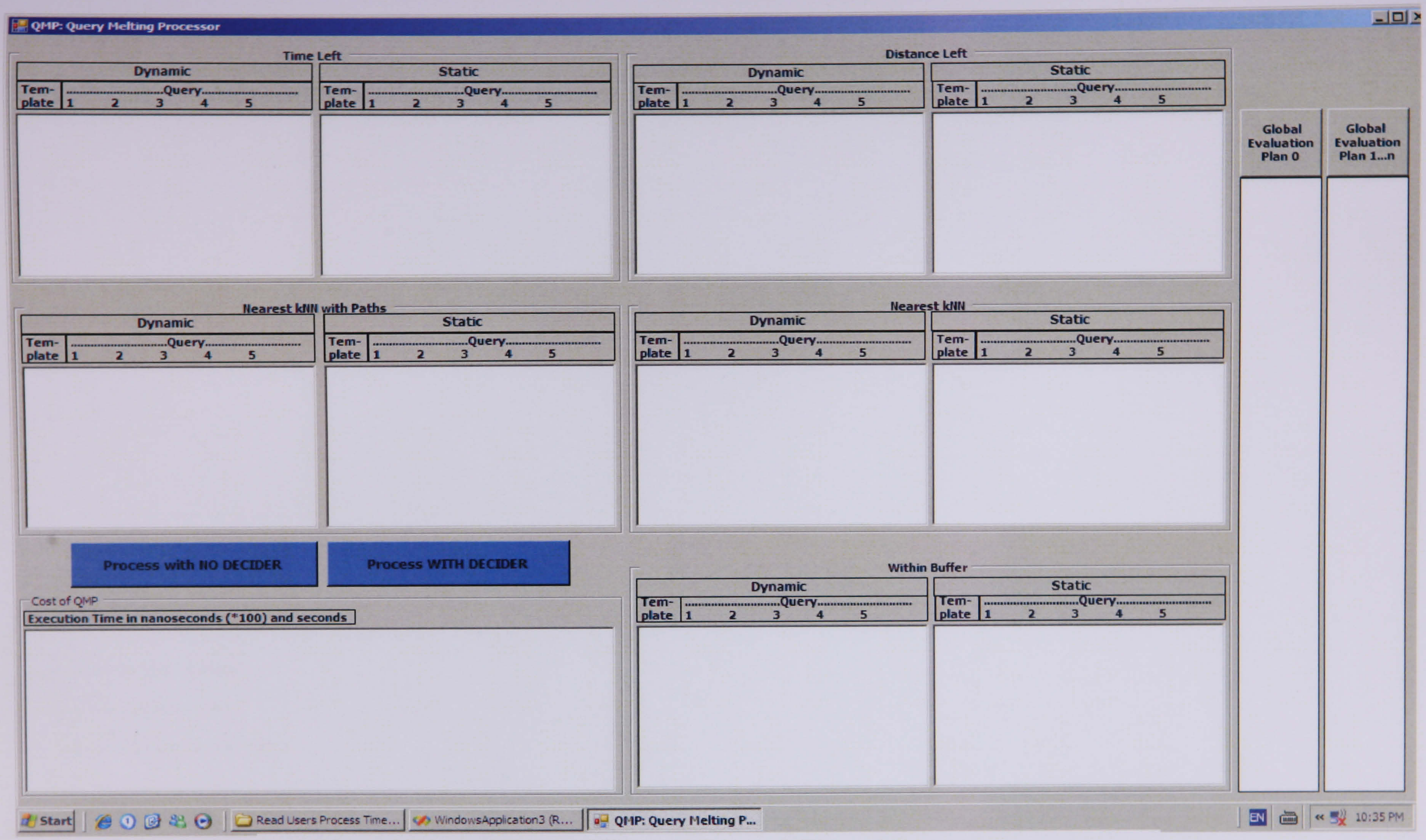
If the Decider is not activated, it melts the templates, decomposes the queries, melts them, and generates the Global Evaluation Plan. If it is, it checks if the combination of the operators that are included has been previously used. In other words it checks if the scenario code has been processed before. If yes, it uses the plan that was previously generated. Otherwise, it melts the templates, generates a new Melted Templates Plan, and stores it in memory for later retrieval. Then, it proceeds with decomposing the queries, melting them, and generating the Global Evaluation Plan.

While melting the templates, the processor is in fact performing common sub-expression elimination CSE. It examines the functions of the templates and eliminates any repetition that does not affect the result of the complex query. For example, the "make closest facility layer" method is repeated in all the templates concerned with proximity analysis. The processor keeps the first occurrence of the method and eliminates all subsequent occurrences.

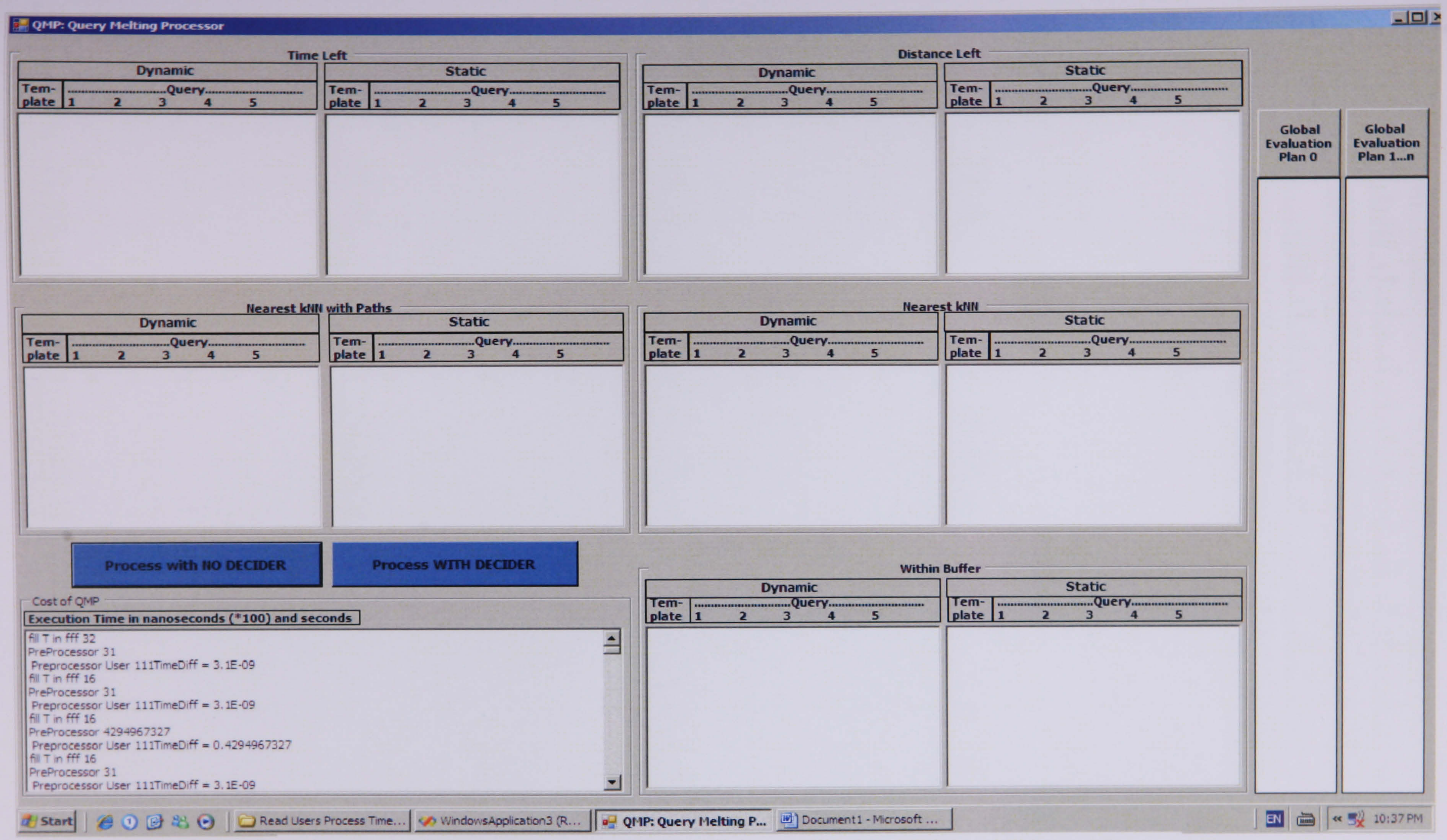
While melting the queries, the processor is also performing sharing object of interest. When two or more consecutive queries have the same object of interest, the processor keeps its first occurrence and eliminates all subsequent occurrences. It is enough to find the object of interest once and use it more than one time. For example, if the first query is to find the nearest hotel and the second the time left to the nearest 3 hotels, setting the hotel is done once and used to produce the results of the two queries.

It is also performing the sharing of spatial area. For example, if the first query is to find restaurants within 200 meters and the second within 100 meters, the processor eliminates completely the second query because it is totally included in the first. The same occurs for sharing the time interval. For example, if the first query is to find the nearest hotel for the next hour and the second for the next 2 hours the processor eliminates completely the first query as it totally time included in the second.

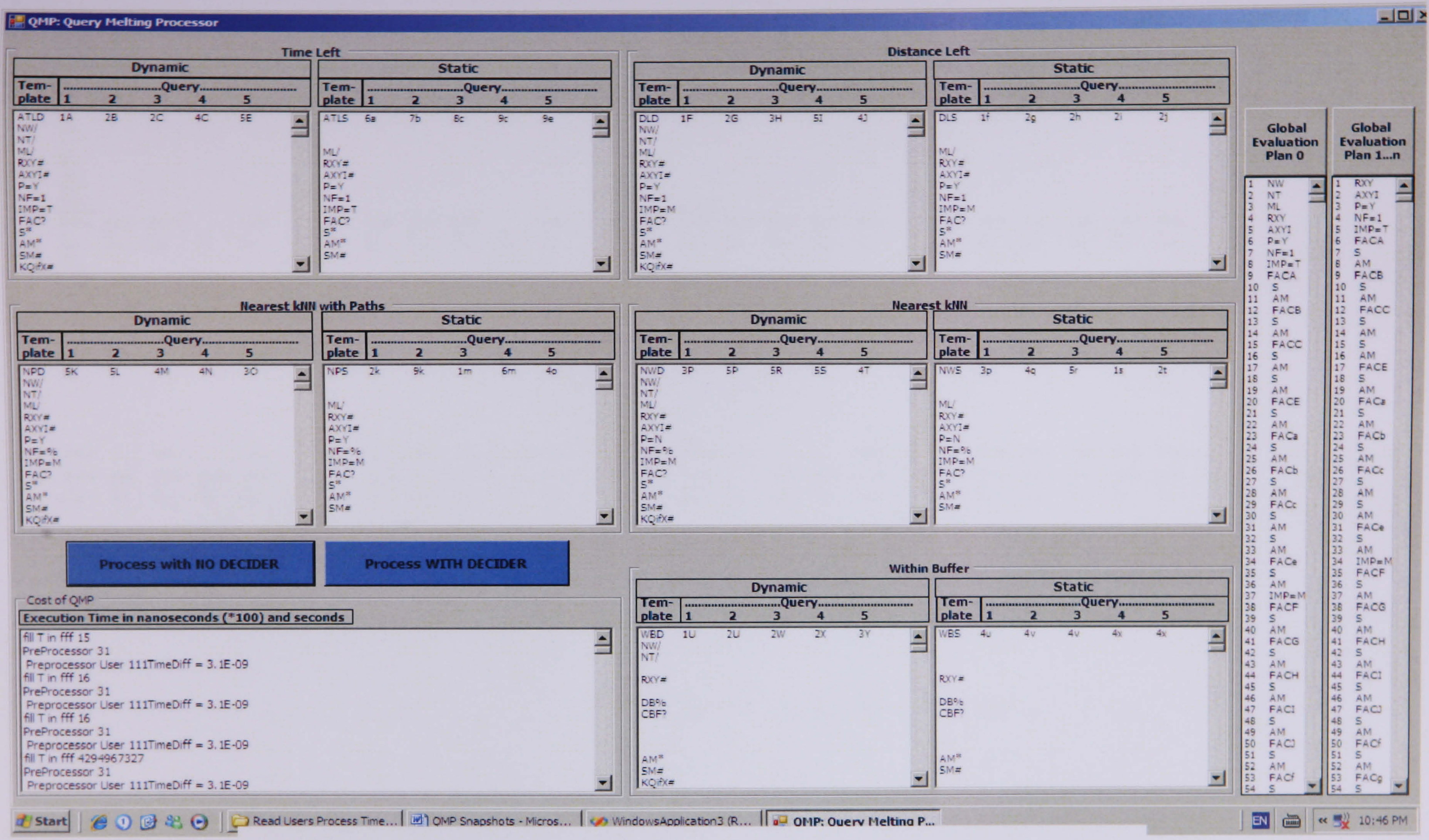
By using the Decider in the processor, the Cost Optimization is achieved. An experimental evaluation is carried out in order to evaluate the cost effectiveness of using the Decider. In the experiment, the queries are processed on the fly so as to avoid any I/O overhead. The Melted Template Plans are stored in RAM but not on secondary storage media. Also, a special version is implemented in order to allow the activation of the Decider. Two buttons are used. The first is named "Process with No Decider" and the second "Process with Decider". In order to get more accurate results, all the functions that display data in ListBoxes are disabled to eliminate their time cost effect. Figures 7.5 to 7.8 show snapshots of the user interface while the processor is running.



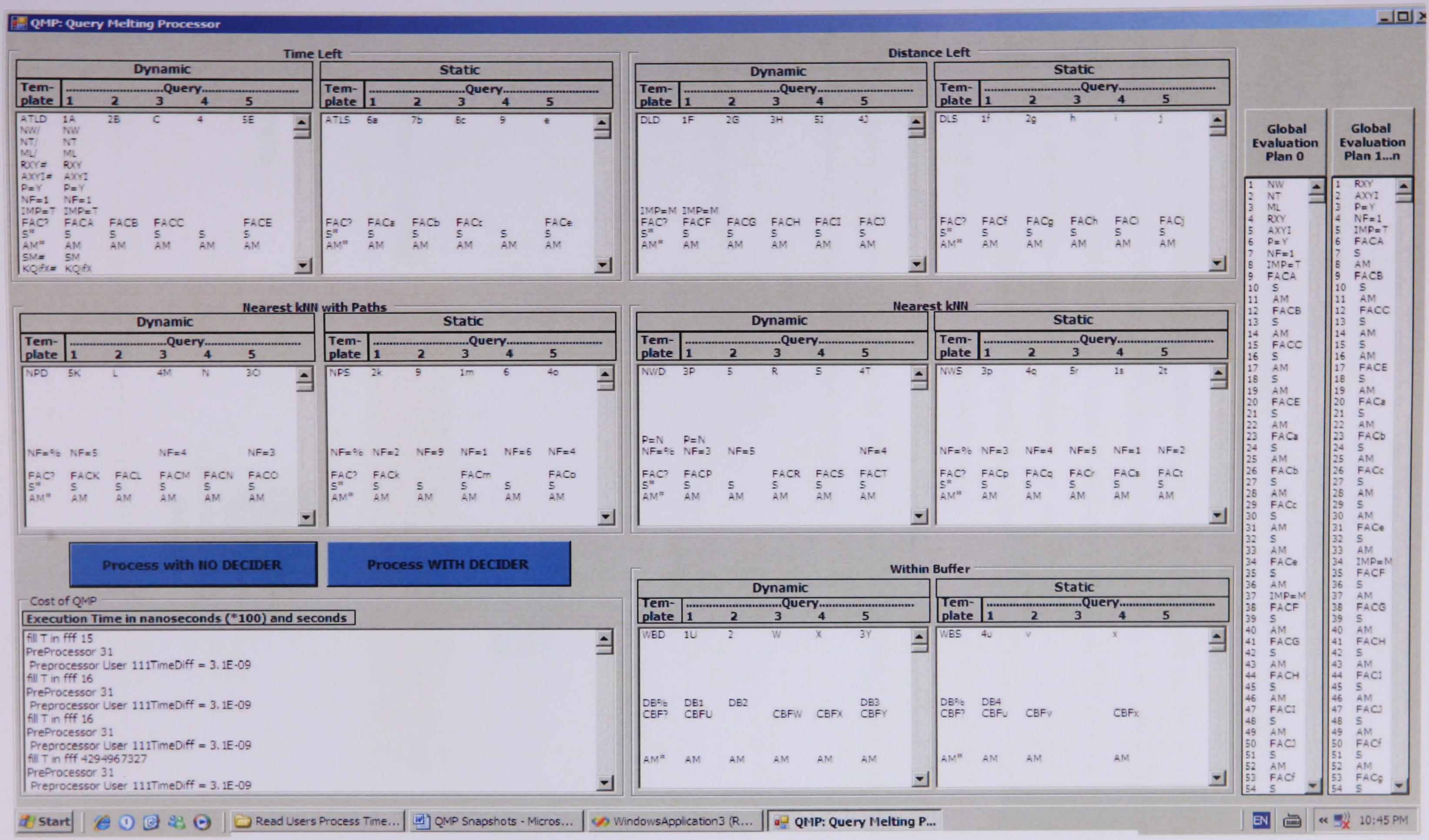
[Figure 7.5: The Query Melting Processor Graphical User Interface at Start-Up.



[Figure 7.6: The Query Melting Processor Running without Displays.



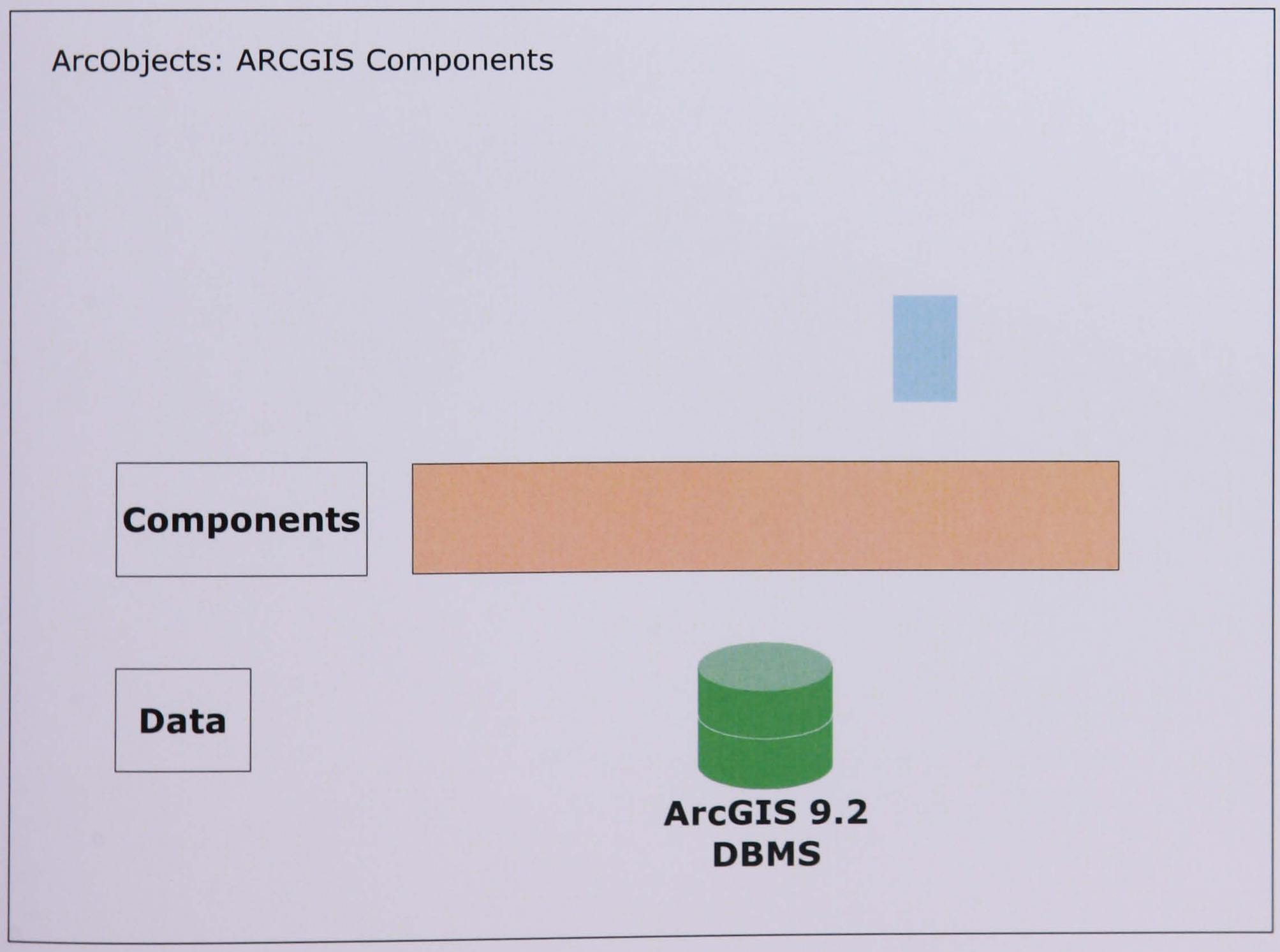
[Figure 7.7: The Query Melting Processor before Melting 50 Queries of a user.



[Figure 7.8: The Query Melting Processor after Melting 50 Queries of a user.

7.2.4 ArcGIS Objects

The underlying Geographic Information System is the ESRI ArcMap and ArcView version 9.2 which are products of the ESRI ArcGIS Software that offers software and technology for desktop mapping, online maps, GIS data, and products [ESR]. The ArcMap allows users to access its objects, controls, and toolboxes. They products are used to build a complete GIS in the areas of geo-processing and map visualization by single users or multiple users on desktops or servers and are based on a common library called ArcObjects which consists of shared GIS software components. The wide variety of these programmable components allows developers to use them with standard programming frameworks. Before starting to program an application, the ArcGIS objects and controls must be loaded into the development environment. Figure 7.9 shows how the ArcObjects can be used.



The ArcObject Geodatabase is used for handling workspaces, datasets, and feature classes. The ArcObject Controls allow the developer to insert controls in their forms. The control MapControl is used to add a map to the form and later zoom in, zoom out, and pan it. The ArcObject Carto is used to handle Layers and add layers to maps. The ArcObject NetworkAnalyst is used to make closest facility layers, add incidents, add facilities, set the parameters such as the number of facilities to find, the impedance attribute (Meters,..), show the path, travel directions, etc. The ArcObject AnalysisTool is used to handle a variety of tools such as the Buffer that draws a buffer circle around the user location and the Clip that finds facilities in the buffer area.

7.2.5 Simulation of Mobile User X Y Locations

A simulator is used to play the role of a satellite streamer that supplies the locations of a mobile phone user who is on the move. The user interface of the simulator is shown in Figure 7.11.

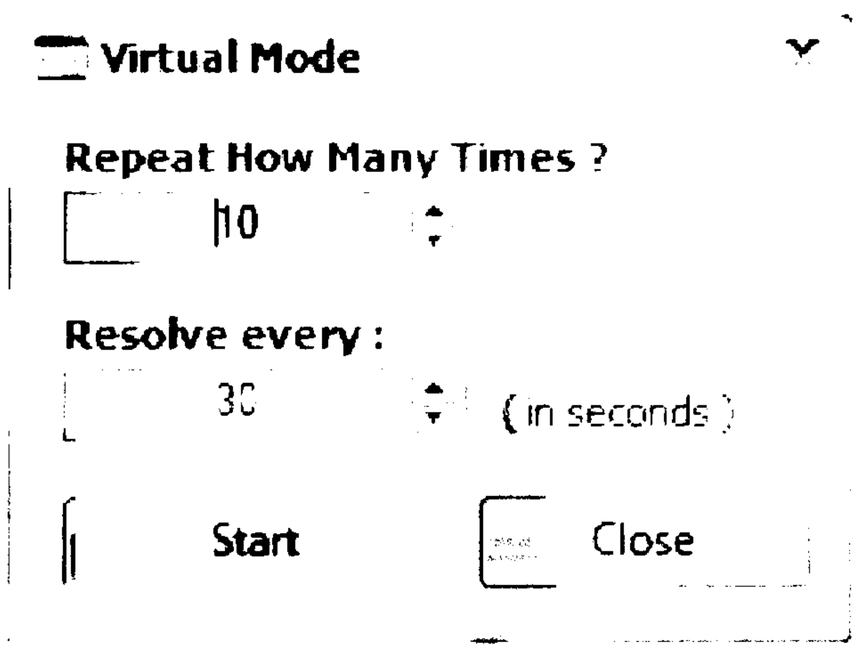


Figure 7.11: The Simulator User Interface.

The first input value specifies the number of locations to be generated for each mobile user. The second input value specifies the interval of time between two locations. The simulator launches or invokes a timer and at every new time interval it generates random X Y coordinates taken from the map of Paris. The output of the simulator is a text file that contains for each user multiple records each of which representing a new location. Each record includes the mobile number, date, time, speed, X coordinate, and Y coordinate. The simulator during execution is shown in Figure 7.12 using the map of Paris to generate the random locations.

Figure 7.12: The Simulator while Generating Locations from the Map of Paris.

7.2.6 Plan Execution of the Global Evaluation Plans

The Plan Evaluator is responsible for executing the Global Evaluation Plan of each dynamic complex query. It processes the methods sequentially by calling the corresponding private functions. Each private function uses the ArcObjects to execute the command on the map and the ArcGIS database. If the complex query is made up of static operators only the job is terminated after displaying the map. If it contains one or more dynamic operators it processes the plan and displays the result map for every location received by the streamer.

The Plan Evaluator processes the complex queries as a Multithreading task where a thread is created for each user. Thus, the Plan Evaluator is in fact performing Sharing the Underlying space (map) and intermediate results. The Paris map remains active in the thread of the user as an intermediate result, with all the queries and their results are shown on it.

7.2.7 Result Map

The result of processing the plan is the map of Paris showing the output of the dynamic complex query based on the location of the mobile user at a particular time instance. The result map is supposed to be sent to the mobile user as a ping file (.png) through Multimedia Messaging Service (MMS). But since an emulator is used for the purpose of experimentation in order to facilitate the implementation the map is displayed within the current implementation environment. Figure 7.13 shows an example result map of Paris including the output of the dynamic complex query.

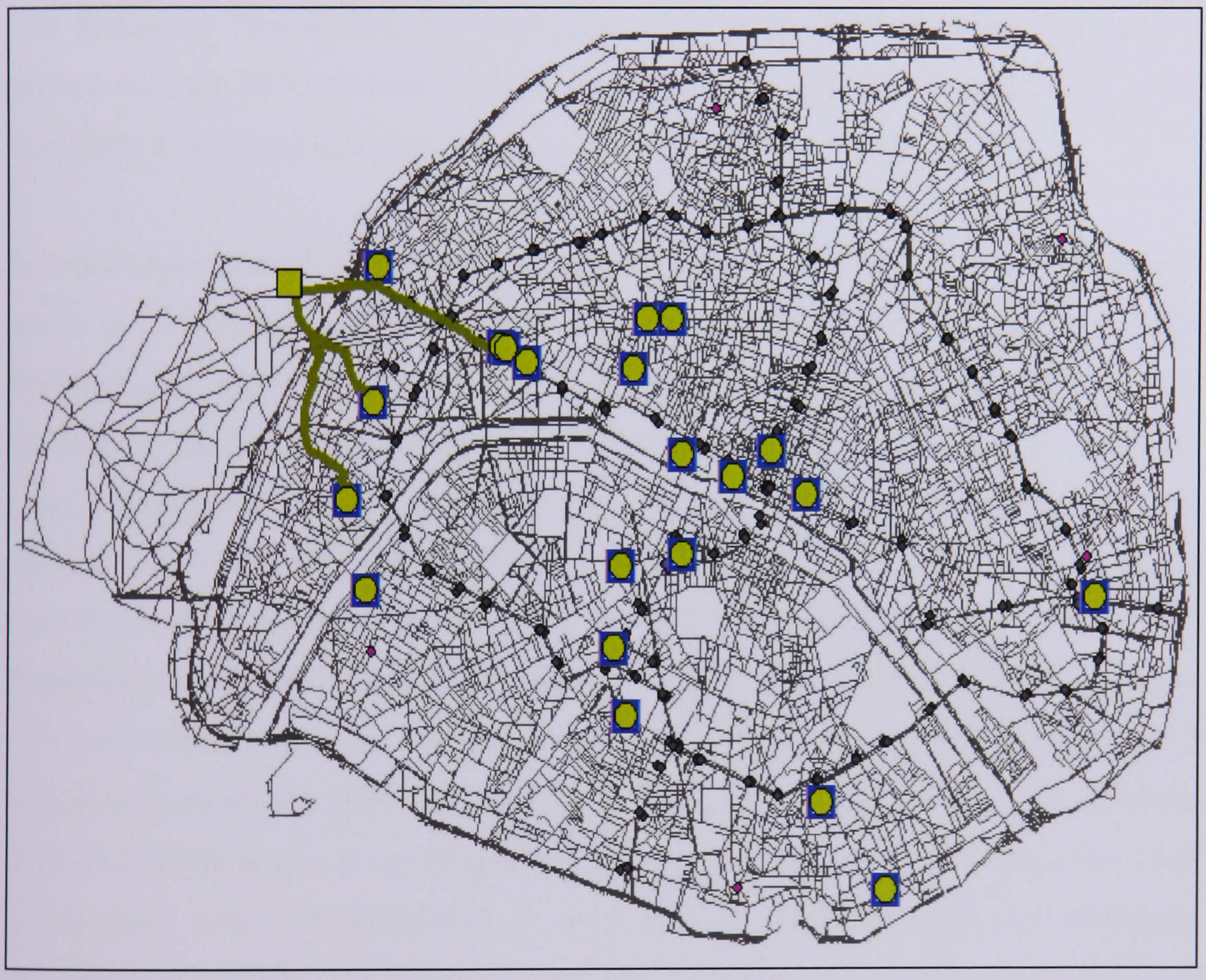


Figure 7.13: The Map of Paris showing the Result of a Dynamic Complex Query.

7.3 An Explicit Scenario

The explicit scenario goes through the user interface of IVQL, the Query Melting Processor (QMP), and the result map in action with the aim to show how the system works. The scenario is about three mobile users where each user formulates one dynamic complex query (DCQ), the QMP processes each DCQ, and the result map of each user is displayed. The QMP is executed twice to process the three DCQ, once without the time cost optimizer (NO DECIDER) and once with the time cost optimizer (WITH DECIDER), with the purpose to show that processing the three DCQ with the time cost optimizer is faster and more cost effective than without it. The difference between the two strategies is that in the first one the templates are melted for each DCQ whereas in the second one the multiple DCQ that have similar scenarios, formulated using the same operators, share the same melted plan.

In order to consider all the possible combinations, user 1 formulates his DCQ which is made up of 4 simple queries using the two operators NPD (Find Nearest with Path Dynamically) and WBD (Find Within-Buffer Dynamically), where the first 2 simple queries use the NPD operator and the other 2 simple queries use the WBD operator. User 2 formulates his DCQ which is made up of 5 simple queries using the three operators NPD, WBD, and WBS (Find Within-Buffer), where the first 2 simple queries use the NPD operator, the next 2 the WBD, and the last one the WBS. User 3 formulates his DCQ which is made up of 3 simple queries using the two operators NPD and WBD, where the first simple query uses the NPD operator and the other 2 simple queries use the WBD operator. Both user 1 and user 3 use the same operators NPD and WBD, hence, it can be said that they have similar scenarios. When the QMP is executed with NO DECIDER, it melts the templates of each user operators. Whereas with WITH DECIDER, it melts the templates of the user 1 operators and stores them in RAM because they use a new scenario, does the same for user 2, and uses the stored Melted Template Plan of user 1 for user 3 instead of melting user 3 templates because user 3 has the same scenario as user 1. This is how the new Sharing Plans paradigm is implemented and why the new time cost optimizer is cost effective, because it is saving the time to melt the templates of similar scenario DCQ.

7.3.1 IVQL User Interface

The DCQ of user 1 is to dynamically find the nearest 4 restaurants and 4 undergrounds (Metro stations), and dynamically find within 2000 m all undergrounds and hospitals. The mobile phone GPS system is set to update the GIS server with the user location every 5 minutes. Figures 7.14(a) and 7.14(b) show the IVQL user interface of each simple query of user 1. The "AND" operator is selected between the simple queries in order to formulate the complex one. After the complex query is formulated, it is shown in text only in this work in order to check and validate that the queries are properly translated as shown in Figure 7.14(c). When the user selects the "Send" command, the complex query is sent to a special folder where a listener launches the pre-processor as shown in Figure 7.14(d). The DCQ of user 2 is to dynamically find the nearest 3 restaurants and 2 hospitals, dynamically find within 1000 m all hospitals and undergrounds, and based on my location now find within 1000 m all restaurants. Figures 7.15(a) - 7.15(e) show the IVQL user interface of each simple query of user 2. The DCQ of user 3 is to dynamically find the nearest 2 hospitals, dynamically find within 2000 m all restaurants, and dynamically find within 1000 m all undergrounds. Figures 7.16(a), 7.16(b), and 7.16(c) show the IVQL user interface of each simple query of user 3.

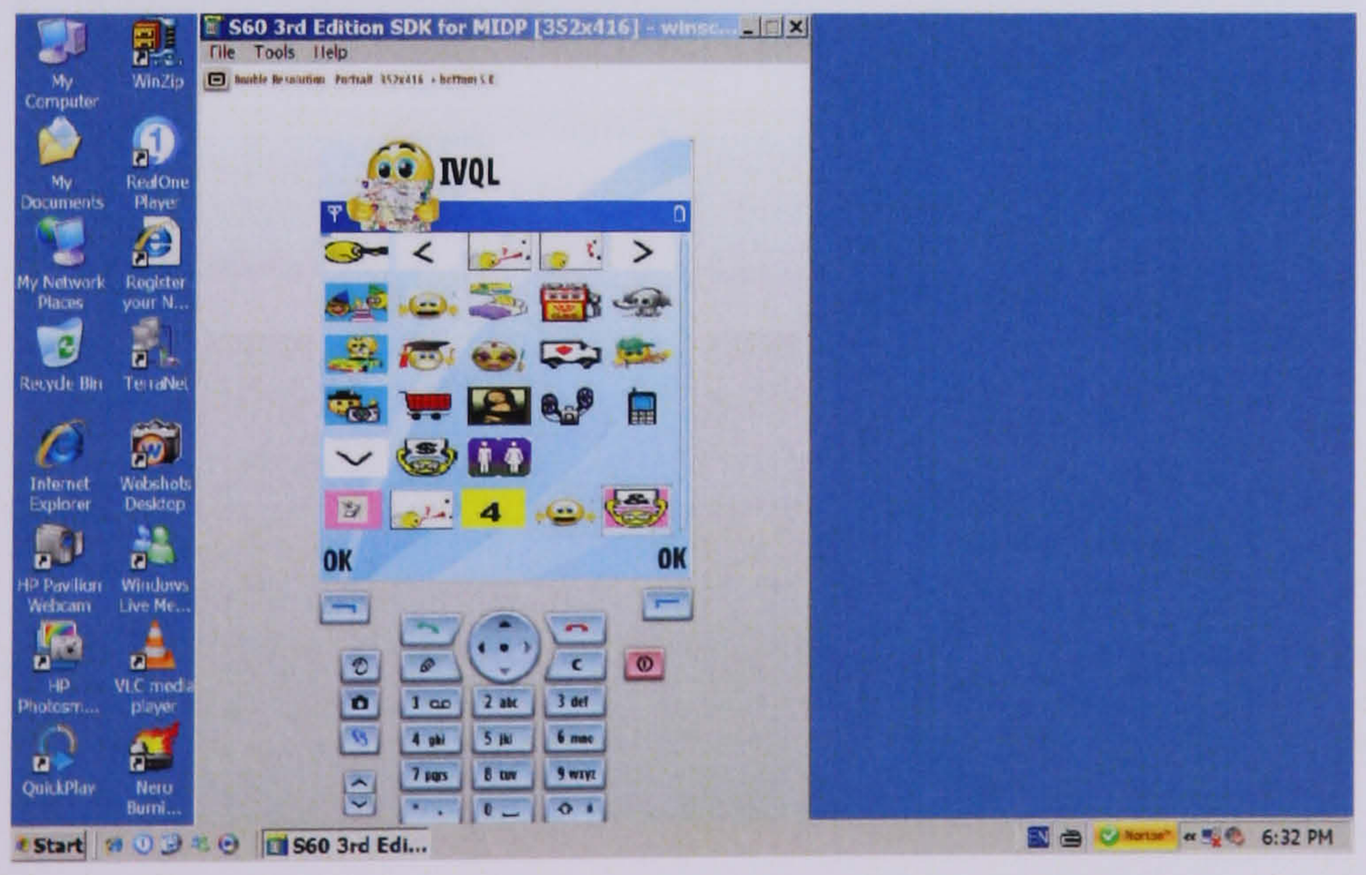


Figure 7.14(a): Query 1 of User 1.

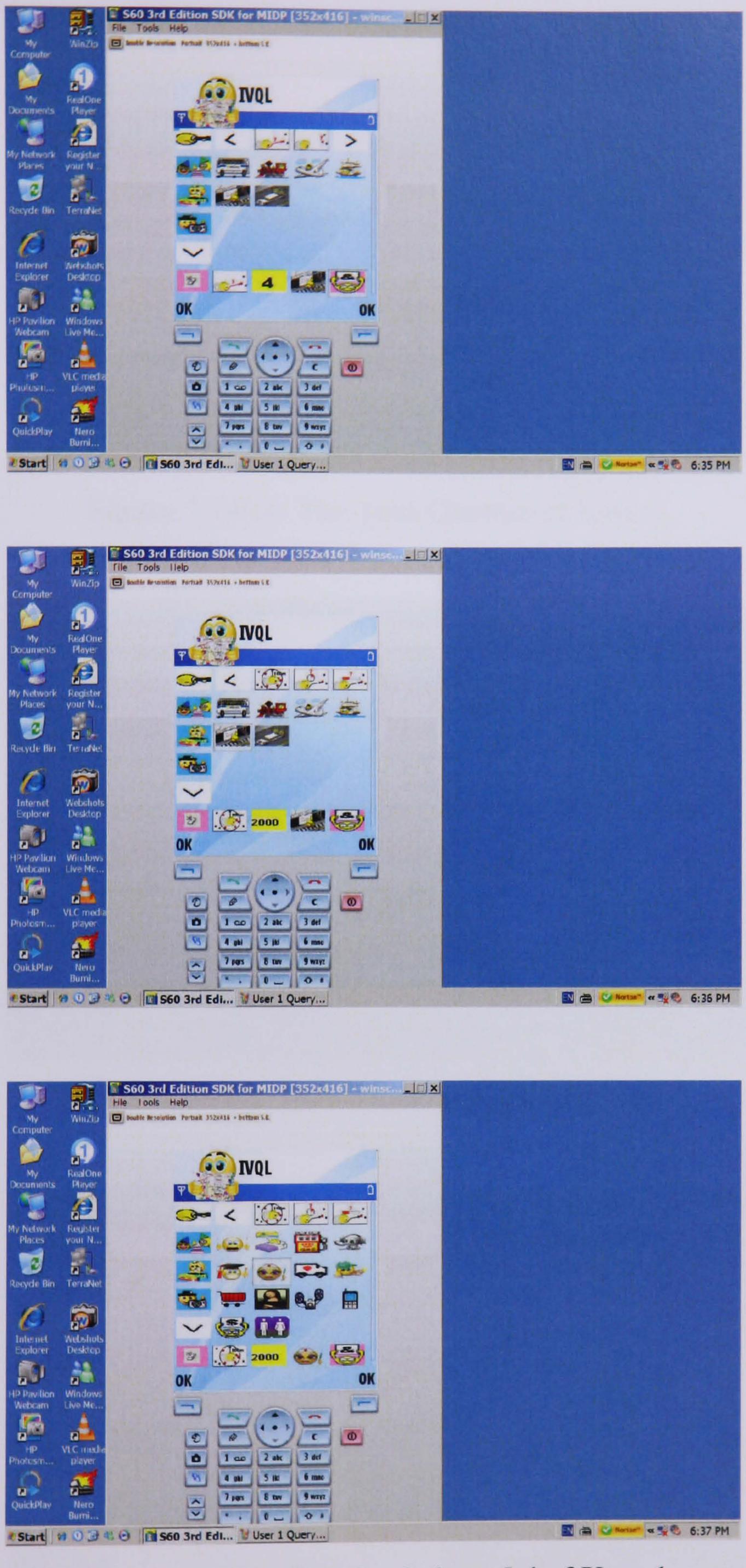


Figure 7.14(b): Queries 2, 3, and 4 of User 1.

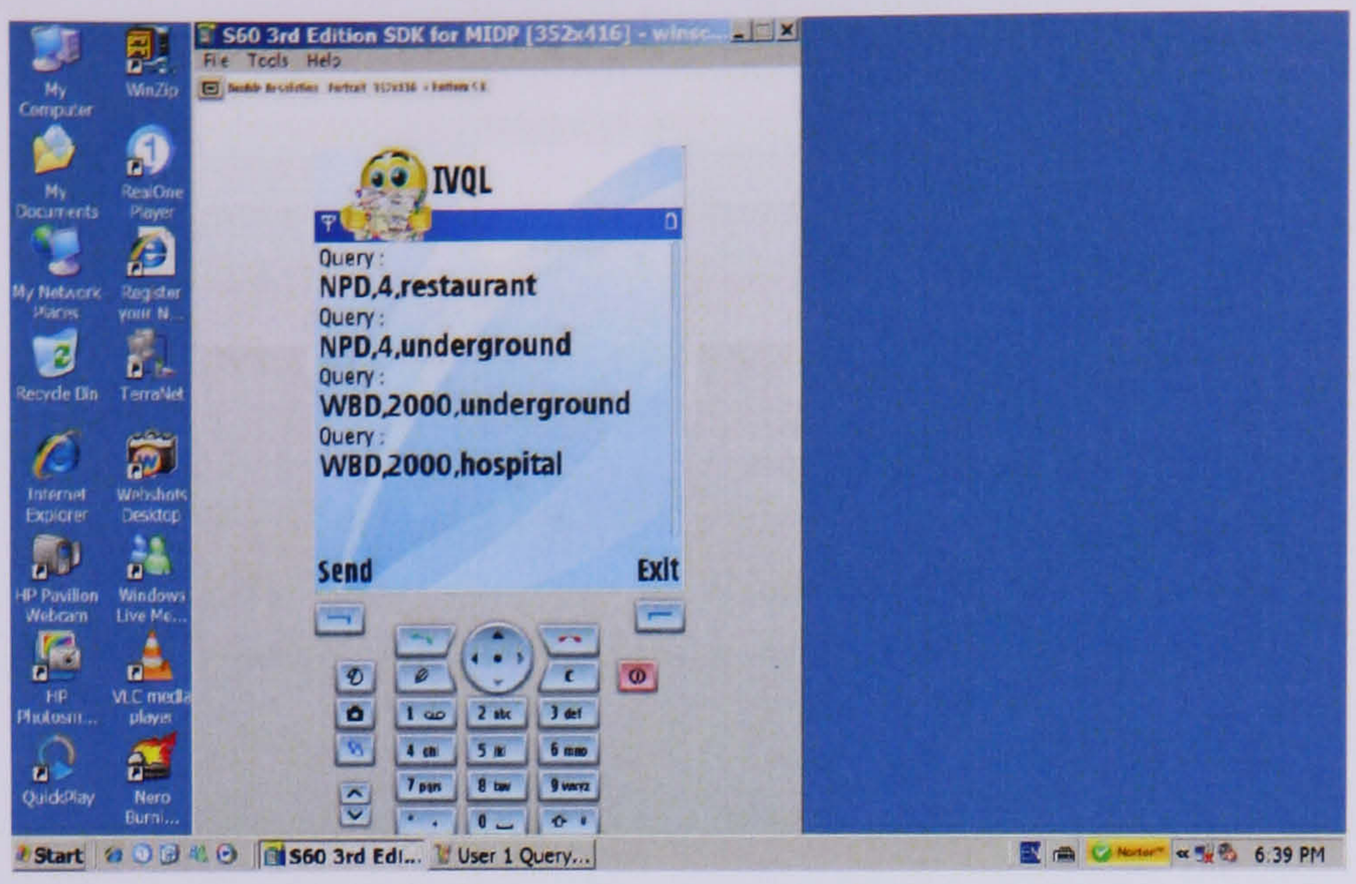


Figure 7.14(c): The Text Queries of User 1.

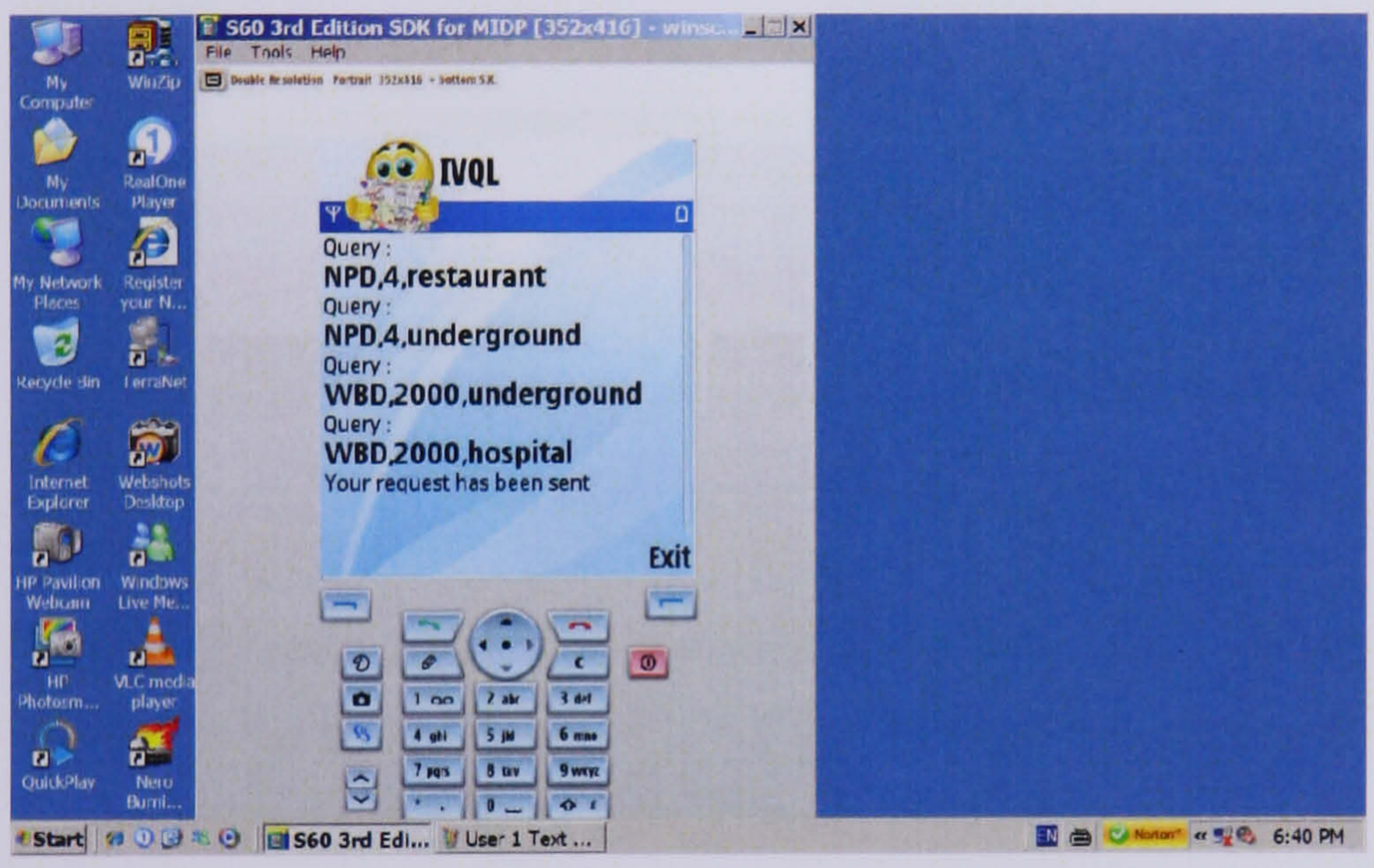


Figure 7.14(d): The SEND command of User 1.

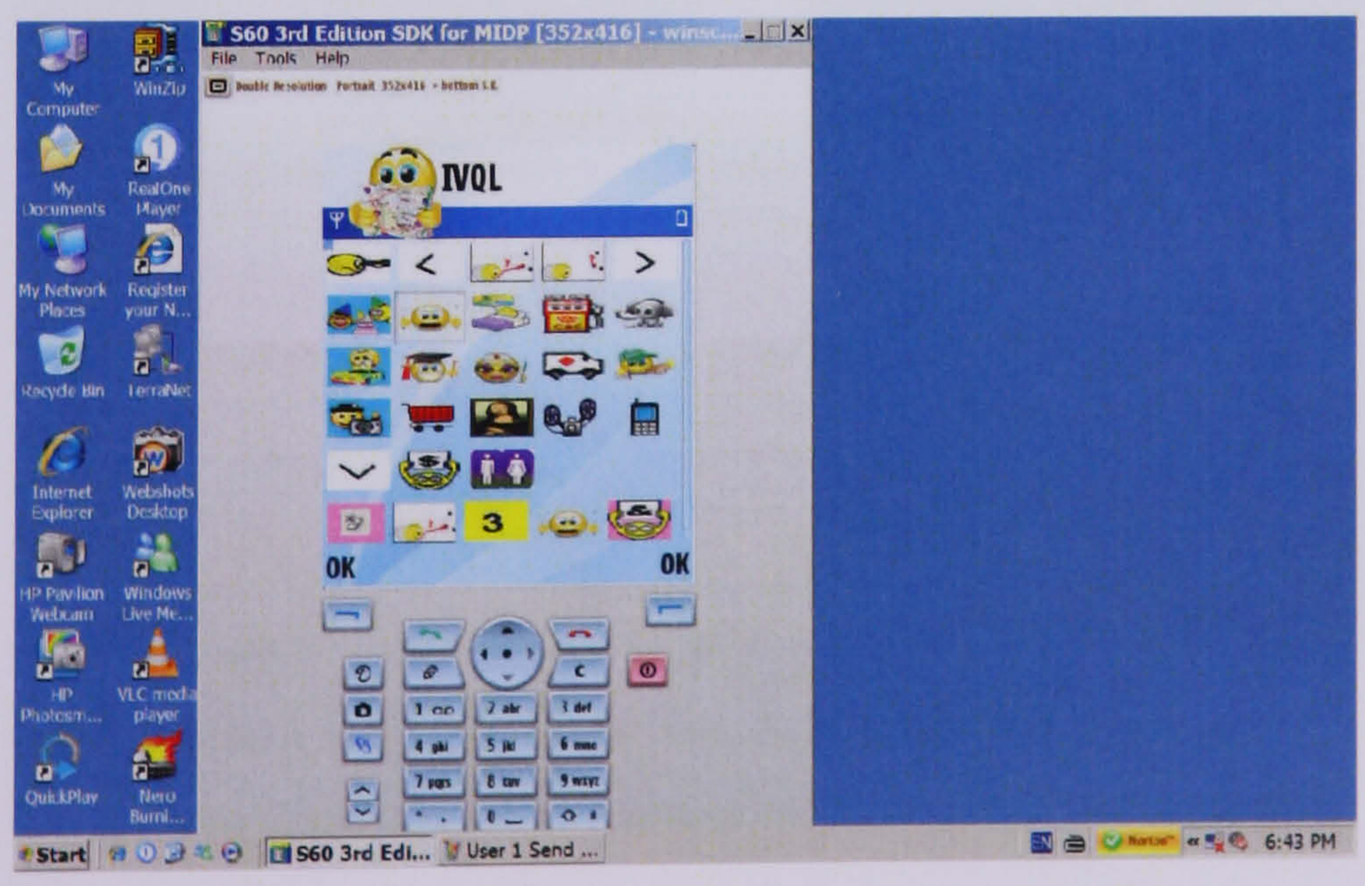


Figure 7.15(a): Query 1 of User 2.

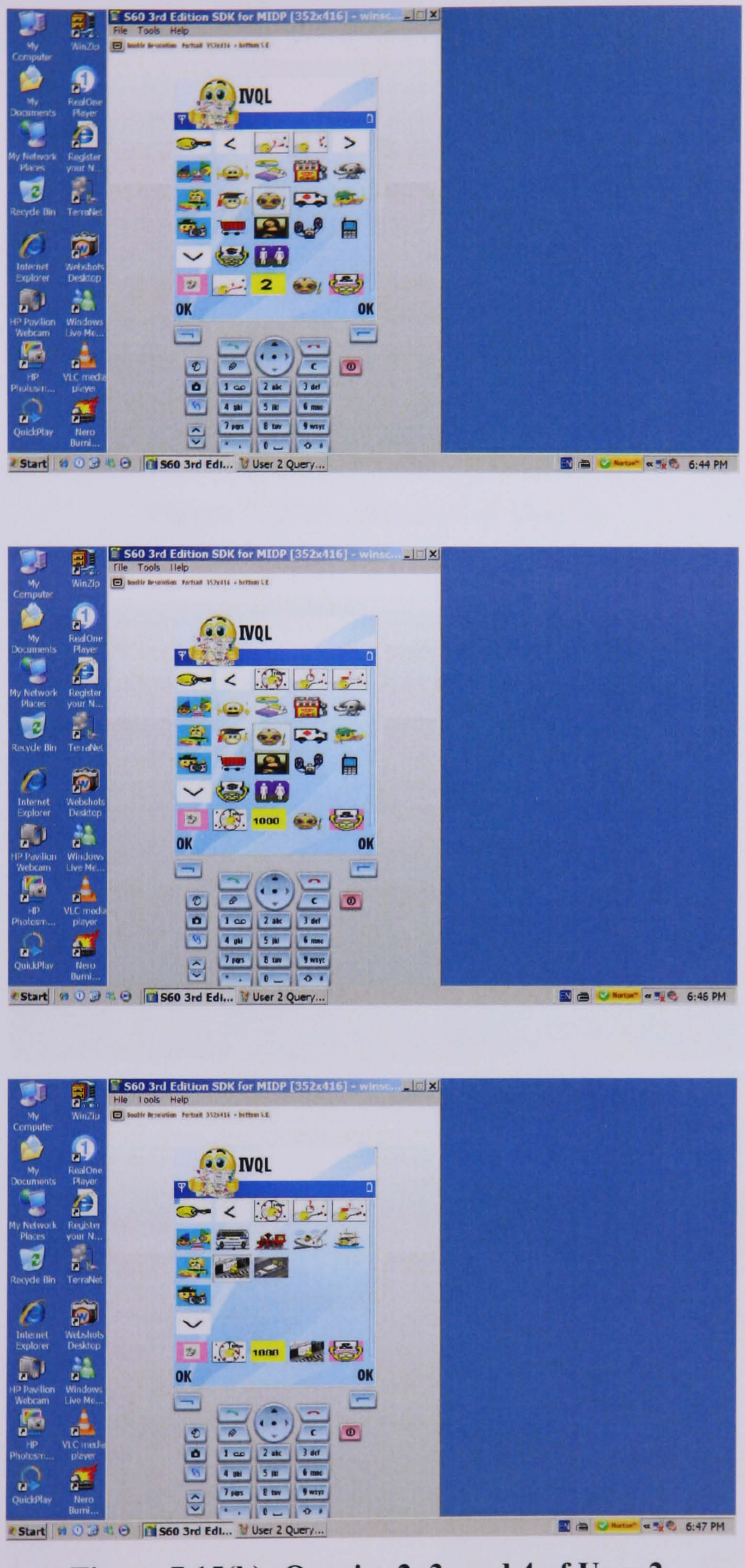


Figure 7.15(b): Queries 2, 3, and 4 of User 2.

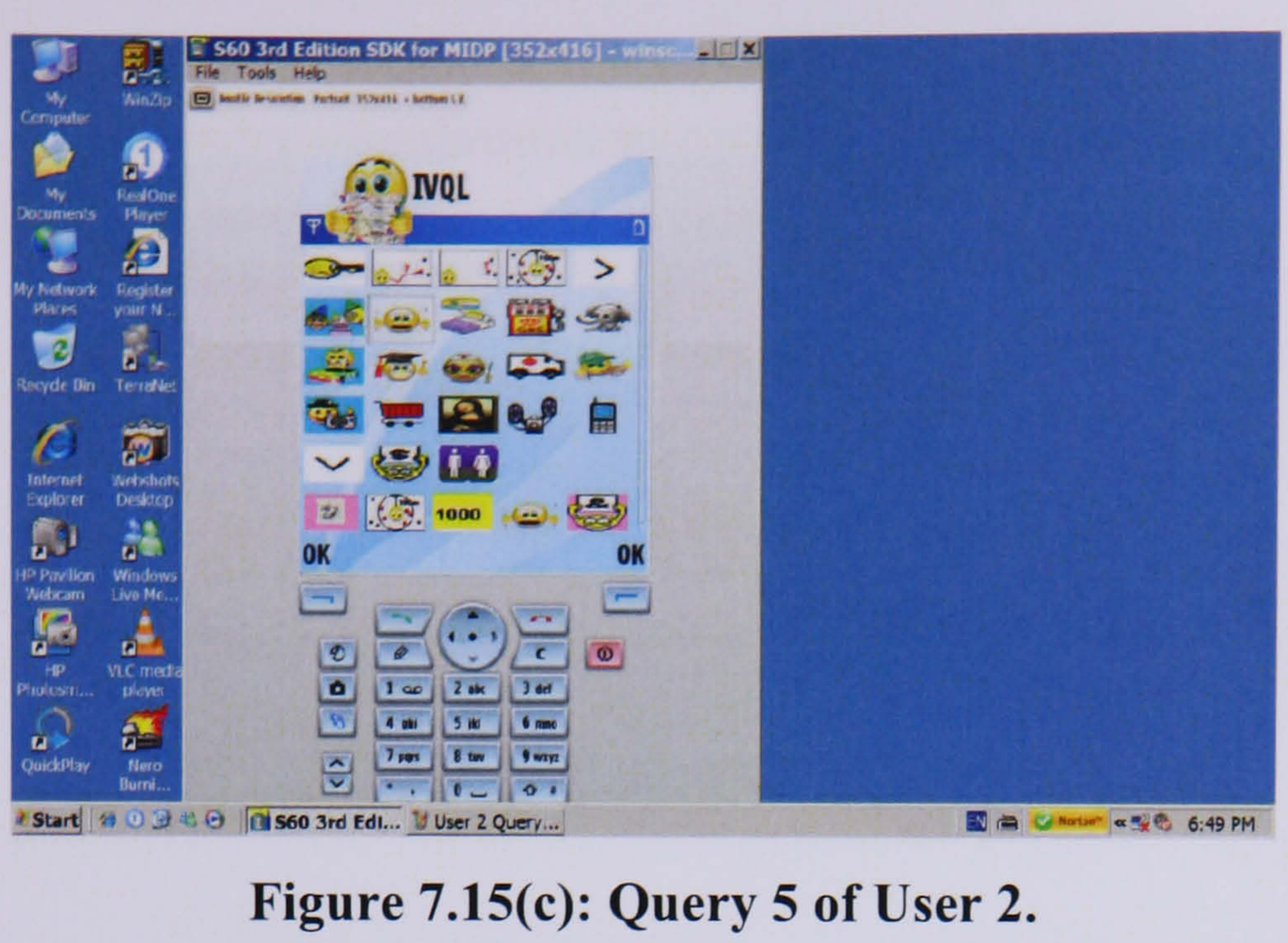


Figure 7.15(c): Query 5 of User 2.

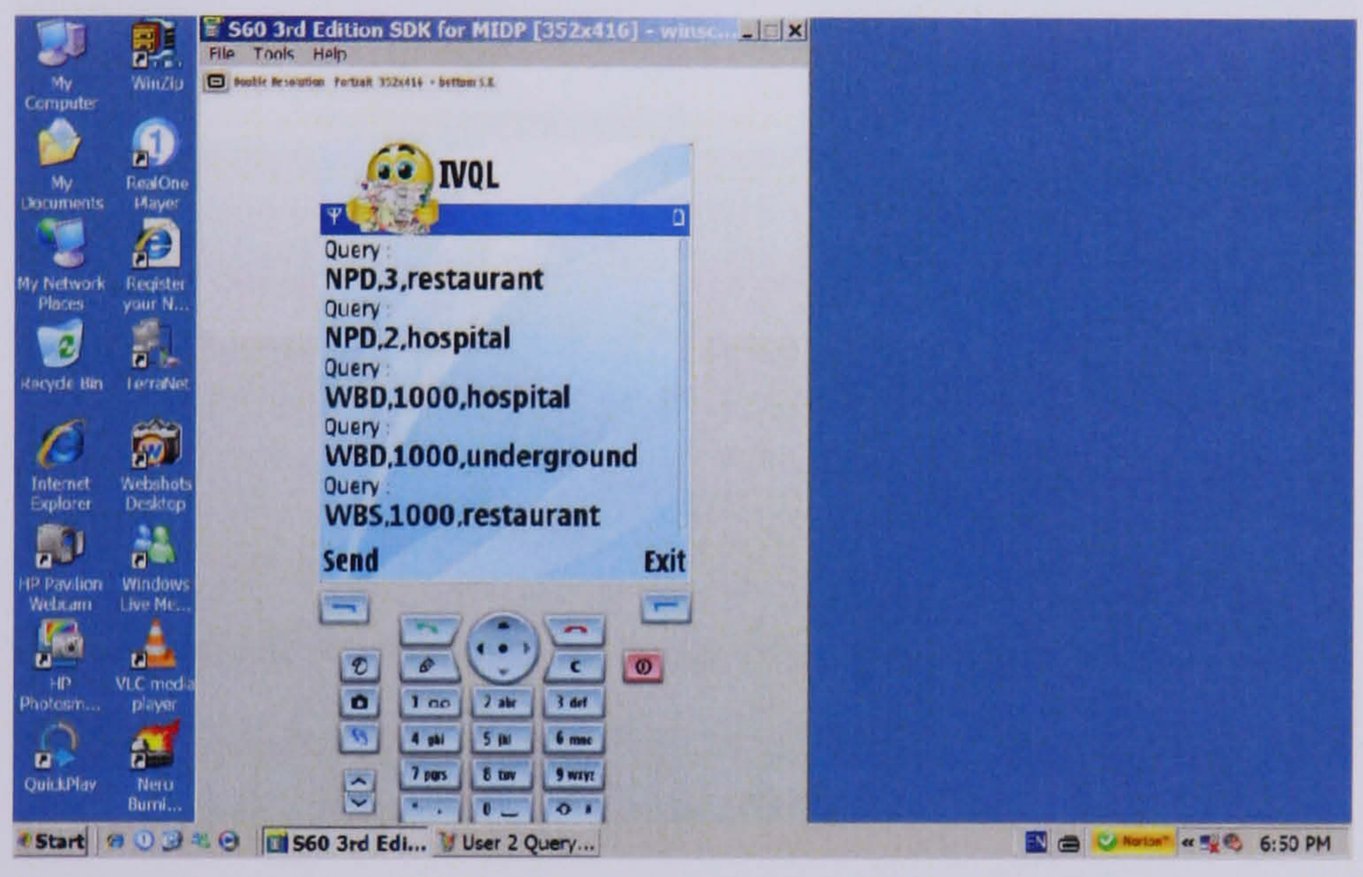


Figure 7.15(d): The Text Queries of User 2.

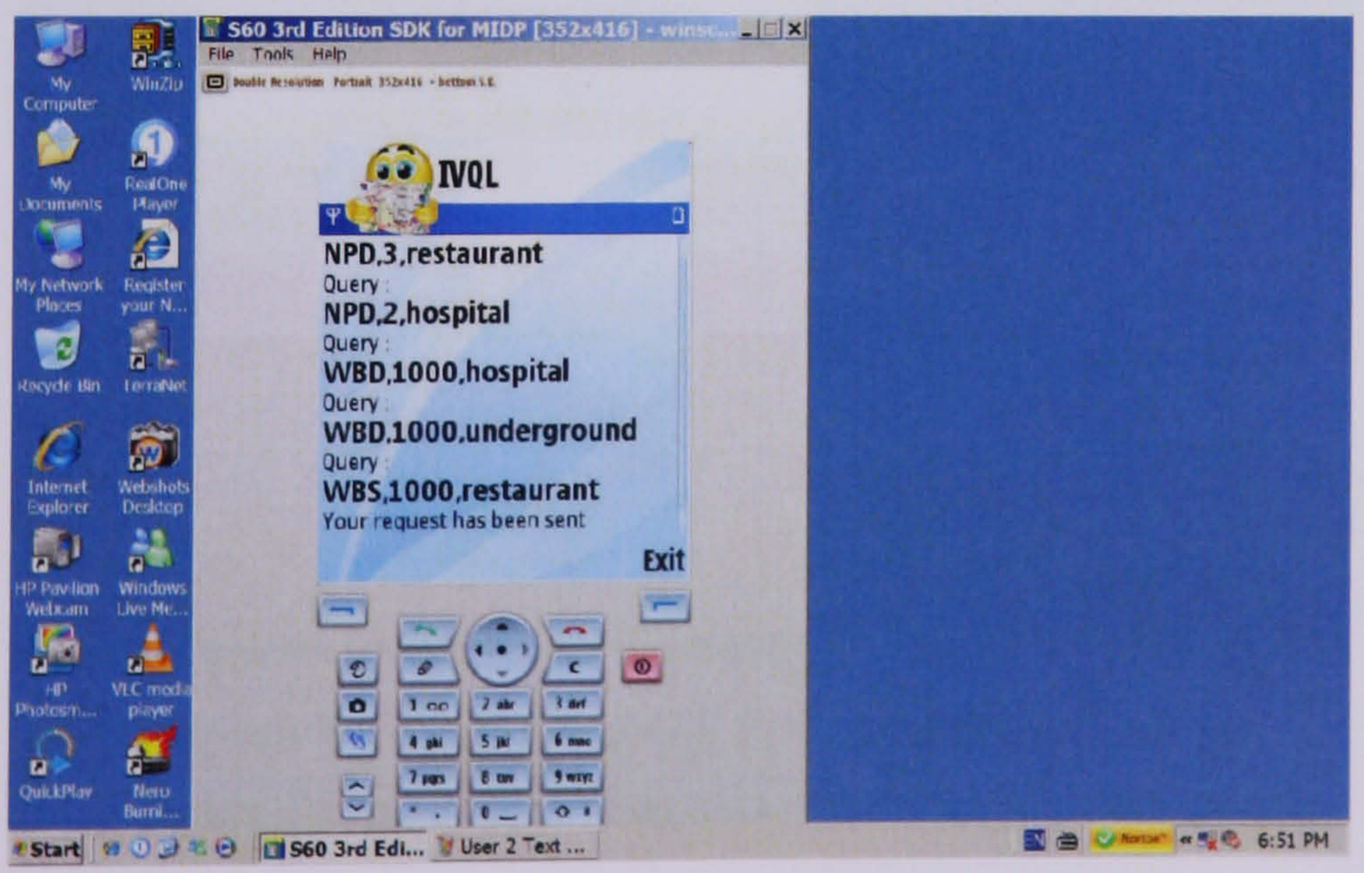


Figure 7.15(e): The SEND Command of User 2.

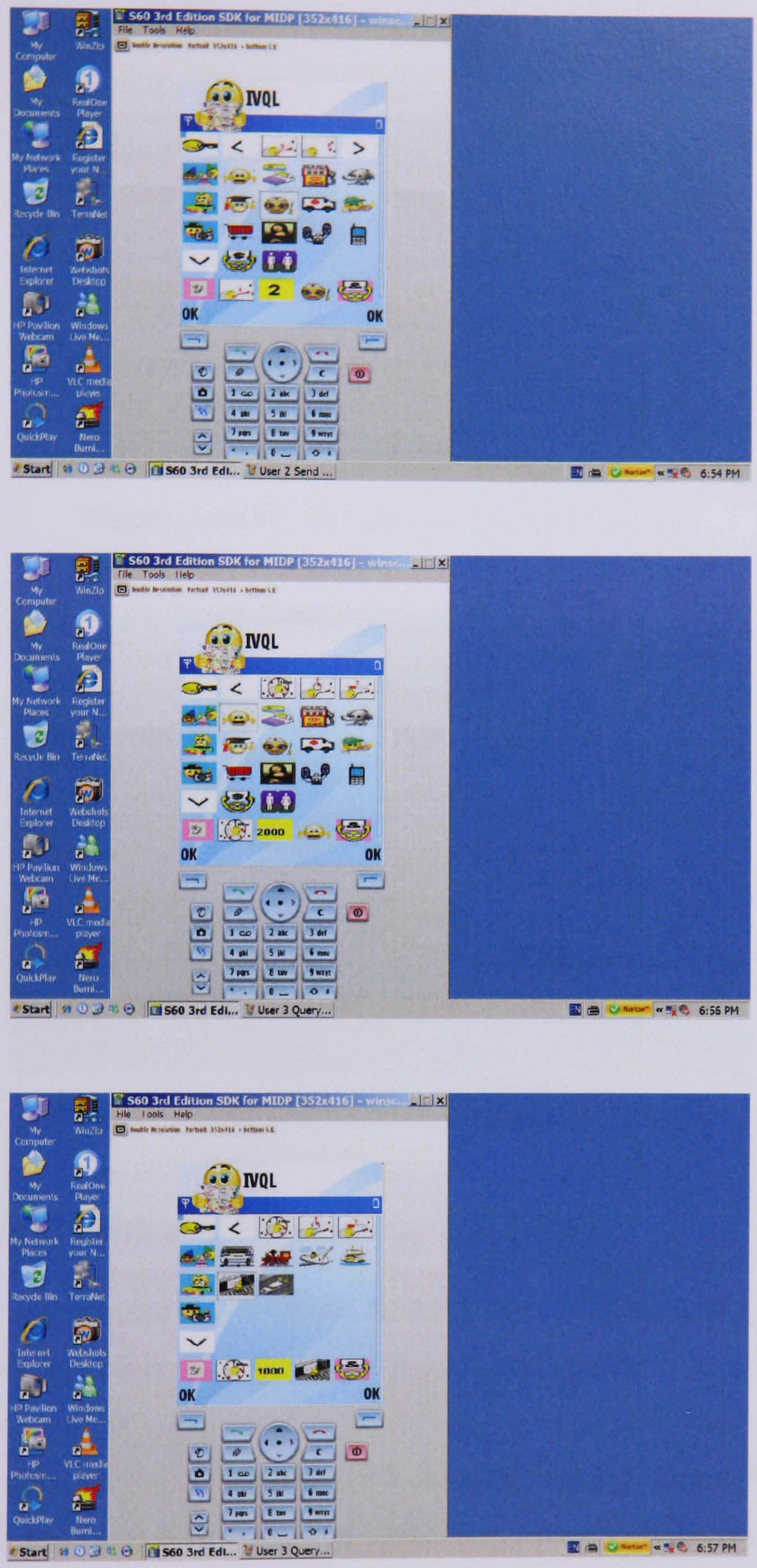


Figure 7.16(a): Queries 1, 2, and 3 of User 3.

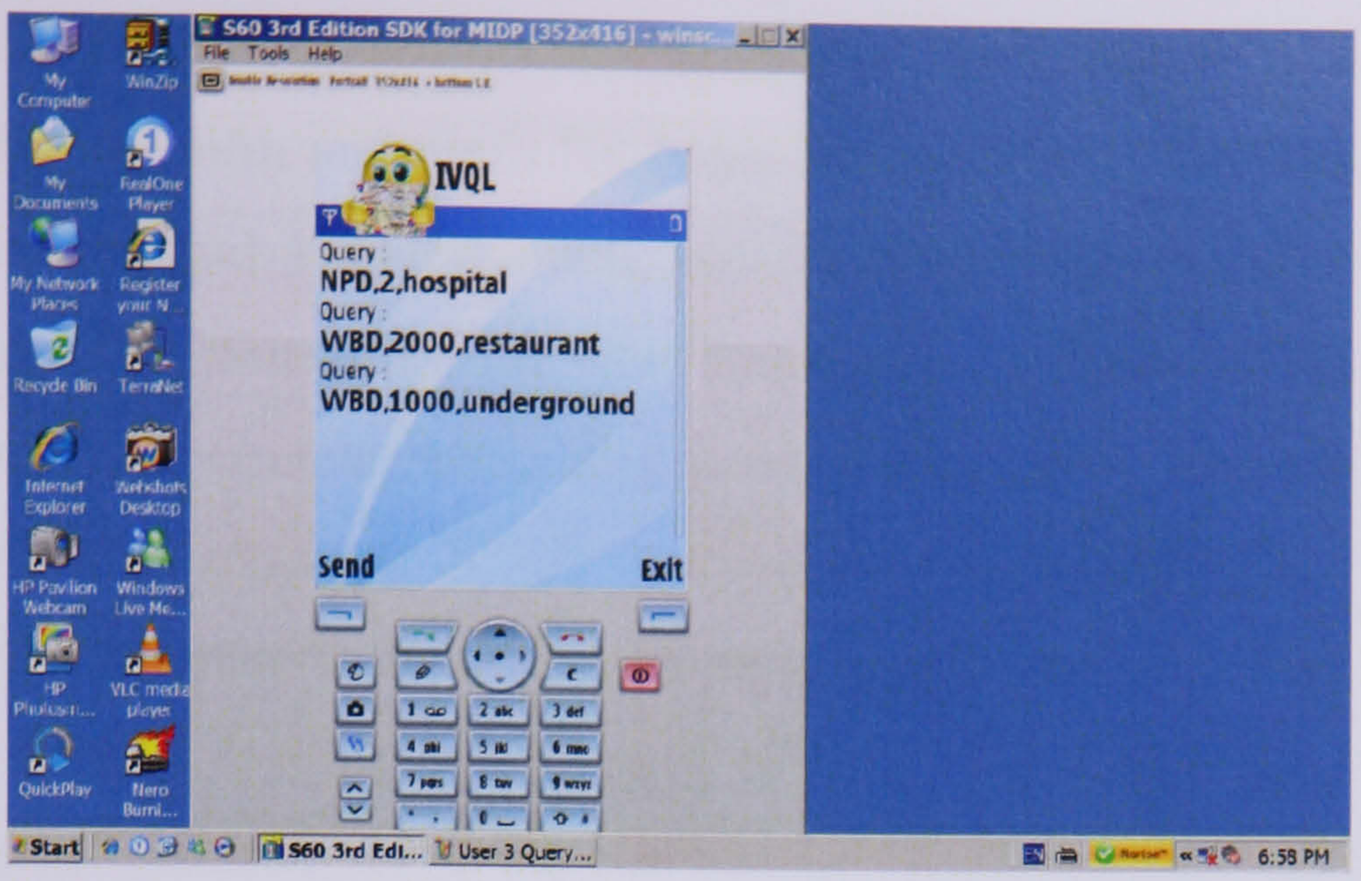


Figure 7.16(b): The Text Queries of User 3.

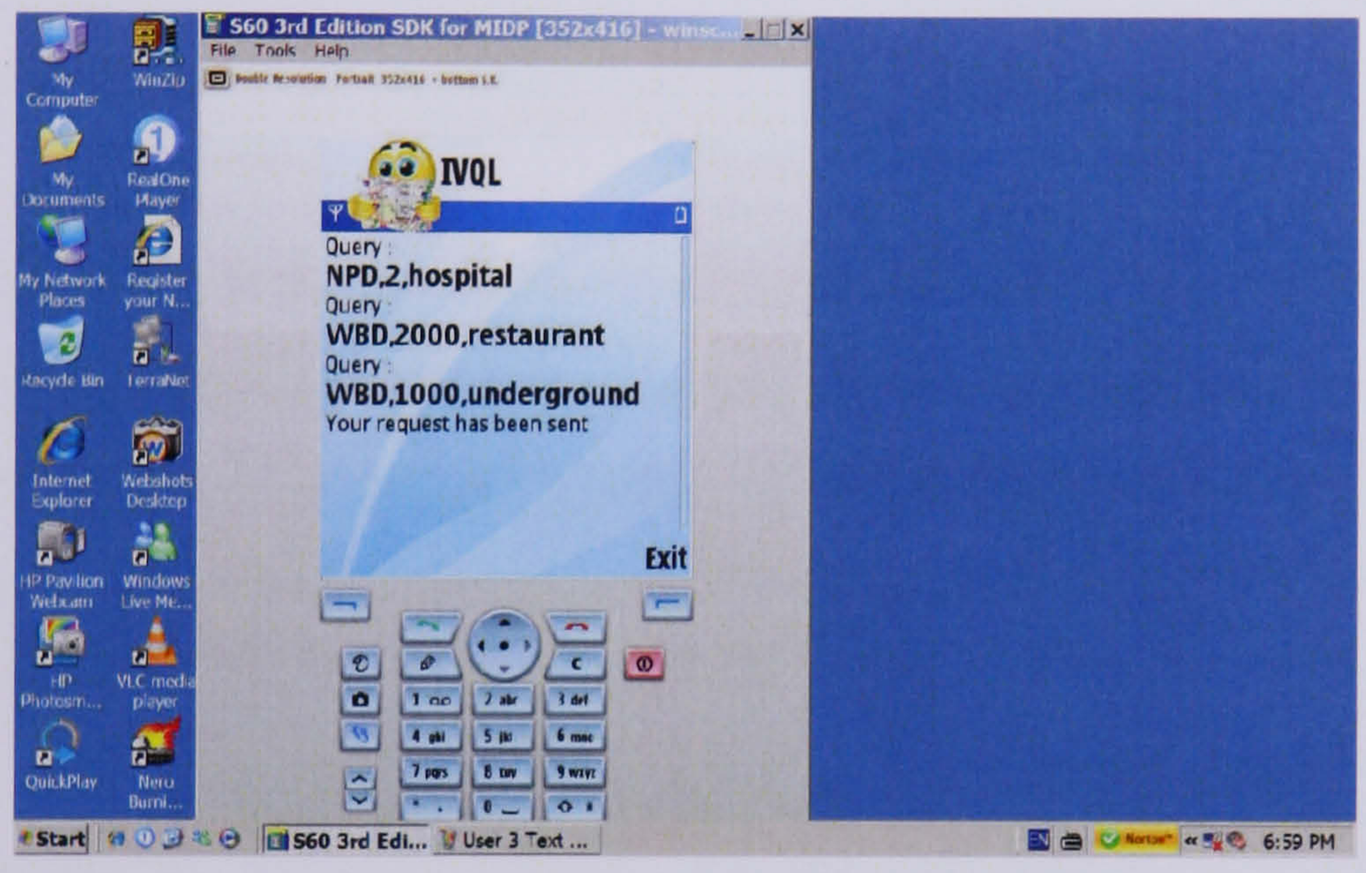


Figure 7.16(c): The SEND Command of User 3.

7.3.2 The QMP With and Without the DECIDER

Once a text complex query is received in the folder of queries, a special listener launches the pre-processor that reads it and parses it into multiple simple queries. The attributes of each simple query are the user number, X location, Y location, date, time, speed, the operator, the value, and the object. The time cost optimizer (TCOP) checks if the scenario has been previously generated. If it has, TCOP routes the complex query to the previously generated plan instead of generating a new one. Otherwise it proceeds with melting the templates.

The first execution of the QMP is done with NO DECIDER where the queries are melted for each one of the tree users. Figure 7.17 shows the QMP interface running before melting the queries of user 1. Each list-box contains one operator's template in

the first column and the simple queries in the rest. Since user 1 uses the operators NPD and WBD, the fifth and ninth list-boxes are occupied. The fifth contains both queries 4 restaurant and 4 underground, and the ninth 2000 underground and 2000 hospital. Figure 7.18 shows the same after melting the queries. The first six functions of the template of the operator NPD have been eliminated because they already exist in the template of the operator ATLD that occupies the first list-box. The functions NF=4 and IMP=M appear in the first query (4 restaurant) but not in the second one (4 underground) because it is enough to apply it to the first, use it for the second, and save the time of re-executing it for the second. The function FAC which means AddFacility is appended with the object restaurant in the first query and the object underground in the second because the objects are different. Otherwise, the same object would have been read once and used in both queries.

Figure 7.17 shows the template of the WBD operator before melting in the ninth list-box. After melting, Figure 7.18 shows that the functions of rows 1, 2, 4, 12, and 13 have been melted. The function DB which means DrawBuffer contains the value 2000 in the first query only (2000 underground) however it has been melted in the second query (2000 hospital) due to the fact that one buffer can be drawn and used to find undergrounds then hospitals instead of drawing a buffer for finding undergrounds then another buffer for finding hospitals. The functions that remain in the list-boxes are included the global execution plan GEP0 and GEP1…n, as shown in the two right most list-boxes, one user after the other. The same is done for the queries of user 2 in the ninth list-box the buffer 1000 is drawn once and used to find the objects hospital and underground as shown in Figures 7.19 and 7.20. However, the functions that appear in the tenth list-box which correspond to the query WBS 1000 restaurants, are included in the Plan0 but not in the Plan 1…n because it is a static query which means that user 2 wants the result at this current time instance only as shown in the two right most list-boxes. The same also is done for the queries of user 3 as shown in Figures 7.21 and 7.22.

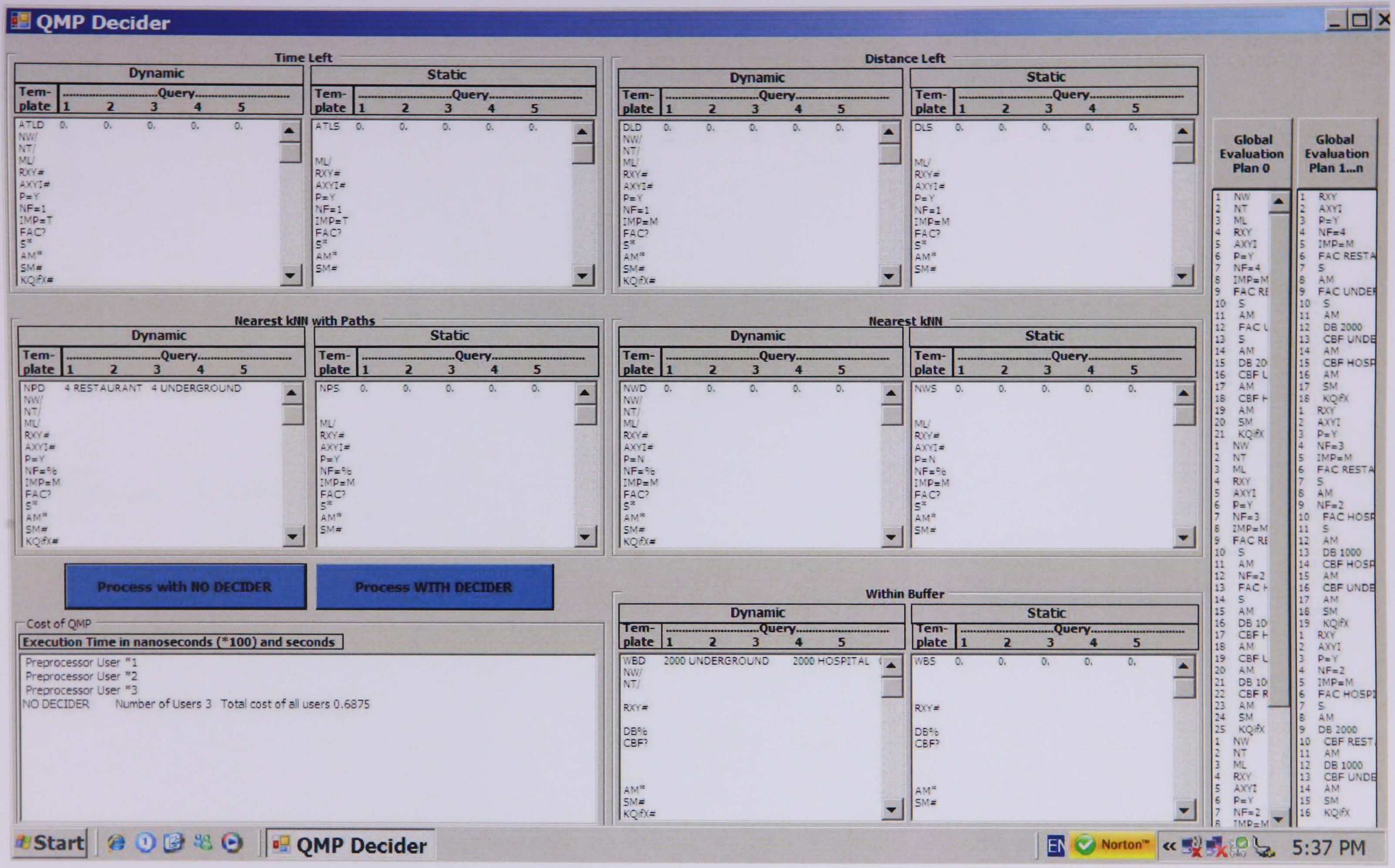


Figure 7.17: The Query Melting Processor with NO DECIDER before Processing User 1 Queries.

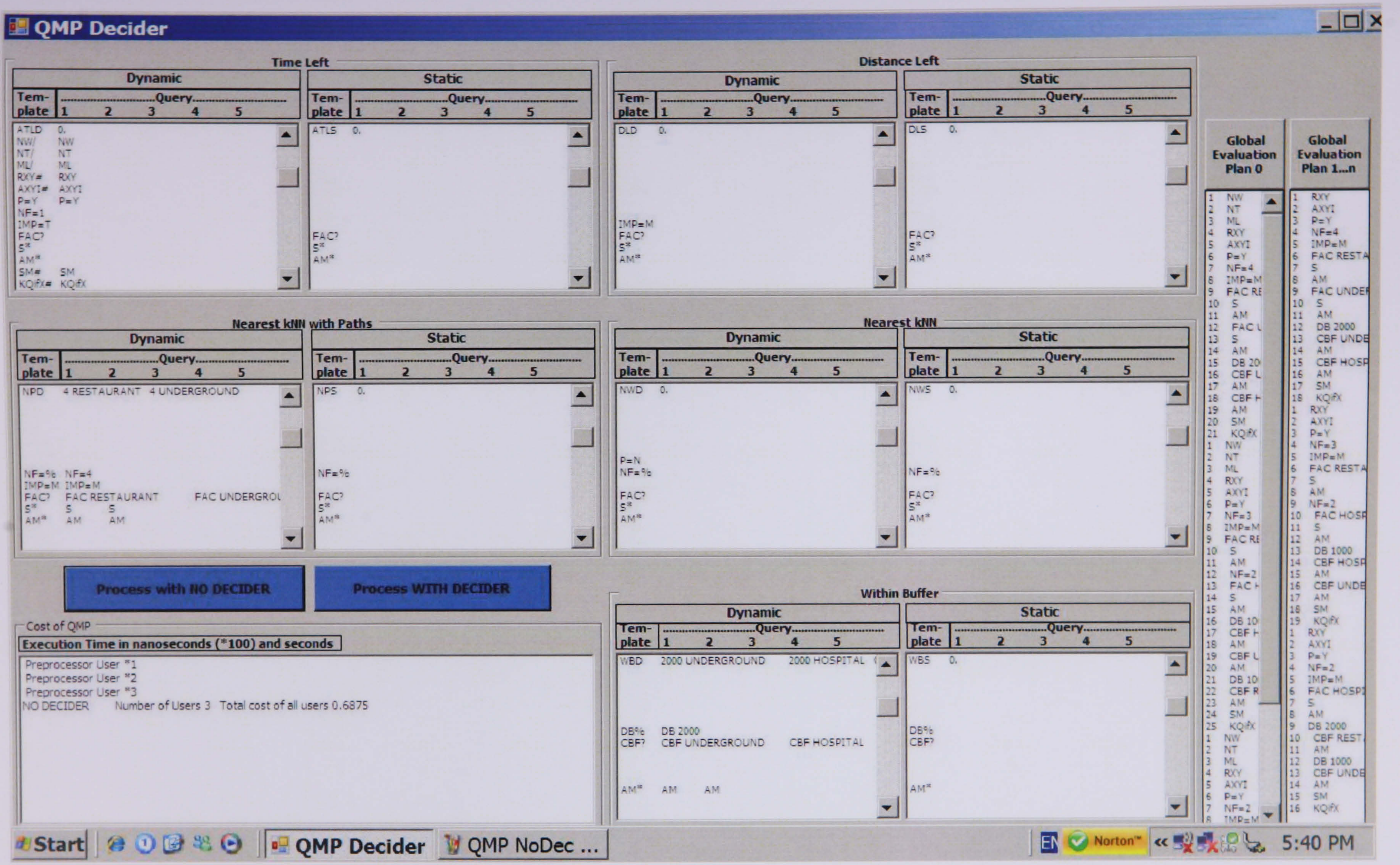


Figure 7.18: The Query Melting Processor with NO DECIDER after Processing User 1 Queries.

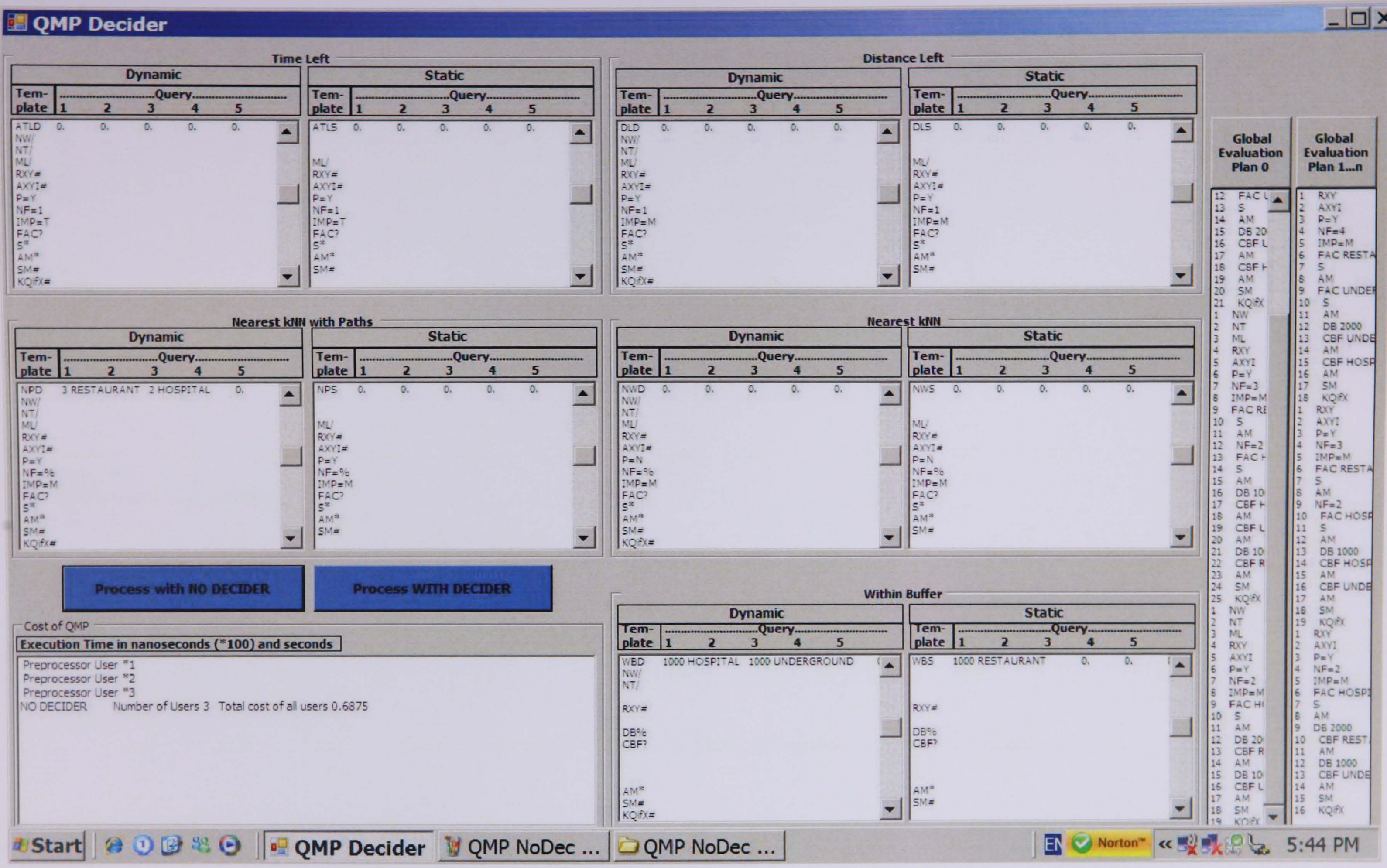


Figure 7.19: The Query Melting Processor with NO DECIDER before Processing User 2 Queries.

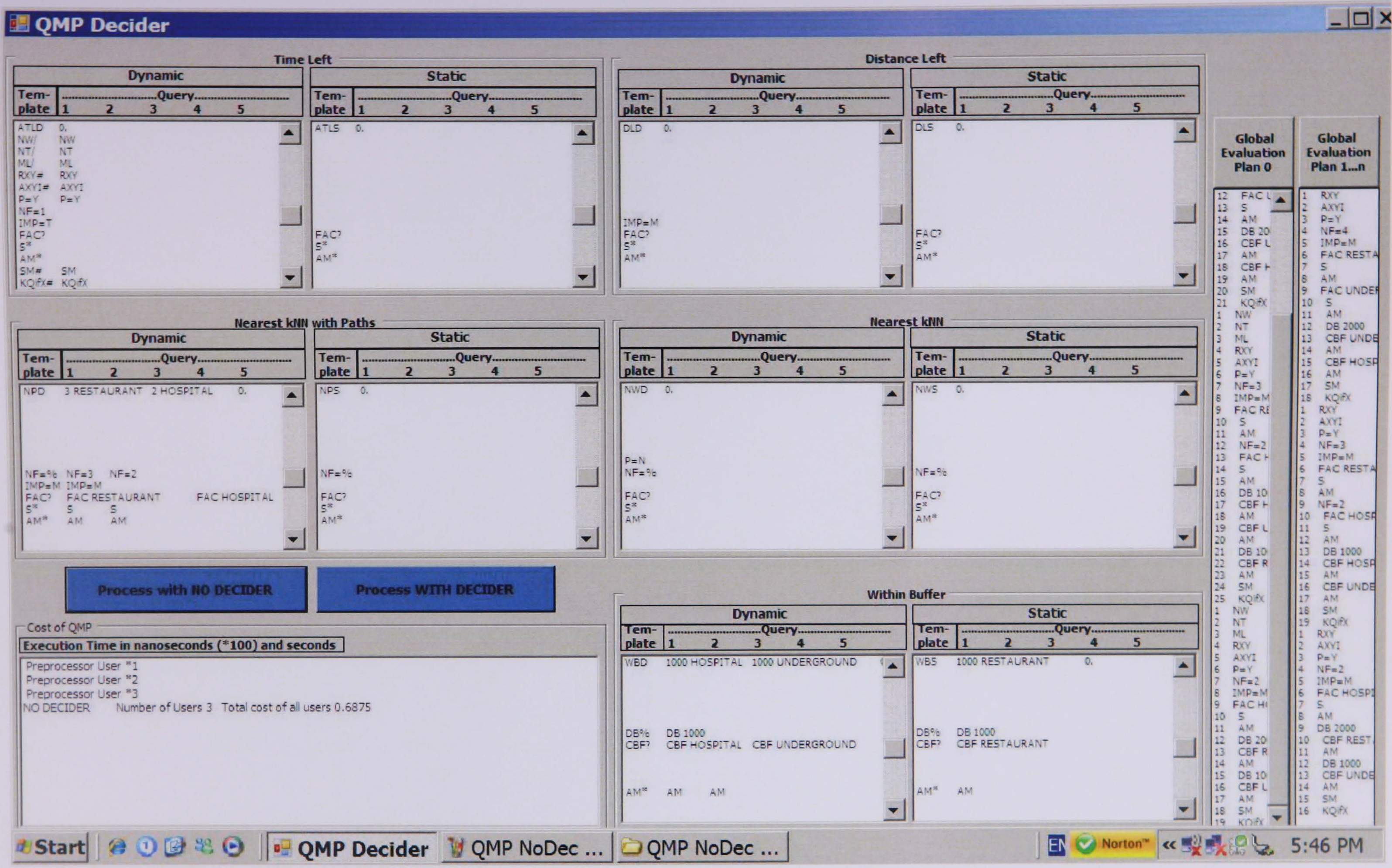


Figure 7.20: The Query Melting Processor with NO DECIDER after Processing User 2 Queries.

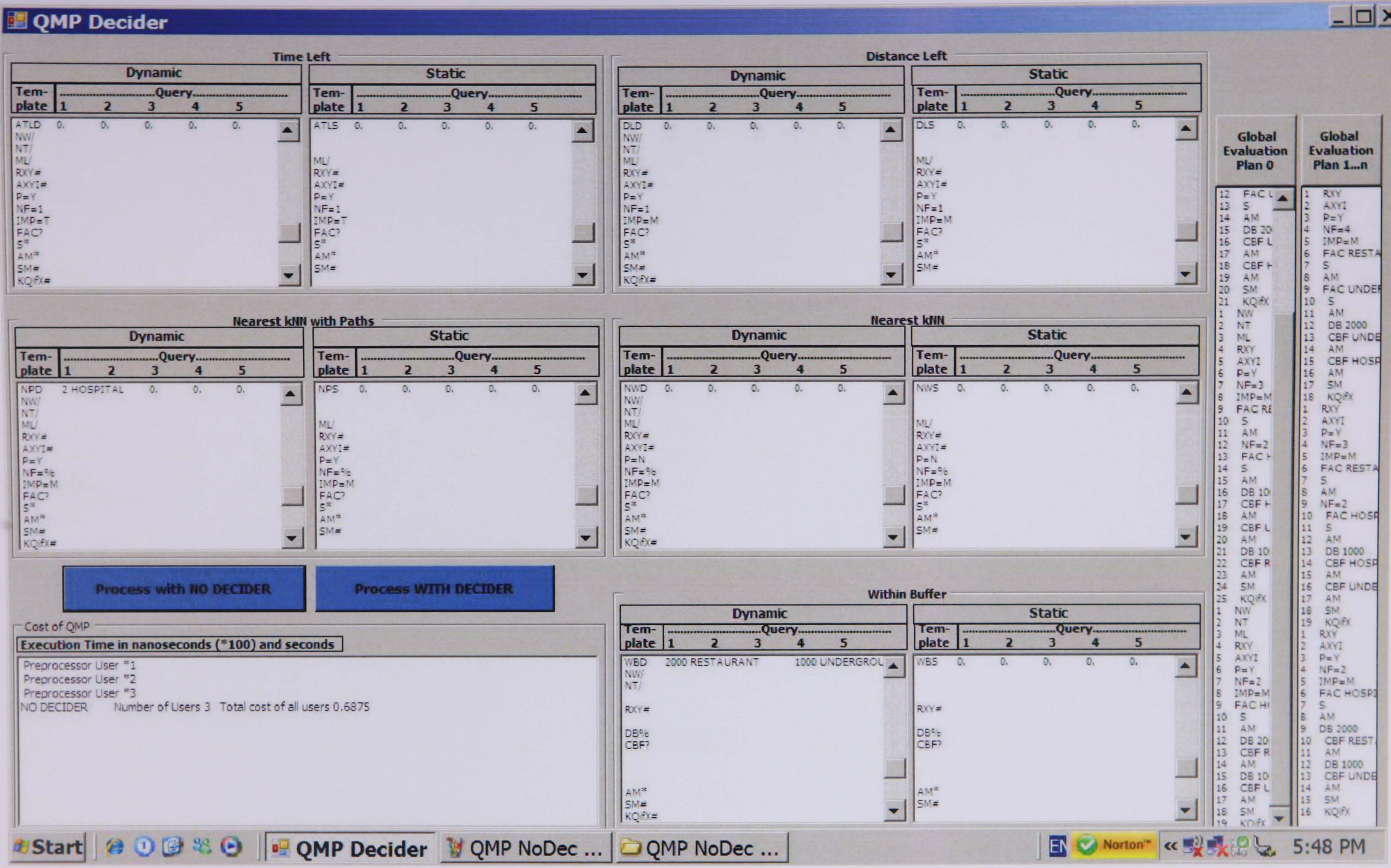


Figure 7.21: The Query Melting Processor with NO DECIDER before Processing User 3 Queries.

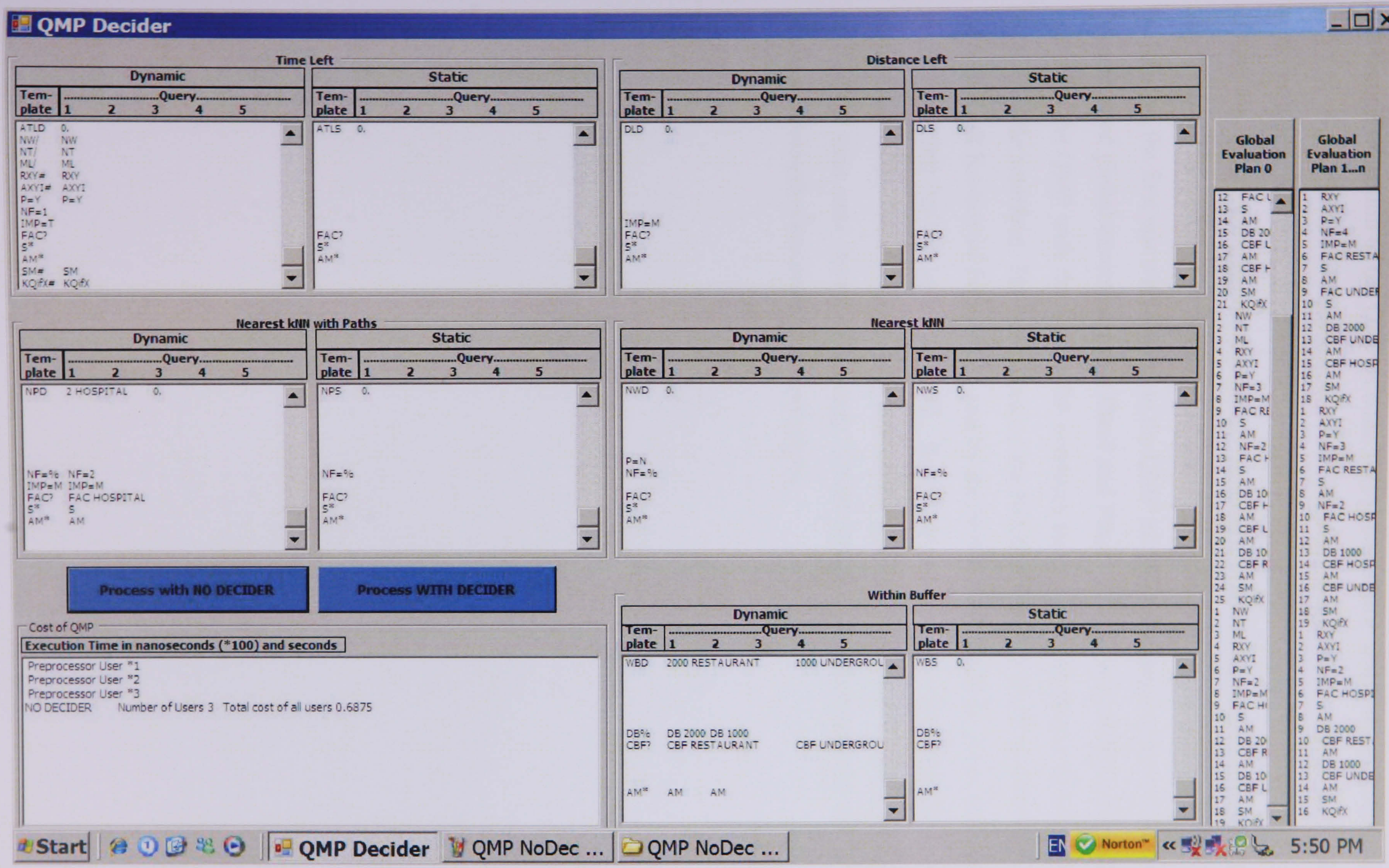


Figure 7.22: The Query Melting Processor with NO DECIDER after Processing User 3 Queries.

The second execution of the QMP is done WITH DECIDER where the queries of the first two users are melted similarly to QMP with NO DECIDER because they have each one a new scenario. Whereas, since the queries of user 3 have the same scenario as user 1's, the TCOP uses the previously generated plan of the melted templates of user 1 instead of melting the templates of user 3. The result of the second execution is the same as the first one with respect to the QMP interface, elements in list-boxes, and the generated global execution plans Plan0 and Plan1…n. The only difference is that executing the QMP with the DECIDER produces faster results as shown in lower left list-box of the interface. The time taken by the NO DECIDER is 68.75 ns as shown in the lower left list-box of Figure 7.22 and by the WITH DECIDER 31.25 ns as shown in the lower left list-box of Figure 7.23. It is clear that the time saved is significant, however, an experimental study is carried on in section 7.4 in order to prove that the new TCOP time cost optimizer strategy is cost effective by implementing the new Sharing Previously Generated Plans paradigm.

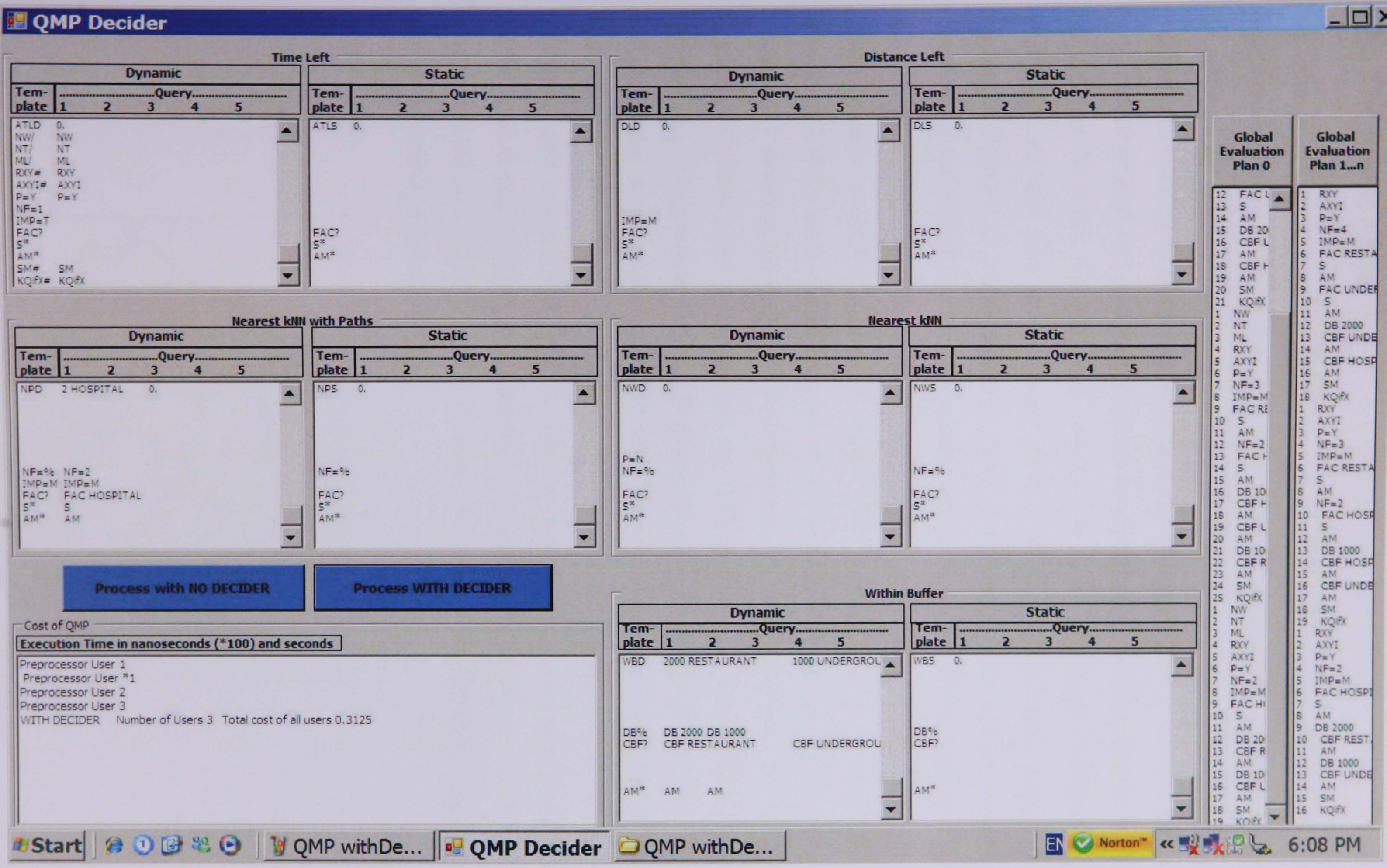


Figure 7.23: The Query Melting Processor WITH DECIDER after Processing User 3 Queries.

7.3.3 The Result Maps

The map of Paris is used to show the results of the queries. It contains the underground stations (Metro) shown in blue, some of the restaurants shown in red, and some of the hospitals shown in black as shown in Figure 7.24. The map can be zoomed in to see a more detailed map as shown in Figures 7.25 and 7.26, zoomed out to see a more general one, and panned which means that the user can move the map in any direction he wants. For each user two X Y locations have been considered, the result maps of each user show the map at each of his locations both normal and zoomed in. Figure 7.27 shows the result of user1 at location 1 showing the nearest 4 restaurants in red with their paths, the nearest 4 undergrounds with their paths in blue, and the 200 m buffer in pink including the hospitals and undergrounds. Figure 7.28 shows the same results but zoomed in. Figure 7.29 shows the results at his location 2 and Figure 7.30 the same results but zoomed in. Figure 7.31 shows the result of user 2 at location 1 showing the nearest 3 restaurants in red with their paths, the nearest 2 hospitals with their paths in black, and the 1000 m buffer in mauve including the hospitals and undergrounds. Figure 7.32 shows the same results but zoomed in. Figure 7.33 shows the results at his location 2 and Figure 7.34 the same results but zoomed in. Figure 7.35 shows the result of user 3 at location 1 showing the nearest 2 hospitals with their paths in black, the 2000 m buffer in light green including the restaurants and the 1000 m buffer in light pink including the undergrounds. Figure 7.36 shows the same results but zoomed in. Figure 7.37 shows the results at his location 2 and Figure 7.38 the same results but zoomed in.

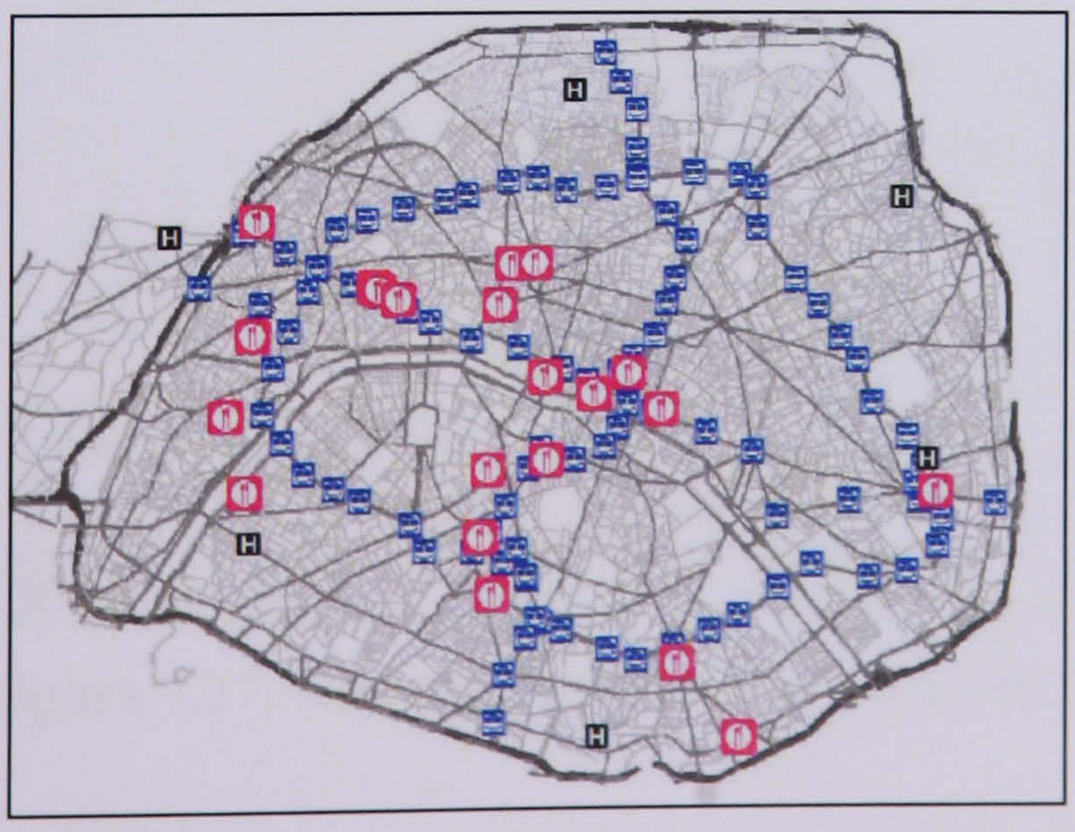


Figure 7.24: The Map of Paris showing Hospitals, Restaurants, and Underground.

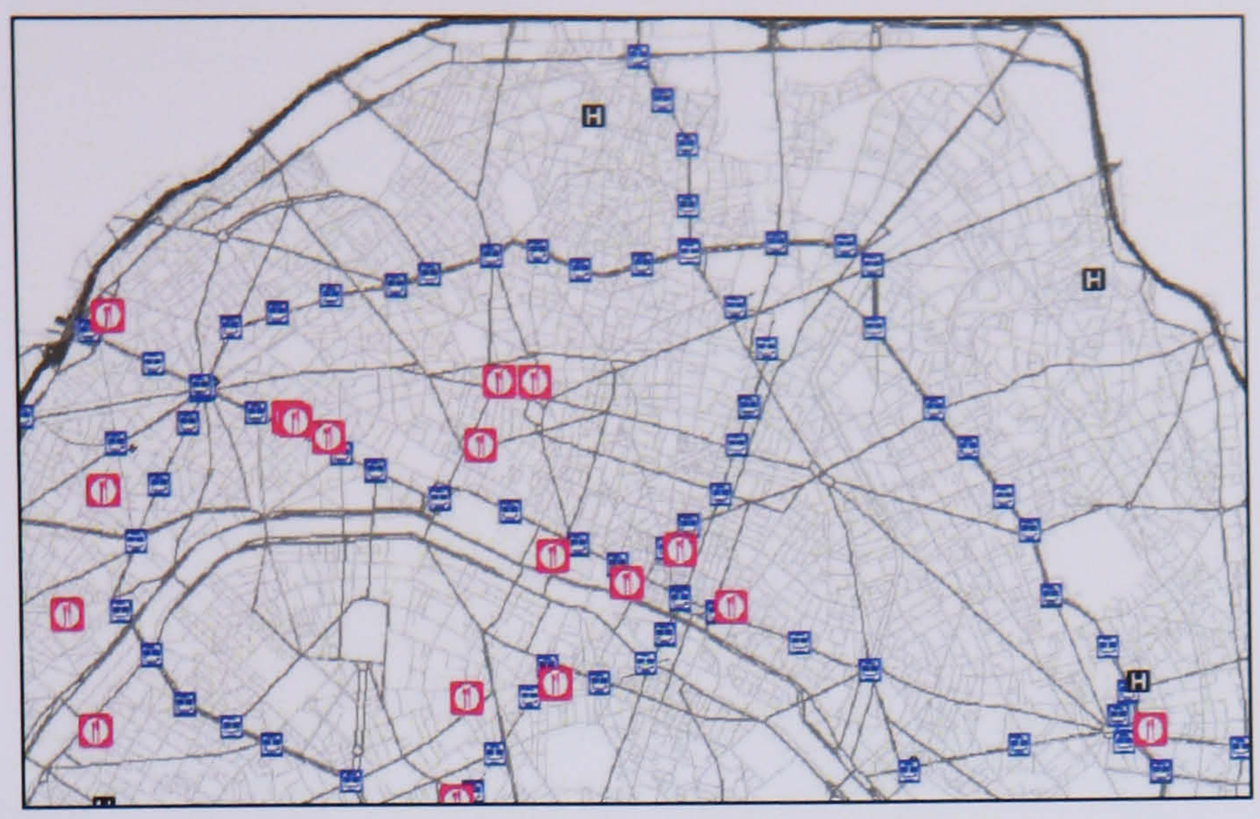


Figure 7.25: The Map of Paris Zoomed In.

Figure 7.26: The Map of Paris Zoomed In More.

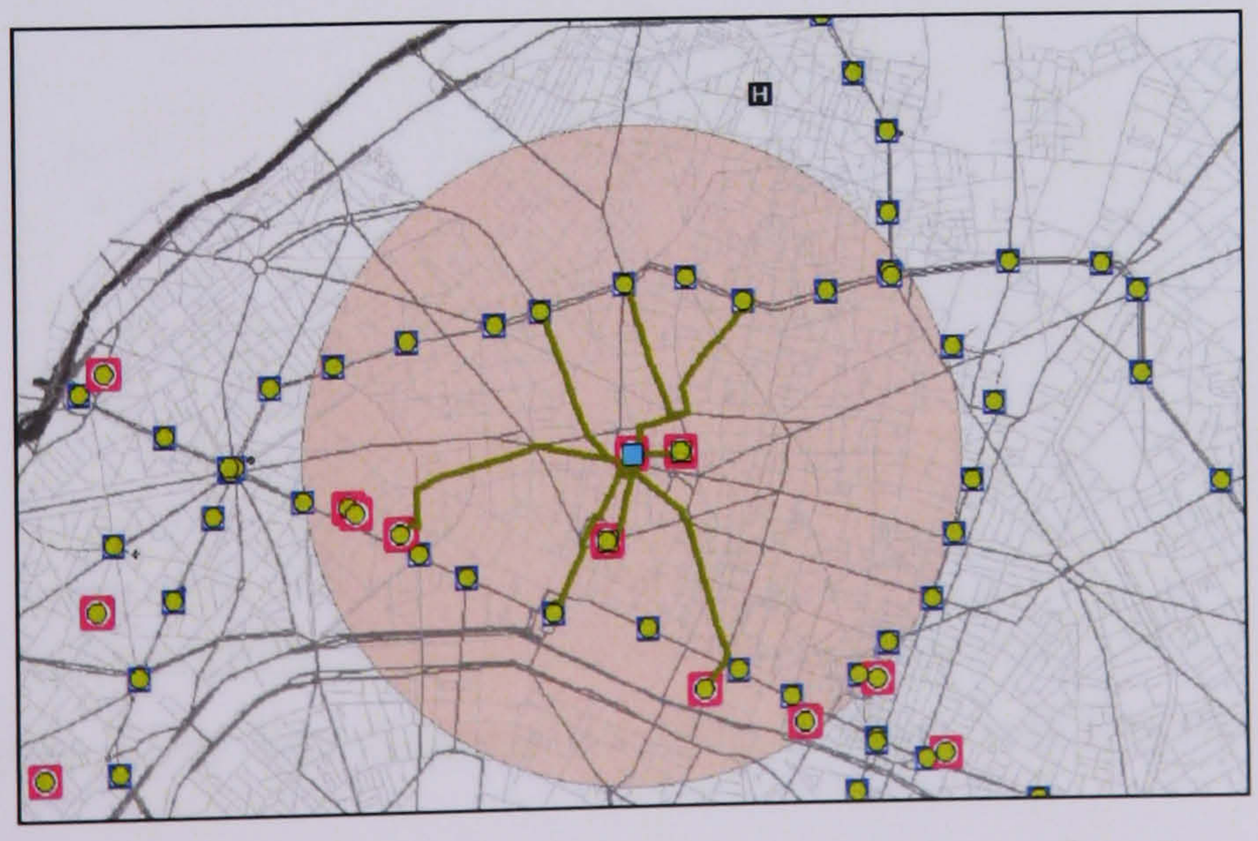


Figure 7.27: The Map of User1 at Location 1.

Figure 7.28: The Zoomed In Map of User 1 at Location 1.

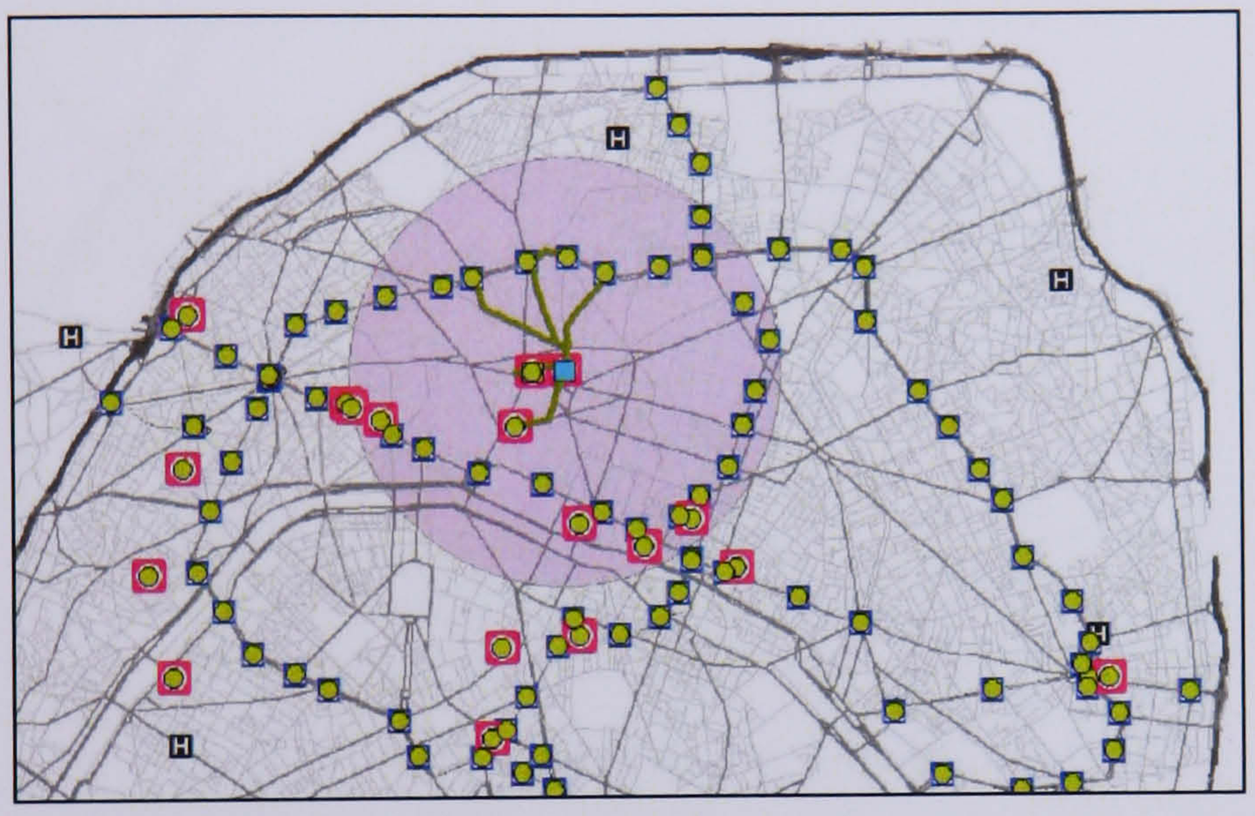


Figure 7.29: The Map of User 1 at Location 2.

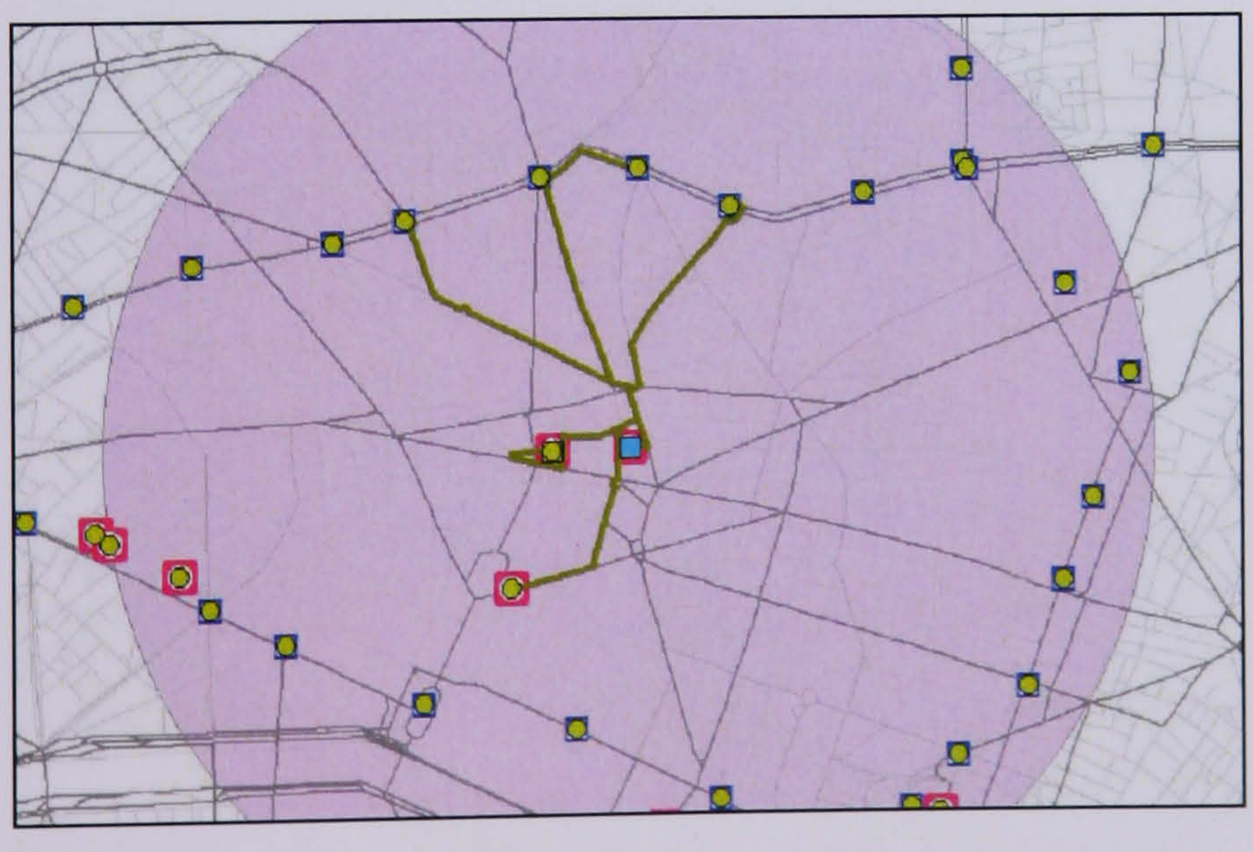


Figure 7.30: The Zoomed In Map of User 1 at Location 2.

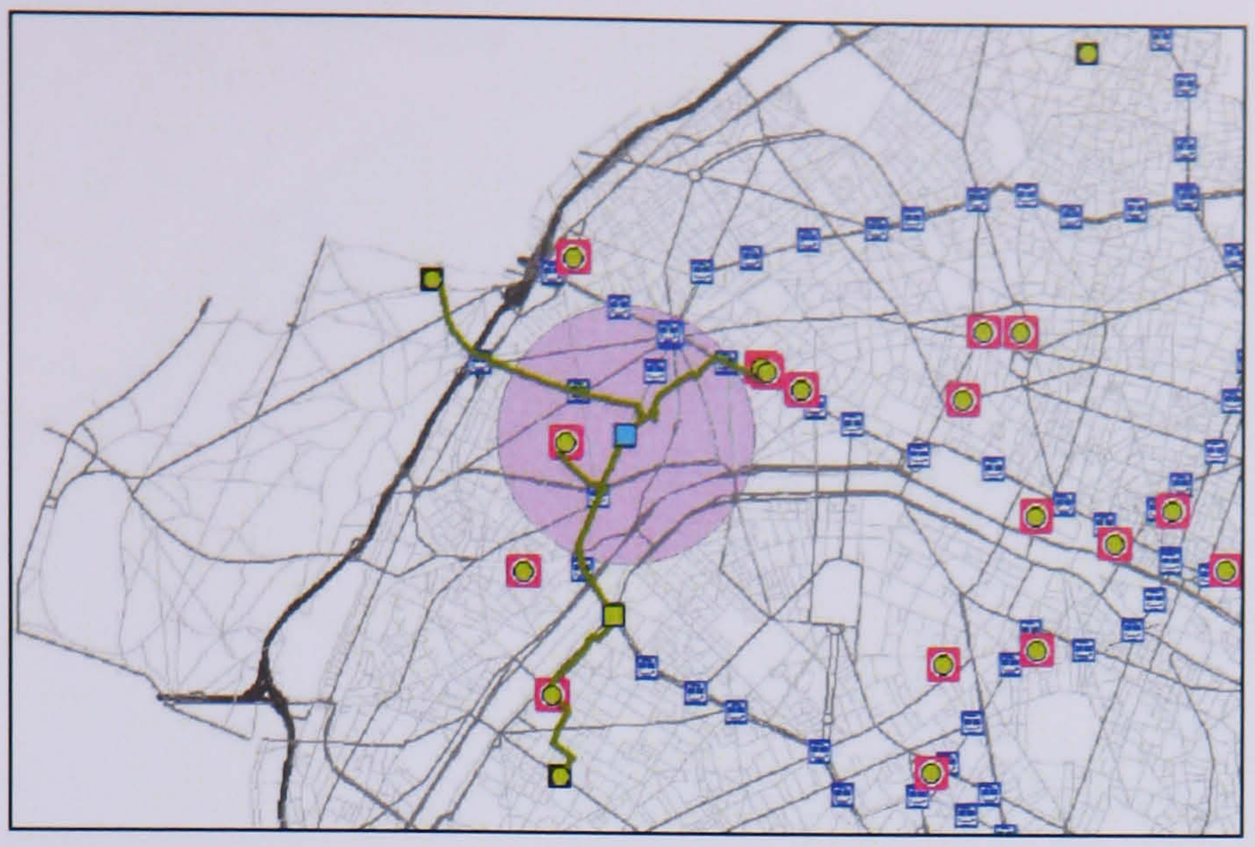


Figure 7.31: The Map of User 2 at Location 1.

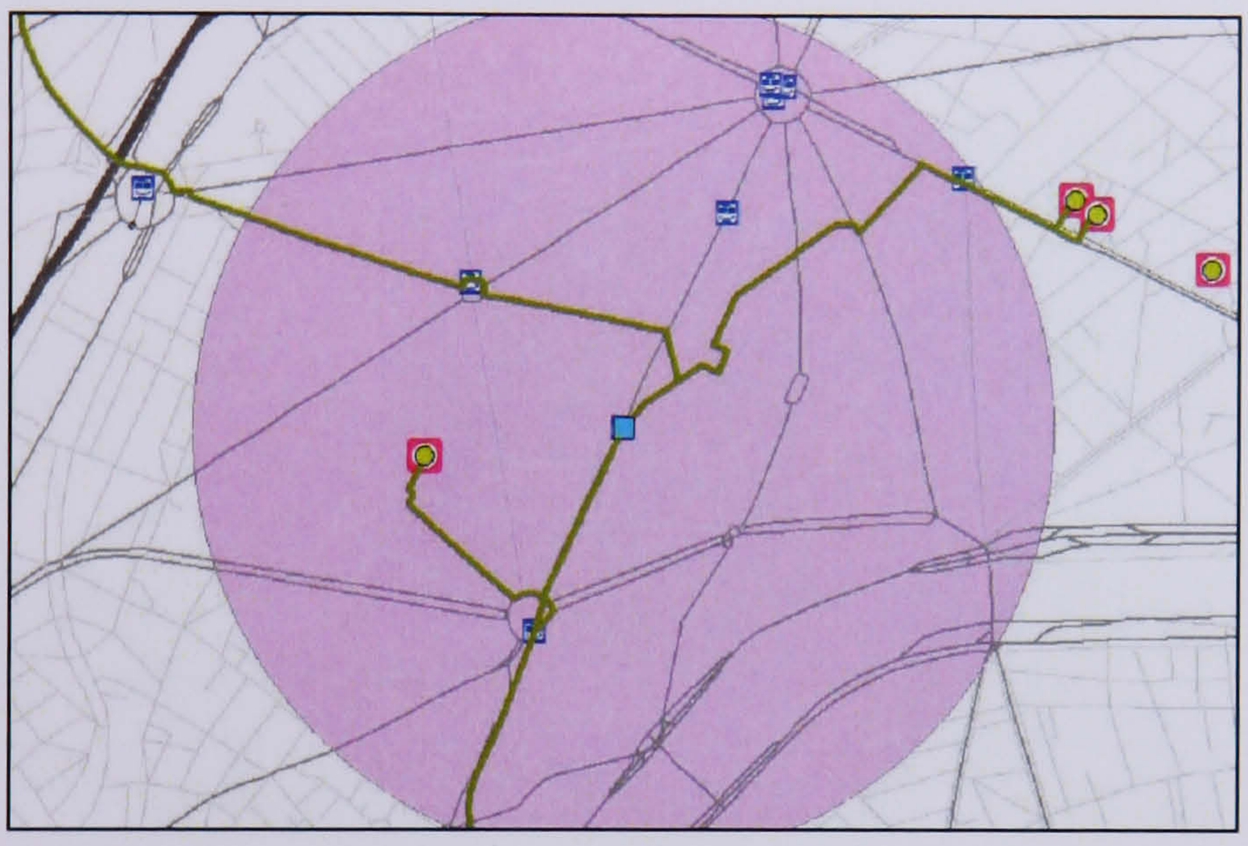


Figure 7.32: The Zoomed In Map of User 2 at Location 1.

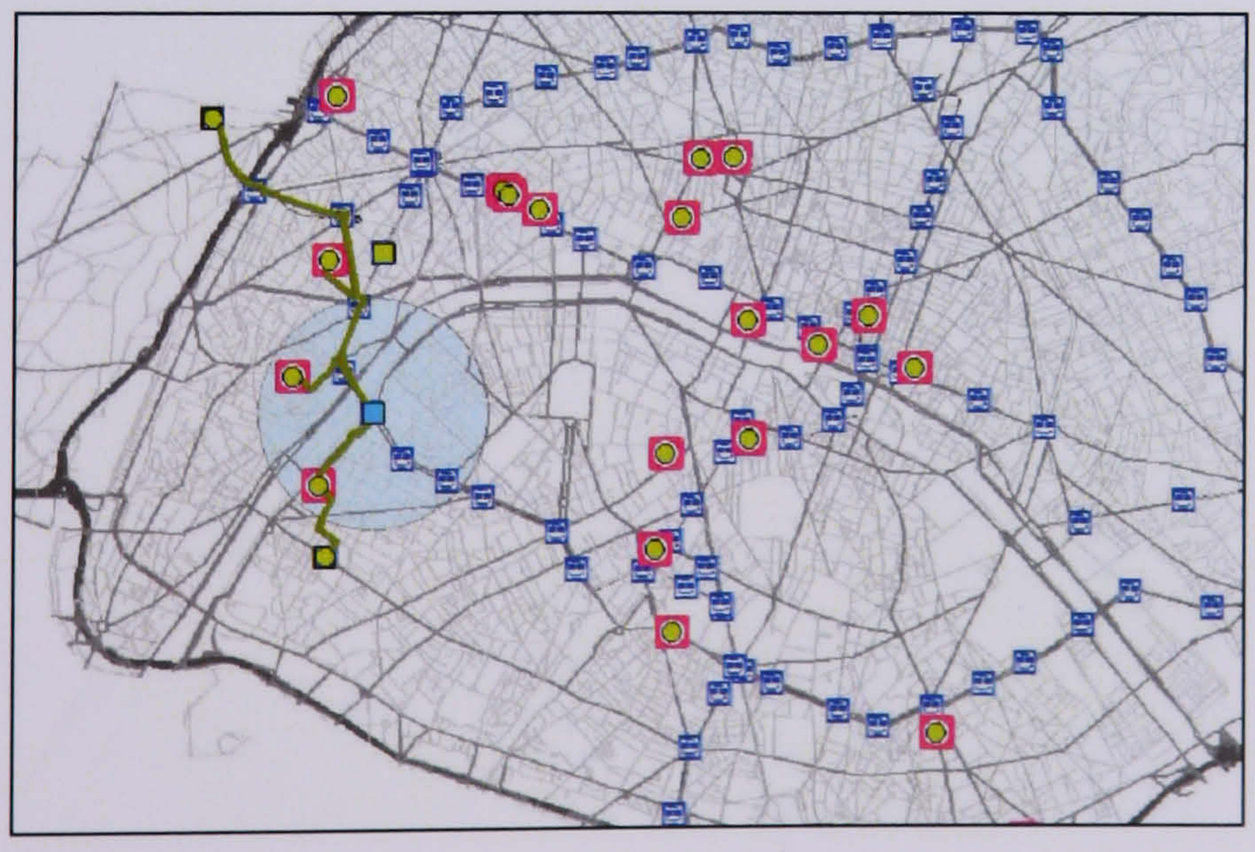


Figure 7.33: The Map of User 2 at Location 2.

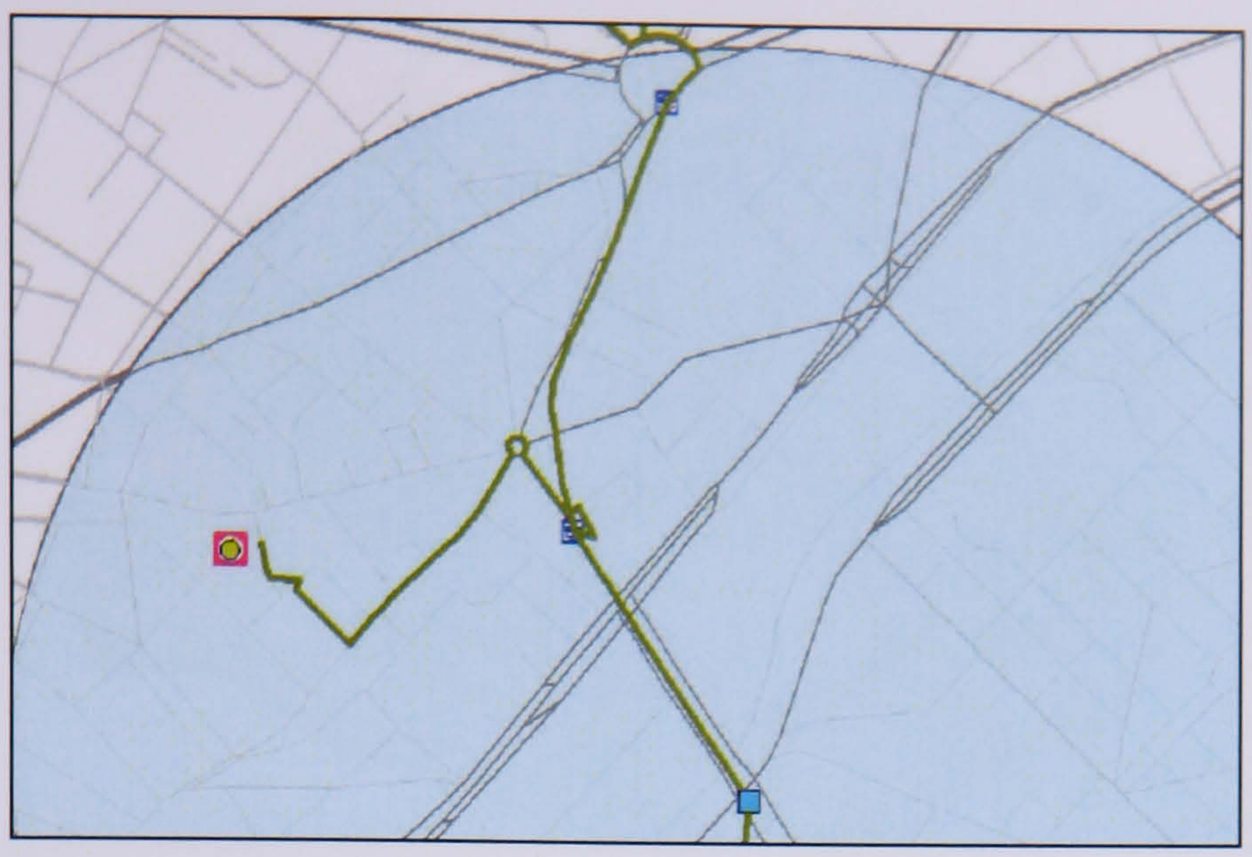


Figure 7.34: The Zoomed In Map of User 2 at Location 2.

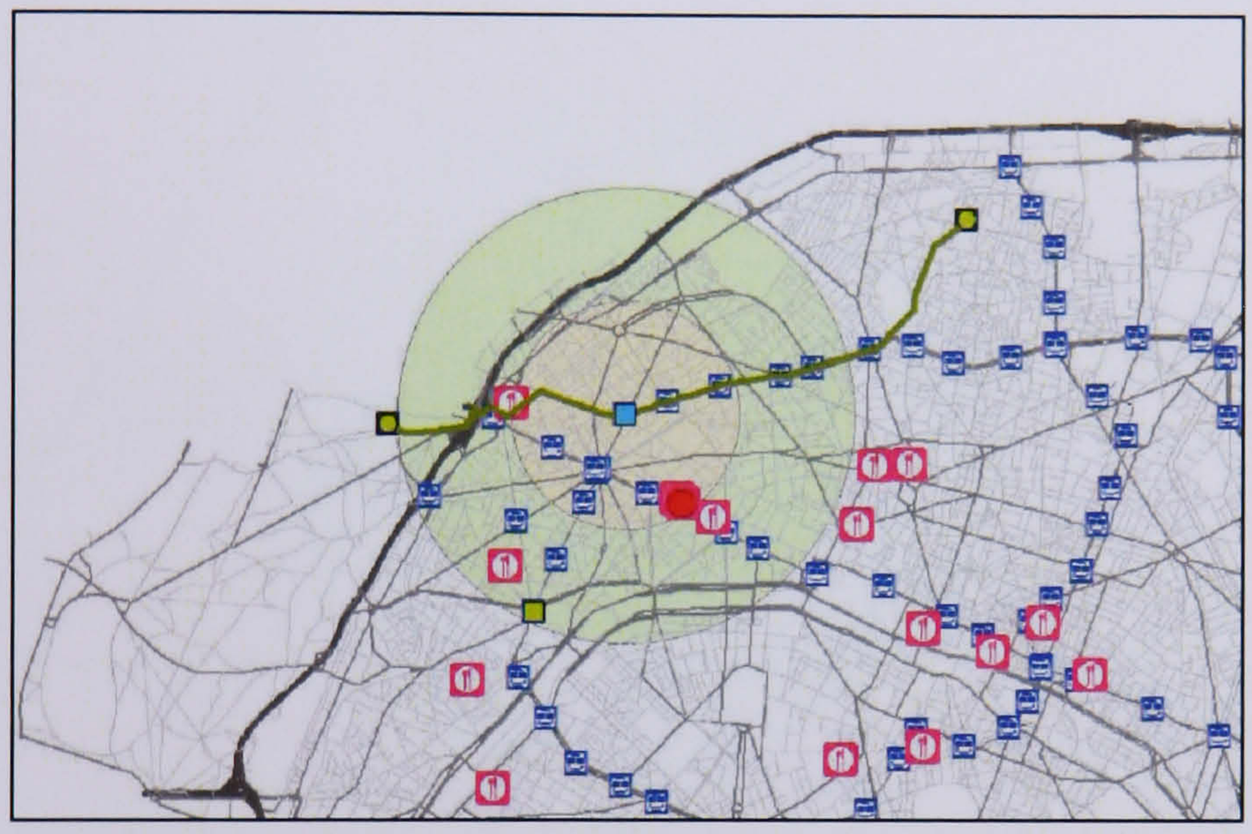


Figure 7.35: The Map of User 3 at Location 1.

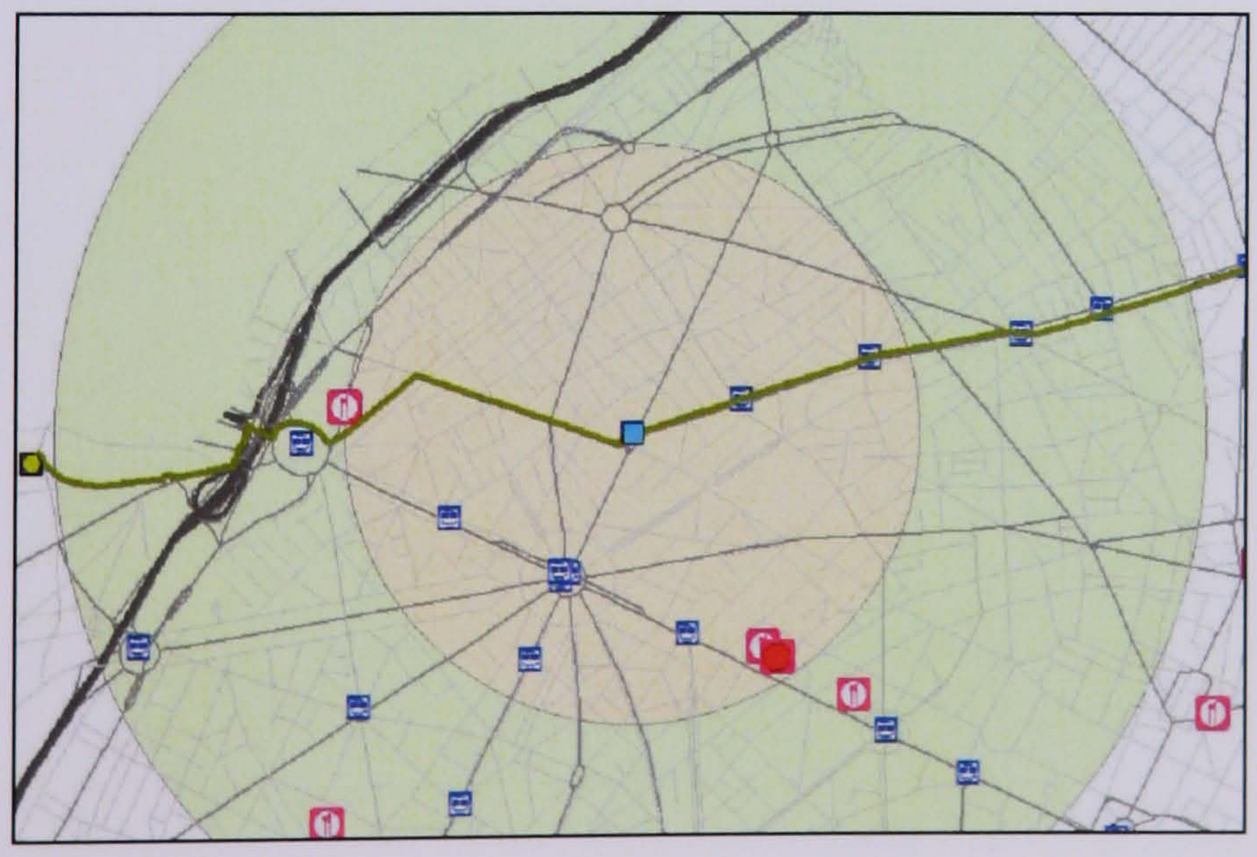


Figure 7.36: The Zoomed In Map of User 3 at Location 1.

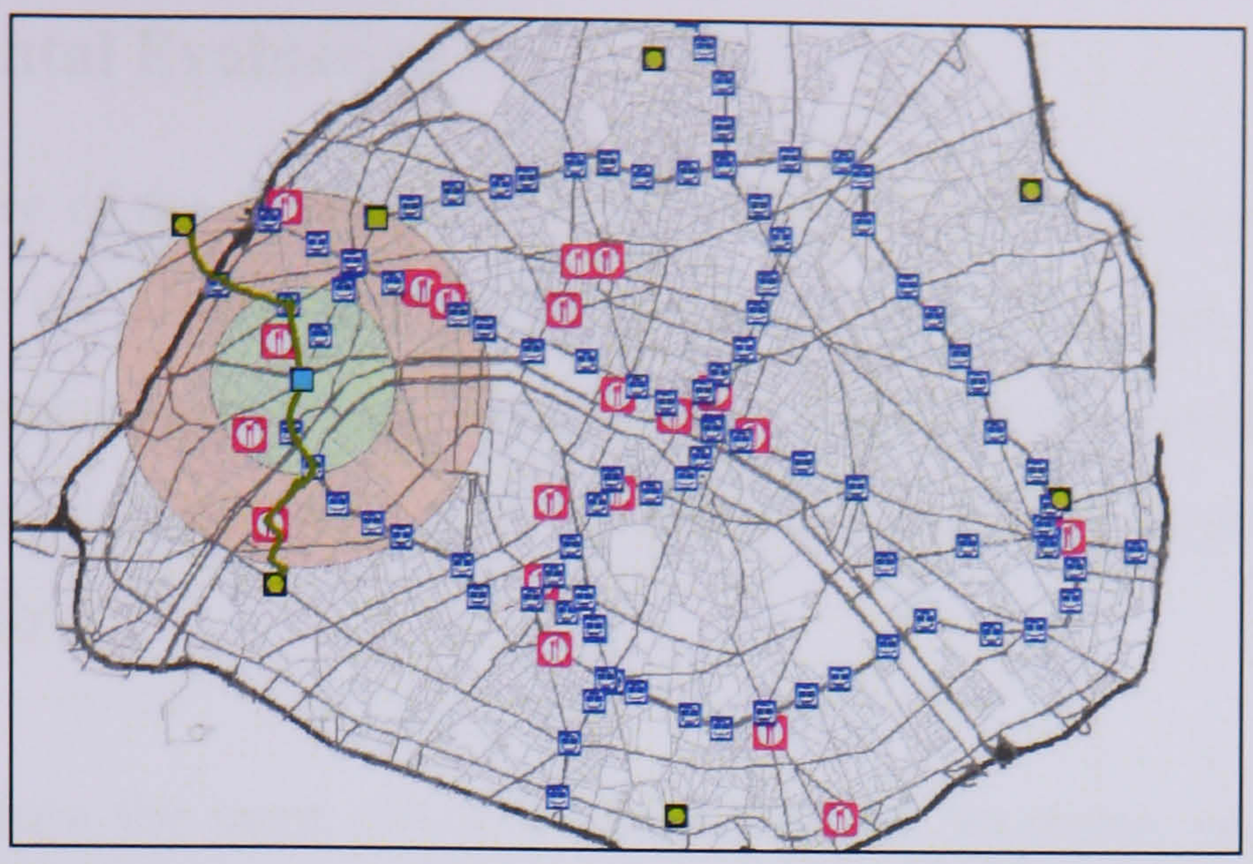


Figure 7.37: The Map of User 3 at Location 2.

Figure 7.38: The Zoomed In Map of User 3 at Location 2.

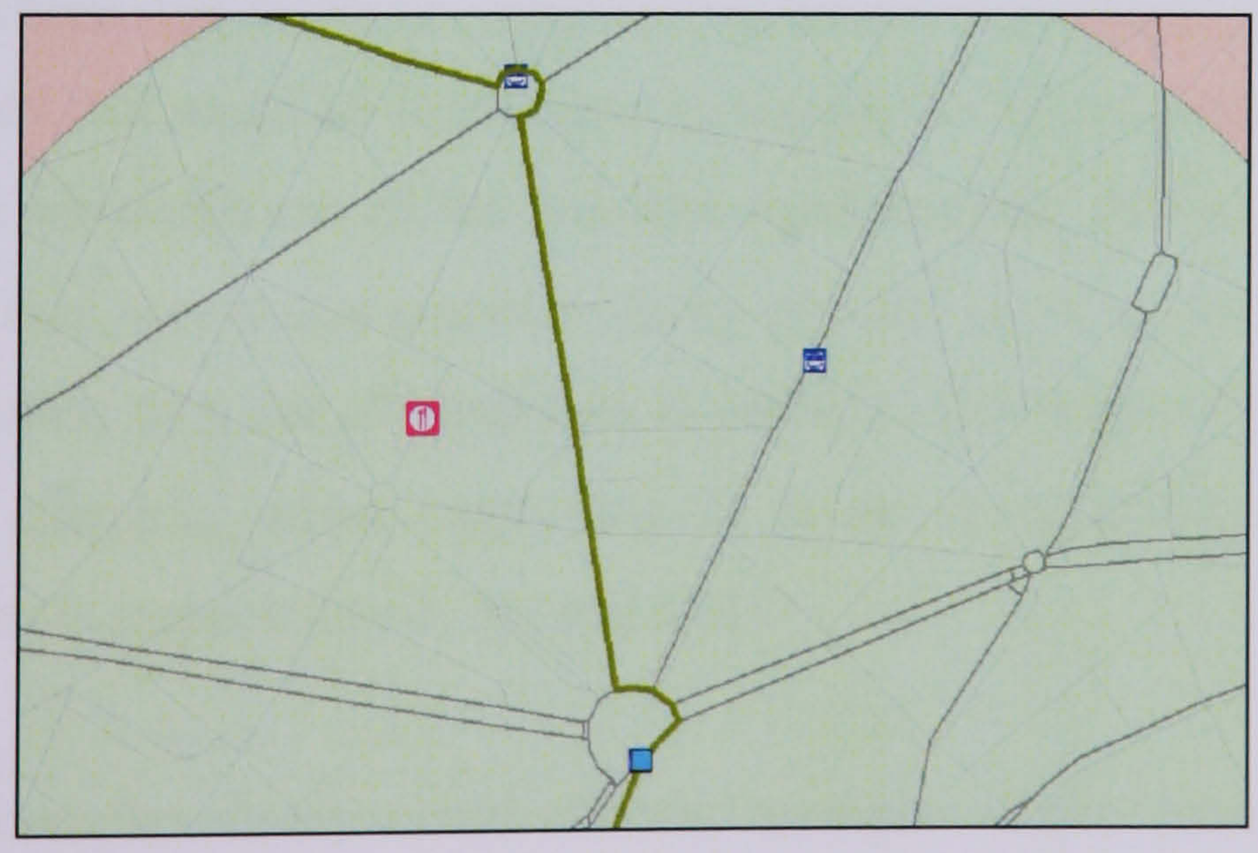


Figure 7.39: The More Zoomed In Map of User 3 at Location 2.

7.4 Experimental Evaluation

The main purpose of the experimental evaluation is to study the execution time cost effectiveness of the Query Melting Processor (QMP). This can be achieved by comparing the execution time cost of using the Decider to the execution time cost of not using the Decider with the QMP. The results are analysed and conclusions are drawn accordingly.

In order to produce the most efficient results, all the operators, operator templates referred to as templates, and objects that were explained earlier are used to formulate complex queries called data files hereafter. The operators that are implemented are Find the nearest facilities (SN), Find the facilities that are within a buffer (SW), Find the nearest facilities and their shortest path (SNP), Find the time left to reach the nearest facility (STL), Find the distance left to reach the nearest facility (SDL), Find the shortest path (SP), Dynamically Find the nearest facilities (DN), Dynamically Find the facilities that are within a buffer (DW), Dynamically Find the nearest facilities and their shortest path (DNP), Dynamically Find the time left to reach the nearest facility (DTL), Dynamically Find the distance left to reach the nearest facility (DDL), and Dynamically Find the shortest path (DP). Each of the listed operators has a corresponding Execution Plan that is called Operator Template or simply Template. A complex query can be formulated with one or more operators of the set of templates that are implemented. For example, suppose that the set of templates is {SN, SW}, the possible scenarios that could be formulated by a complex query consist of the subsets of the templates set which are all the possible combinations: {SN}, {SW}, and {SN, SW}. It can be said here that a complex query is made up of up to 2 templates. The number of scenarios of a set of templates is equal to 2^t-1 where t is the number of templates. For example, when employing 2, 5, or 10 templates the number of scenarios or plans is respectively 3, 31, and 1023.

Moreover, the complex query could contain multiple simple queries of the same operator. One example of a complex query made up of 3 simple queries using one operator is SN1R (Nearest 1 Restaurant), SN4H (Nearest 4 Hotels), and SN6B (Nearest 6 Bus Stations). Another example of a complex query made up of 6 simple queries using 2 operators is SN1R (Nearest 1 Restaurant), SN4H (Nearest 4 Hotels),

SN6B (Nearest 6 Bus Stations), SW100U (Within 100m Universities), SW50H (Within 50m Hotels, and SW70S (Within 70m Schools). The data files of the experiment are generated varying the number of complex queries respectively from 1, 2, 5, 10, 20, 50, to 100. For each data file, other data files are also generated varying the number of templates respectively from 2, 5, to 10. An example of a data file consists of 100 complex queries each of which is formulated with 7 operators that belong to the set of 10 templates, where each operator is used to formulate 5 simple queries making a total of (7× 5) 35 simple queries per complex query and a total of (100× 7× 5) 3500 simple queries which are processed sequentially. The Query Melting Processor is executed for each data file twice, the first time using the Decider and the second without using it and the time cost of each execution is recorded. The experimental results are reported using a comparison between the time costs of the two methods with the aim to conclude which of them is cost effective. Moreover, in order to insure getting the most accurate results all the tasks and applications of the computer are stopped and the network connections are disabled.

7.4.1 Results and Analysis

Quantitative data are being used in order to produce the necessary statistical results. The 'quality characteristic' execution time cost is quantified and measured by the number of nanoseconds. Three processes costs are taken into consideration. The first one is when the processor is executed without using the Decider which reflects the cost of Melting the Templates. The second and third ones are when the processor is executed using the Decider. The second is the cost of Generating a New Melted Templates Plan and Storing it in the RAM. The third is the cost of Accessing a Previously Generated Melted Templates Plan referred to as Old Plans hereunder. Table 7.1 shows the time cost of processing complex queries (CQ) that are formulated with up to two operators of two templates varying the number of CQs from 1, 2, 10, 20, 50, to 100. The cost, in nanoseconds, of the Melting Templates process 16, cost of Generating and Storing a New Plan process 24, and cost of Accessing an Old Plan process 4.8, have the same values across the number of CQs. This means that the number of CQs does not affect the execution time of each process and that their values are independent. The Total Cost of Melting the Templates is equal to the number of CQs multiplied by the Processing Time of Melting the Template of each

CQ because each CQ takes the same time to be processed. The Total Cost of Generating and Storing a New Plan is 24 ns per CQ for the first 3 CQs and proceeds with 72 ns for the rest since there are always 3 new plans to generate for the first 3 CQs only and the rest would access old plans, hence the 72 ns is continuously the cost of generating the 3 plans of the 3 different scenarios. The Total Cost of Accessing the Old Plan of the first CQ is 0 ns since at this point there are no old plans generated yet to be accessed. The same applies when the number of complex queries is 2 and 3. Whereas, starting from 4 complex queries and for the rest of them, the Total Cost of Accessing Old Plans is equal to (the number of complex queries less 3) multiplied by the Processing Time of Accessing the Old Plan of each CQ. Finally, the Cumulative Cost is equal to the sum of the Total Cost of Generating and Storing New Plans and the Total Cost of Accessing Old Plans. The table shows that the Cumulative Cost of 'With Decider' has higher values than the Total Cost of Melting Templates of 'No Decider' in the first 5 rows, whereas it becomes less for the rest of the rows. Therefore and after the breakeven point, the time cost of 'With Decider' is less than the time cost of the 'No Decider'.

	Complex Queries with Up to 2 Operators of 2 Templates (3 Plans)						
	No Decider		**With Decider**				
	Cost of Melting Templates in Nanoseconds		Cost of Generating and Storing a New Plan		Cost of Accessing an Old Plan		
Number of Complex Queries	Processing Time of a Single Complex Query	Total Cost of Melting Templates	Processing Time of a Single Complex Query	Total Cost of Generating and Storing New Plans	Cost of Accessing a Single Plan	Total Cost of Accessing Old Plans	Cumulative Cost
1	16	16	24	24	4.8	0	24
2	16	32	24	48	4.8	0	48
3	16	48	24	72	4.8	0	72
4	16	64	24	72	4.8	4.8	76.8
5	16	80	24	72	4.8	9.6	81.6
10	16	160	24	72	4.8	33.6	105.6
20	16	320	24	72	4.8	81.6	153.6
50	16	800	24	72	4.8	225.6	297.6
100	16	1600	24	72	4.8	465.6	537.6

Table 7.1: The Time Cost of Each Process for Complex Queries with Up to 2 Operators of 2 Templates.

Table 7.2 shows similar results to the previous one but for complex queries that are formulated with up to five operators of 5 templates. The time costs of the processes Melting Templates, Generating and Storing a New Plan, and Accessing an Old Plan are respectively 32, 48, and 12.5 nanoseconds.

Complex Queries with Up to 5 Operators of 5 Templates (31 Plans)							
No Decider			With Decider				
	Cost of Melting Templates in Nanoseconds		Cost of Generating and Storing a New Plan		Cost of Accessing an Old Plan		
Number of Complex Queries	Processing Time of a Single Complex Query	Total Cost of Melting Templates	Processing Time of a Single Complex Query	Total Cost of Generating and Storing New Plans	Cost of Accessing a Single Plan	Total Cost of Accessing Old Plans	Cumulative Cost
1	32	32	48	48	12.5	0	48
2	32	64	48	96	12.5	0	96
3	32	96	48	144	12.5	0	144
4	32	128	48	192	12.5	0	192
5	32	160	48	240	12.5	0	240
10	32	320	48	480	12.5	0	480
20	32	640	48	960	12.5	0	960
30	32	960	48	1440	12.5	0	1440
31	32	992	48	1488	12.5	0	1488
32	32	1024	48	1488	12.5	12.5	1500.5
33	32	1056	48	1488	12.5	25	1513
50	32	1600	48	1488	12.5	237.5	1725.5
100	32	3200	48	1488	12.5	862.5	2350.5

Table 7.2: The Time Cost of Each Process for Complex Queries with Up to 5 Operators of 5 Templates.

Table 7.3 shows similar results to the previous one but for complex queries that are formulated with up to ten operators of 10 templates. The time costs of the processes Melting Templates, Generating and Storing a New Plan, and Accessing an Old Plan are respectively 47, 79, and 19.1 nanoseconds. It is clear that varying the number of templates affects the time cost of the three processes. The more templates there are to melt the higher the time cost.

	Complex Queries with Up to 10 Operators of 10 Templates (1023 Plans)						
	No Decider		With Decider				
	Cost of Melting Templates in Nanoseconds		Cost of Generating and Storing a New Plan		Cost of Accessing an Old Plan		
Number of Complex Queries	Processing Time of a Single Complex Query	Total Cost of Melting Templates	Processing Time of a Single Complex Query	Total Cost of Generating and Storing New Plans	Cost of Accessing a Single Plan	Total Cost of Accessing Old Plans	Cumulative Cost
1	47	47	79	79	19.1	0	79
2	47	94	79	158	19.1	0	158
3	47	141	79	237	19.1	0	237
4	47	188	79	316	19.1	0	316
5	47	235	79	395	19.1	0	395
10	47	470	79	790	19.1	0	790
50	47	2350	79	3950	19.1	0	3950
100	47	4700	79	7900	19.1	0	7900
1000	47	47000	79	79000	19.1	0	79000
1022	47	48034	79	80738	19.1	0	80738
1023	47	48081	79	80817	19.1	0	80817
1024	47	48128	79	80817	19.1	19.1	80836.1
1025	47	48175	79	80817	19.1	38.2	80855.2
2000	47	94000	79	80817	19.1	18660.7	99477.7
3000	47	141000	79	80817	19.1	37760.7	118577.7
4000	47	188000	79	80817	19.1	56860.7	137677.7
5000	47	235000	79	80817	19.1	75960.7	156777.7

Table 7.3: The Time Cost of Each Process for Complex Queries with Up to 10 Operators of 10 Templates.

7.4.2 Cost of Processing Templates of Complex Queries

The line chart in Figure 7.40 shows the cost of processing the templates of multiple CQs that have the same scenario when a CQ is formulated up to 2 operators of 2 templates. The X-axis is labelled with the 1st access, 2nd access, and nth access, representing respectively the 1st CQ, 2nd CQ, and nth CQ. The first line shows the cost of generating and storing a new plan for the new scenario when the "With Decider" is employed, and the second line shows the cost of Melting the Templates when the "No Decider" is employed. The Y-axis is the time cost in nanoseconds. With respect to the 1st Access of the 1st CQ with a new scenario and while using the "With Decider" method, the cost is 24 nanoseconds to generate and store a new plan for the templates. From the second CQ (Access) and on, the cost is reduced to 4.8 nanoseconds due to

the fact that the Plan of the first CQ is used by the second one instead of processing the second CQ templates again. The same applies for CQ numbered 3 and on. With 2 templates, there are $2^2 - 1$ scenarios and plans, so the same cost applies to the 3 scenarios no matter what the number of operators of the complex query is. While using the "No Decider" method, the cost of Melting the templates of the up to 2 operators of 2 templates is 16 nanoseconds no matter which CQ number it is and no matter the number of operators of the complex query.

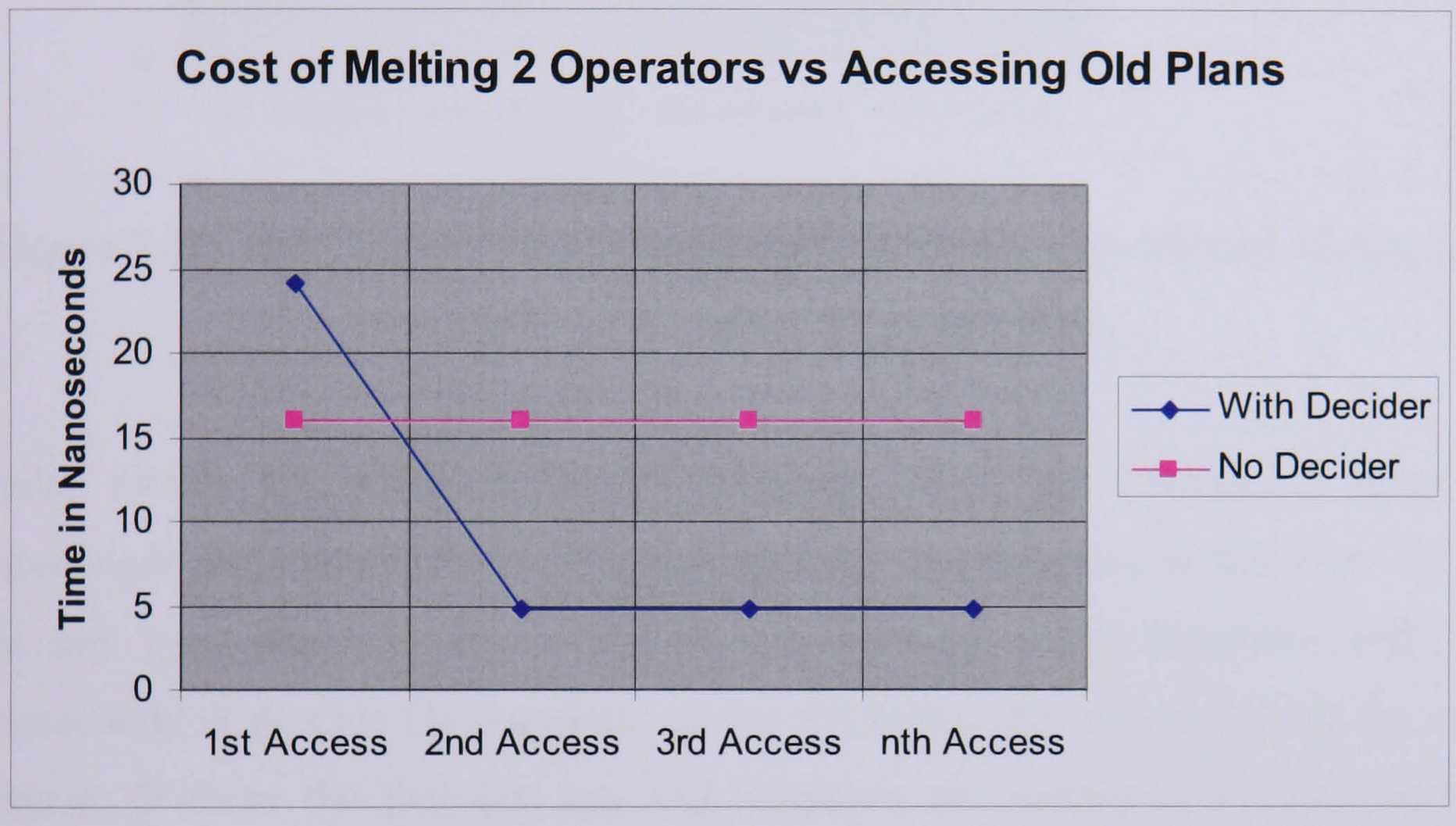


Figure 7.40: The Cost of Processing Complex Queries with Similar Scenarios with Up to 2 Operators of 2 Templates.

Similar results are obtained for 5 templates. With the Decider, it takes 48 nanoseconds to generate and store a new plan for the templates of the first CQ that uses any particular scenario of up to 5 operators of 5 templates and 12.5 nanoseconds to process the templates of the following CQs that are using the same scenario. Without the Decider, any CQ templates are melted in 32 nanoseconds. Figure 7.41 shows the cost of each CQ with and without the Decider.

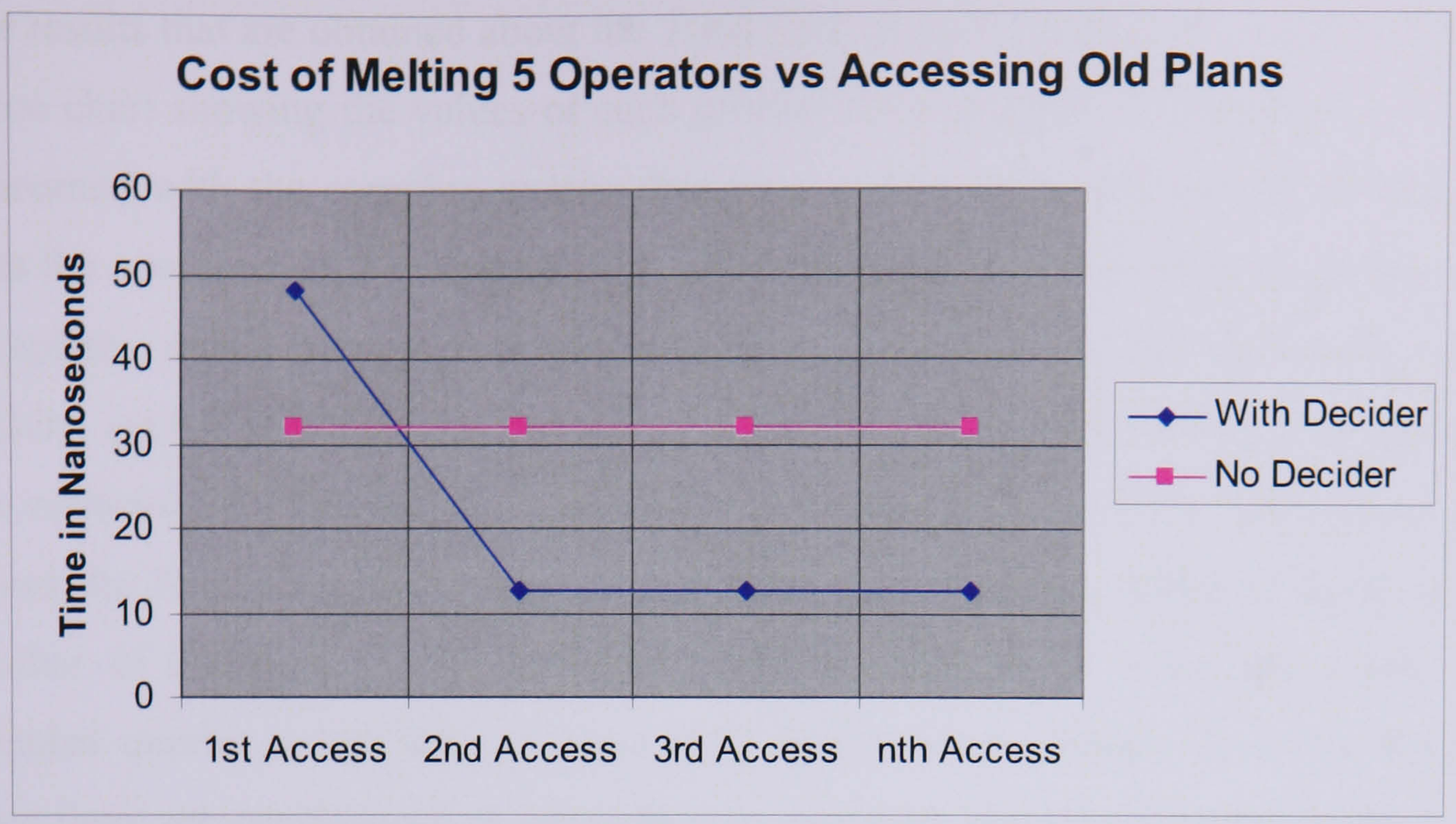


Figure 7.41: The Cost of Processing Complex Queries with Similar Scenarios with Up to 5 Operators of 5 Templates.

Similar results are obtained for 10 templates. With the Decider, it takes 79 nanoseconds to generate and store a new plan for the templates of the first CQ that uses any particular scenario of up to 10 operators of 10 templates and 19.1 nanoseconds to process the templates of the following CQs that are using the same scenario. Without the Decider, any CQ templates are melted in 47 nanoseconds. Figure 7.42 shows the cost of each CQ with and without the Decider.

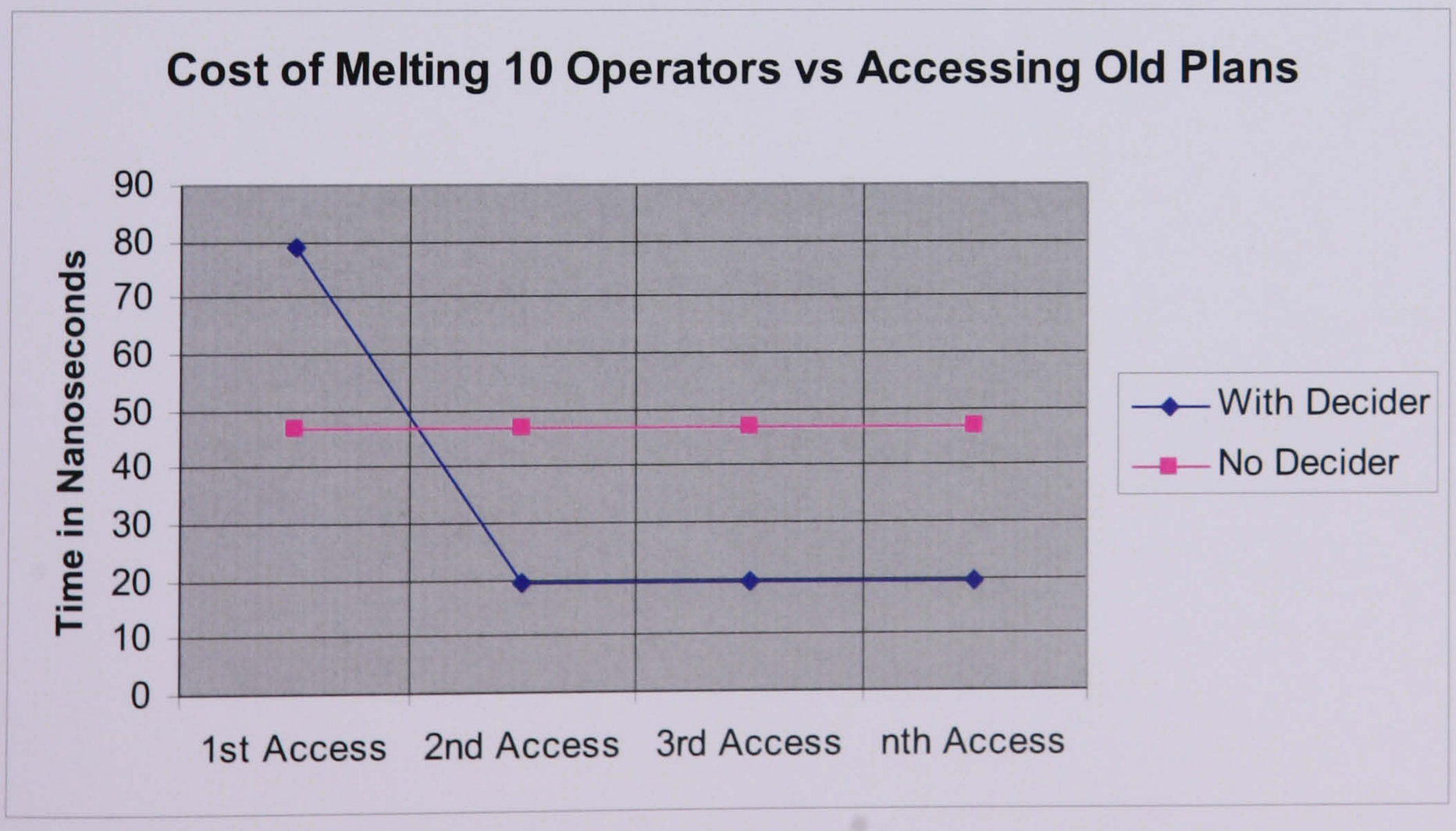


Figure 7.42: The Cost of Processing Complex Queries with Similar Scenarios with Up to 10 Operators of 10 Templates.

The results that are obtained about the Total Cost of each process are visualized using a line chart showing the values of each process for a number of CQs. Figure 7.43 is concerned with the complex queries that have similar scenarios and are formulated with the operators of 2 templates. The first line shows the Total Cost of Melting the Templates, which is equal to the product of the number of CQ and the Time Cost of Melting each Template. The line keeps on incrementing continuously at a constant rate reflecting the fact that the more CQs the higher the Total Cost. The second line shows the Total Cost of Generating and Storing a New Plan, which is equal to the number of complex queries multiplied by the value 24 ns when the number of complex queries is less than or equal to 3, then remains constant 72 ns for the rest since there are continuously 3 new plans to generate. The third line shows the Total Cost of Accessing Old Plans, which starts with the value 0 ns when the number of queries less than or equal to 3 since at this point there are no old plans generated yet to be accessed, then for the rest of the CQs increases at a constant rate based on the product of (the number of CQs less the number of plans 3) and the Time Cost of Accessing their Old Plans. The two lines that represent the Total Cost of Melting Templates and Total Cost of Accessing Old Plans reflect the fact that Accessing Old Plans is faster than Melting Templates. Similar results are visualized for complex queries with similar scenarios using 5 templates as shown in Figure 7.44 and using 10 templates as shown in Figure 7.45.

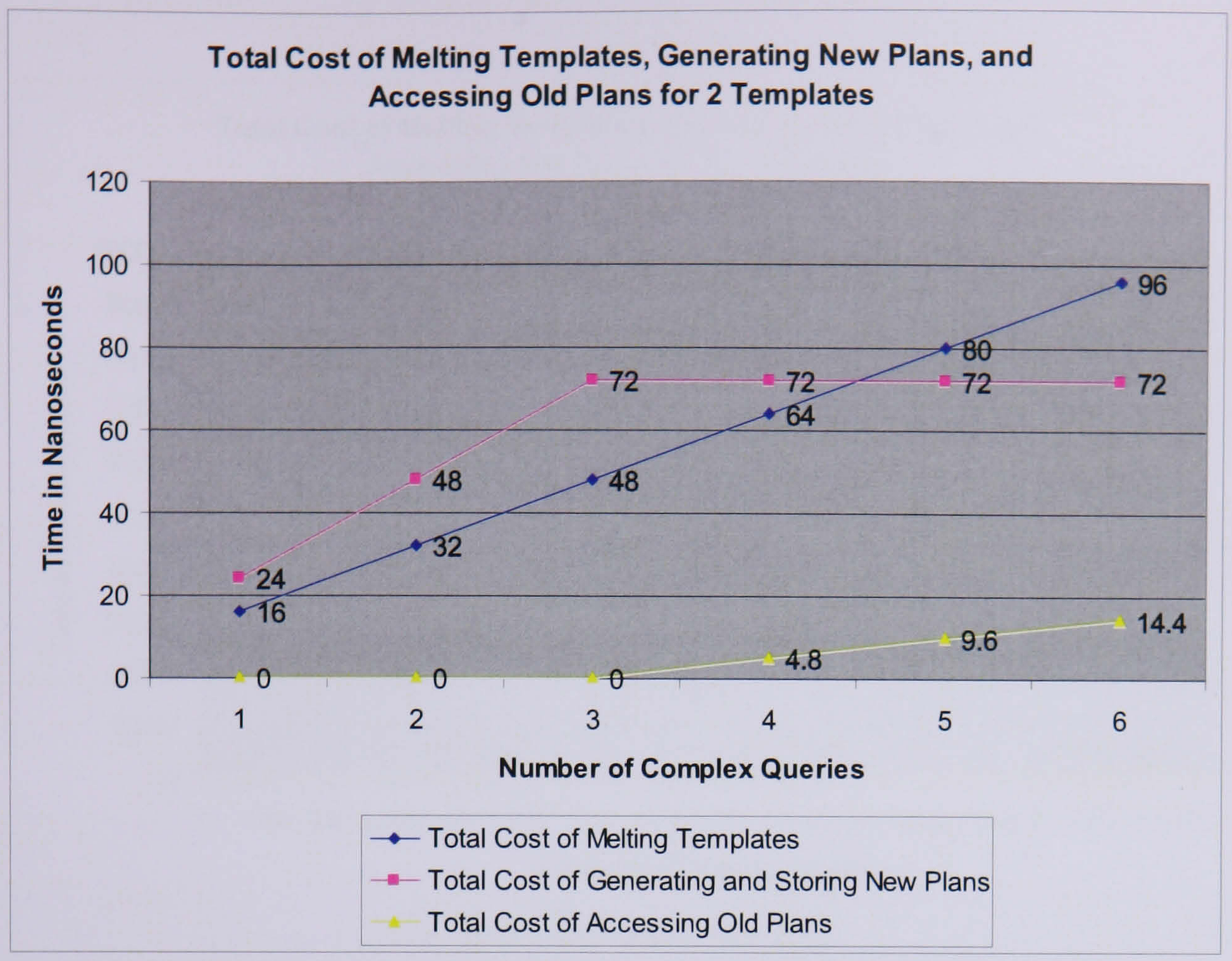


Figure 7.43: The Total Cost of Melting Templates, Generating and Storing New Plans, and Accessing Old Plans for 2 Templates for Complex Queries with Similar Scenarios.

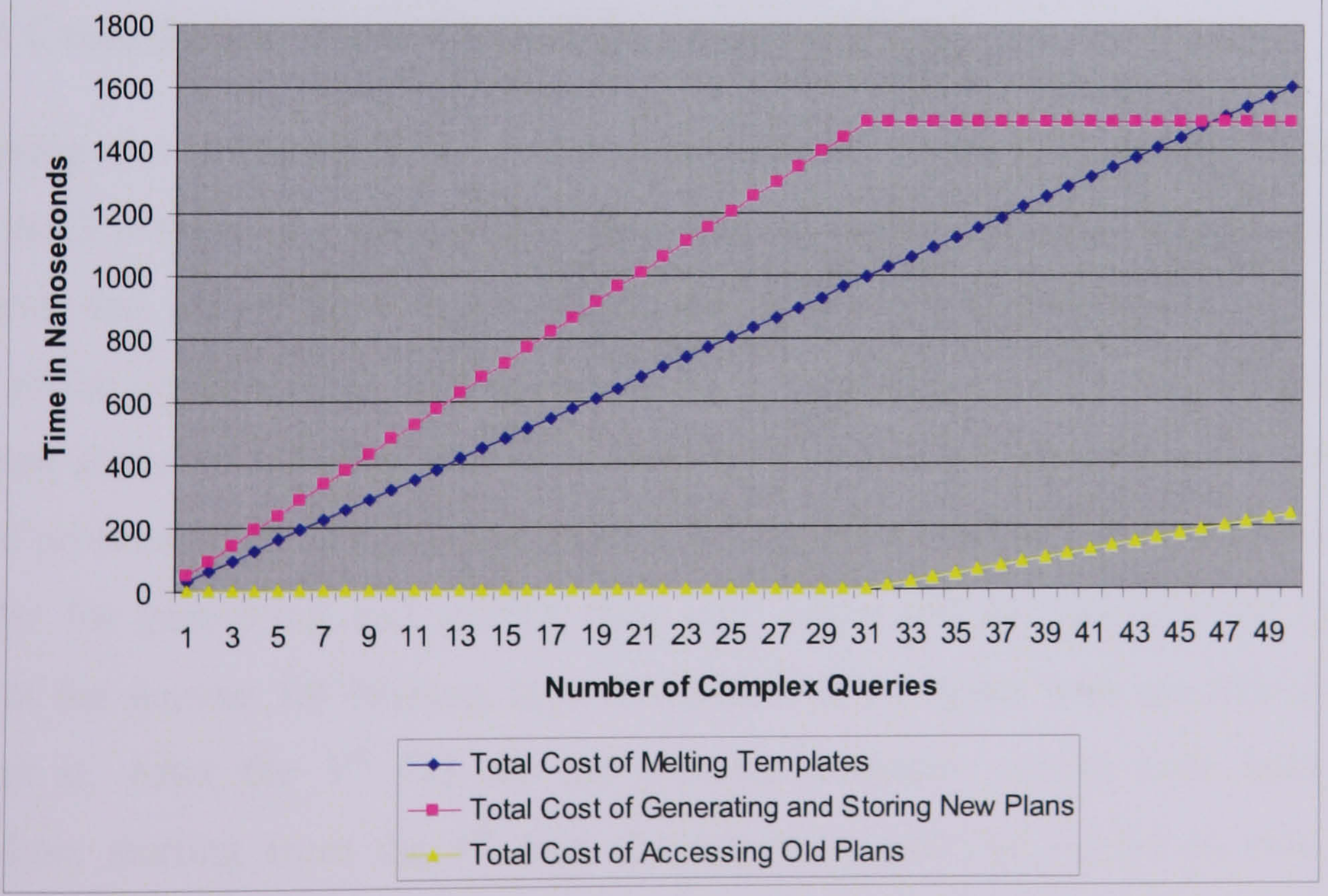


Figure 7.44: The Total Cost of Melting Templates, Generating and Storing New Plans, and Accessing Old Plans for 5 Templates for Complex Queries with Similar Scenarios.

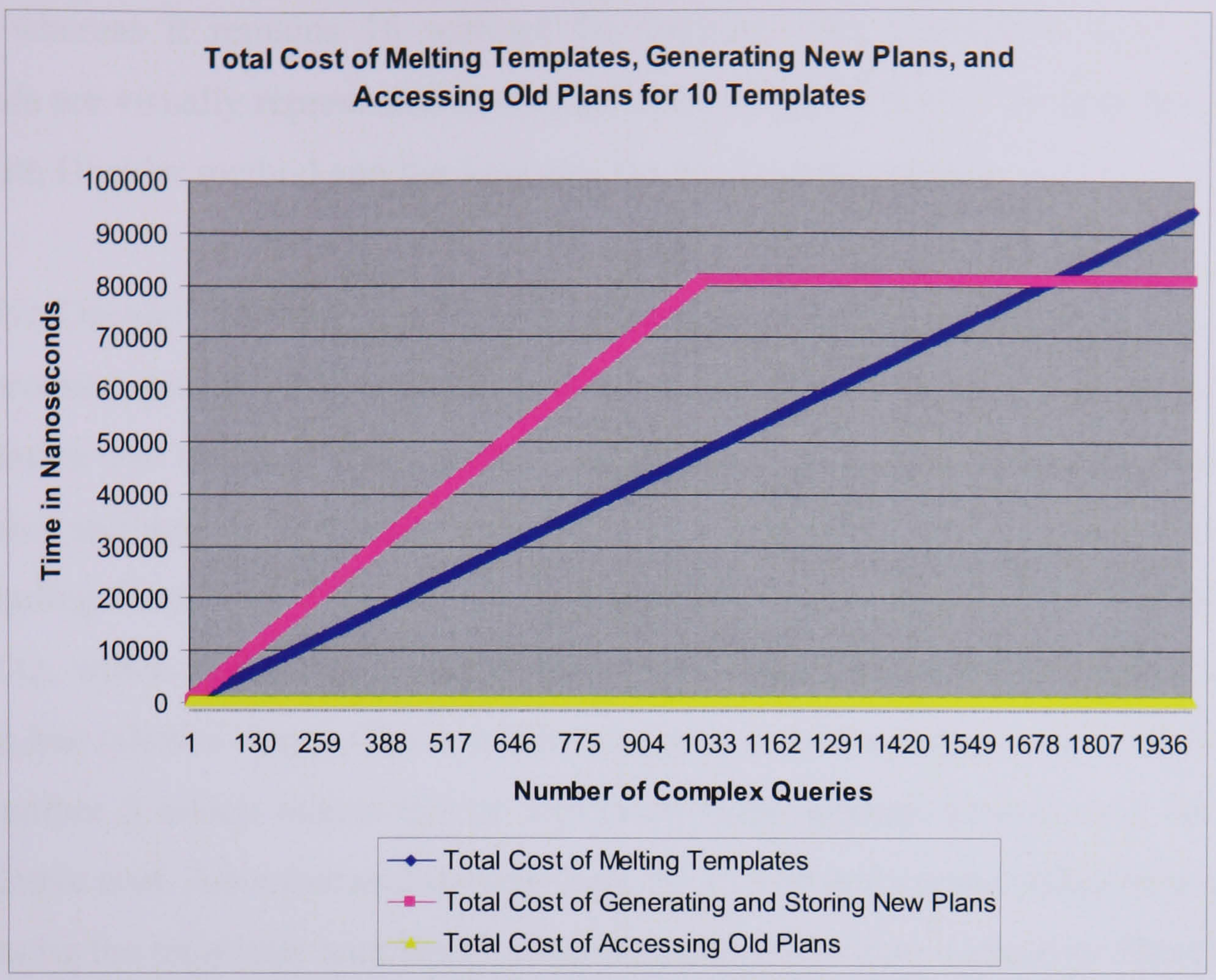


Figure 7.45: The Total Cost of Melting Templates, Generating and Storing New Plans, and Accessing Old Plans for 10 Templates for Complex Queries with Similar Scenarios.

7.4.3 Cumulative Time Cost of Processing Plans

Assuming that the probability of choosing a certain scenario is equally distributed over the 3 plans (when processing 2 templates), where each plan corresponds to a scenario, that are all the possible combinations that can be formulated with one or more of the operators that belong to a set of 2 templates, the first 3 CQs should use different scenarios, in other words, each new CQ is using a new scenario. The total cost of processing the templates of the first 3 CQs is 72 nanoseconds (3× 24) with the Decider for generating and storing their new plans and 48 nanoseconds (3× 16) without the decider for Melting their Templates. It is higher with the Decider than without it. After the 3rd CQ, all the possible scenarios would have been used. Therefore, starting from the 4th CQ, the scenarios must be similar to one of the previous 3. Hence, there would be no need to process again the scenarios that are similar to previous ones that have already been processed and it would be enough to access the previously generated plans. Therefore, the cost of processing the templates

of the 4th CQ becomes 4.8 nanoseconds per CQ with the Decider for accessing old plans whereas it remains 16 without the Decider. The cumulative costs of both methods are visually represented in the line chart of Figure 7.46. The dark line shows the With Decider method and the light one the No Decider method.

The "No Decider" line increments at the same degree (slope) which means that all the CQs produce results at the same speed. Whereas, the "With Decider" line has a higher cumulative cost than the "No Decider" line for the first 3 CQs because they use new scenarios and require generating and storing a new plan for each in 24 ns. After that and starting from the 4th CQ and on, the cumulative cost increases by 4.8 ns for each new CQ, which is the cost of accessing an old plan, reflected by a lower increase degree line (slower slope). The two lines intersect at the breakeven point which is the CQ number 5 which means that at this point both methods almost have the same cumulative cost. After that and starting from the 6th CQ and on, the cumulative cost of processing the templates with the Decider becomes lower than without it. Therefore, it can be concluded that employing the Decider is cost effective after processing the first 5 complex queries. Moreover, for each complex query that has a similar scenario to a previously processed one, the Decider results in a reduction of 70% of the time cost, which means that the decider is 3.33 times faster.

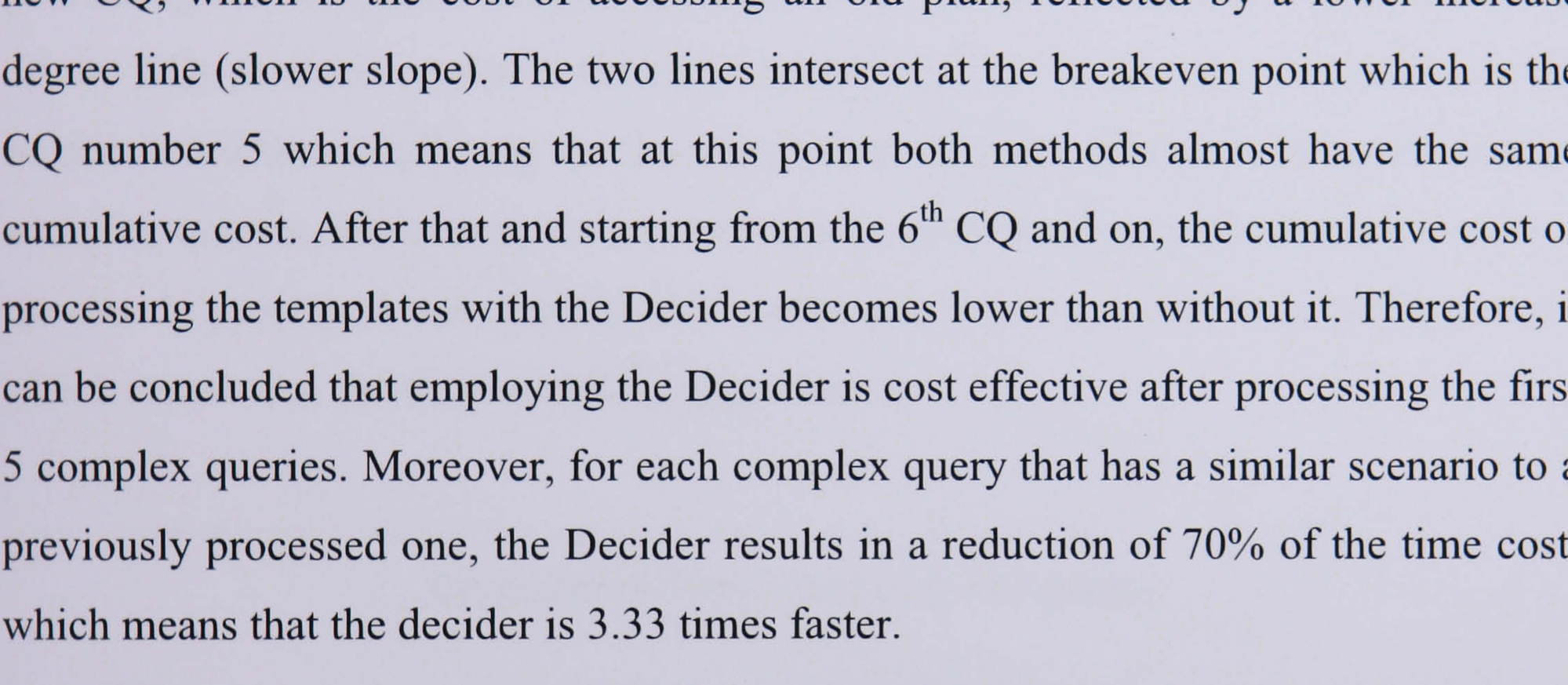


Figure 7.46: The Cumulative Cost of Processing Plans of 2 Templates.

Similar results are obtained for the total cost of processing plans of 5 templates. Assuming that the probability of choosing a certain scenario is equally distributed

over the 31 plans that are all the possible combinations that can be formed with the 5 templates, the first 31 CQs should use different scenarios. The total cost of processing the templates of the first 31 CQs is 1488 nanoseconds (31× 48) with the Decider for generating and storing their new plans and 992 nanoseconds (31× 32) without the decider for Melting their Templates. It is higher with the Decider than without it. After that, it becomes 12.5 nanoseconds per CQ with the Decider for Accessing Old Plans and remains 32 without the Decider. The cumulative costs of both methods are visually represented in the line chart of Figure 7.47. The dark line shows the With Decider method and the light one the No Decider method. The two lines intersect at the breakeven point which is the CQ number 56. After that, the total cost of processing the templates with the Decider becomes lower than without it. However, all the CQs produce results at the same speed without the Decider. Therefore, it can be concluded that employing the Decider is cost effective after processing the first 57 complex queries. Moreover, for each complex query that has a similar scenario to a previously processed one, the Decider results in a reduction of 61% of the time cost, which means that the decider is 2.57 times faster.

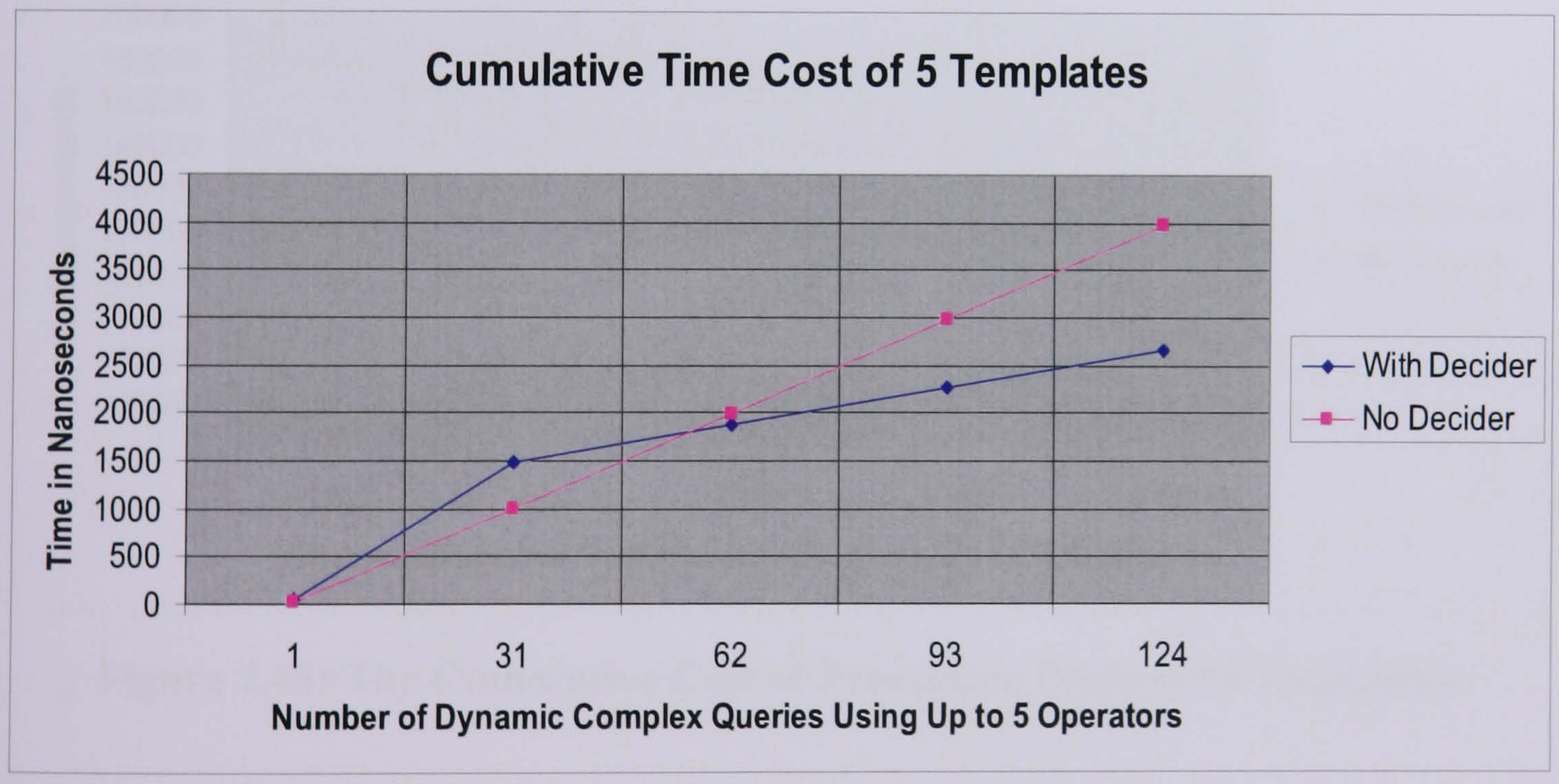


Figure 7.47: The Cumulative Cost of Processing Plans of 5 Templates.

Similar results are obtained for the total cost of processing plans of 10 templates. Assuming that the probability of choosing a certain scenario is equally distributed over the 1023 plans that are the possible combinations that can be formed with the 10 templates, the first 1023 CQs should use different scenarios. The total cost of

processing the templates of the first 1023 CQs is 80817 nanoseconds (1023 × 79) with the Decider for generating and storing their new plans and 48081 nanoseconds (1023 × 47) without the decider for Melting their Templates. It is higher with the Decider than without it. After that, it becomes 19.1 nanoseconds per CQ with the Decider for Accessing Old Plans and remains 47 without the Decider. The cumulative costs of both methods are visually represented in the line chart of Figure 7.48. The dark line shows the With Decider method and the light one the No Decider method. The two lines intersect at the breakeven point which is the CQ number 2196. After that, the total cost of processing the templates with the Decider becomes lower than without it. However, all the CQs produce results at the same speed without the Decider. Therefore, it can be concluded that employing the Decider is cost effective after processing the first 2196 complex queries. Moreover, for each complex query that has a similar scenario to a previously processed one, the Decider results in a reduction of 60% of the time cost, which means that the decider is 2.5 times faster.

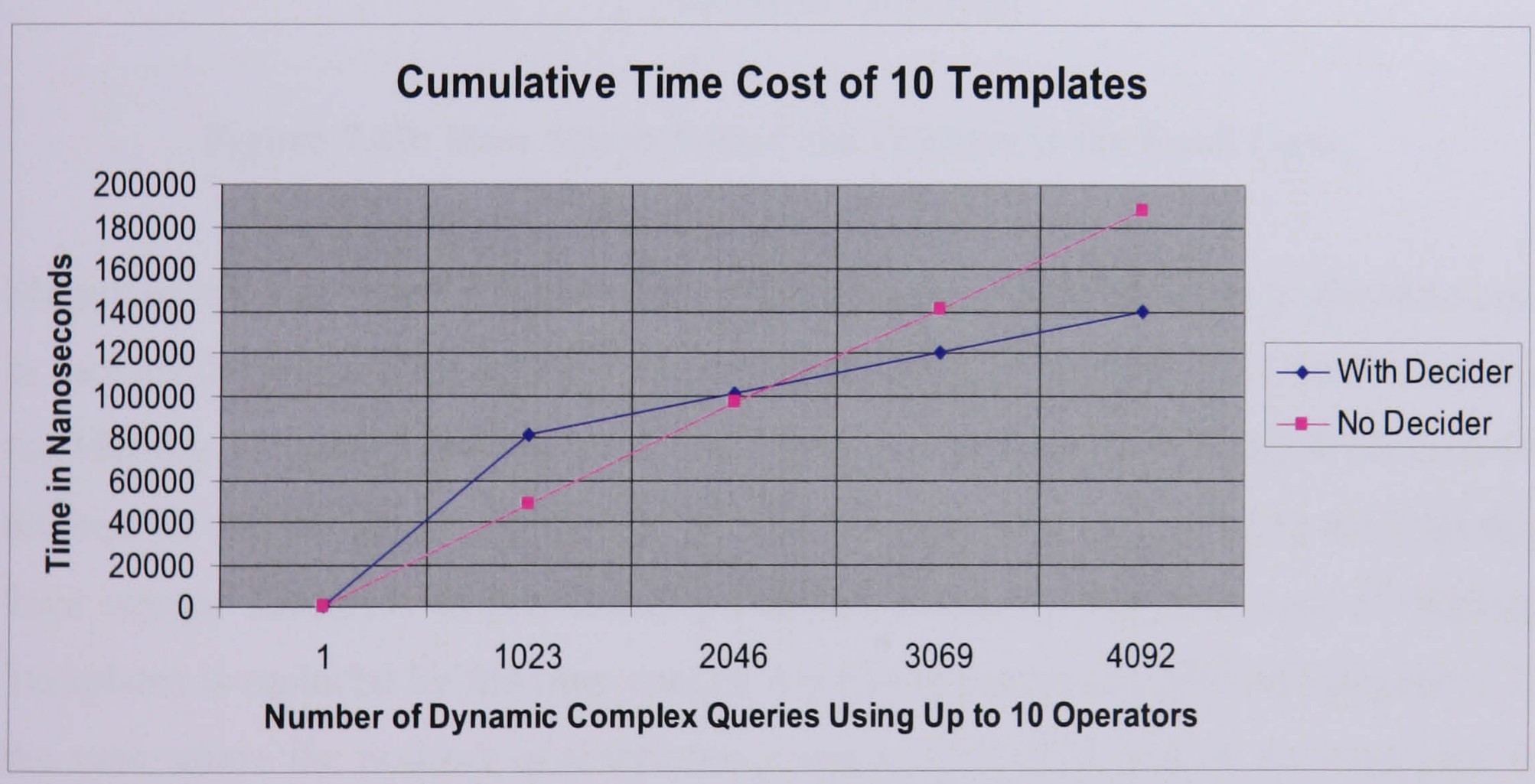


Figure 7.48: The Cumulative Cost of Processing Plans of 10 Templates.

7.4.4 Decider Speed and Cost Effectiveness

It can be concluded from the previous results that for each complex query that has a similar scenario to a previously processed one, the Decider results in a reduction of 70%, 61%, and 60% of the time cost, which means that employing the decider is 3.33, 2.57, and 2.5 times faster, when the number of templates is 2, 5, and 10 respectively. It can also be concluded that on the average, the Decider is 2.8 times faster than

without it. The bar chart in Figure 7.49 is used to visualize how fast the Decider is in each of the cases.

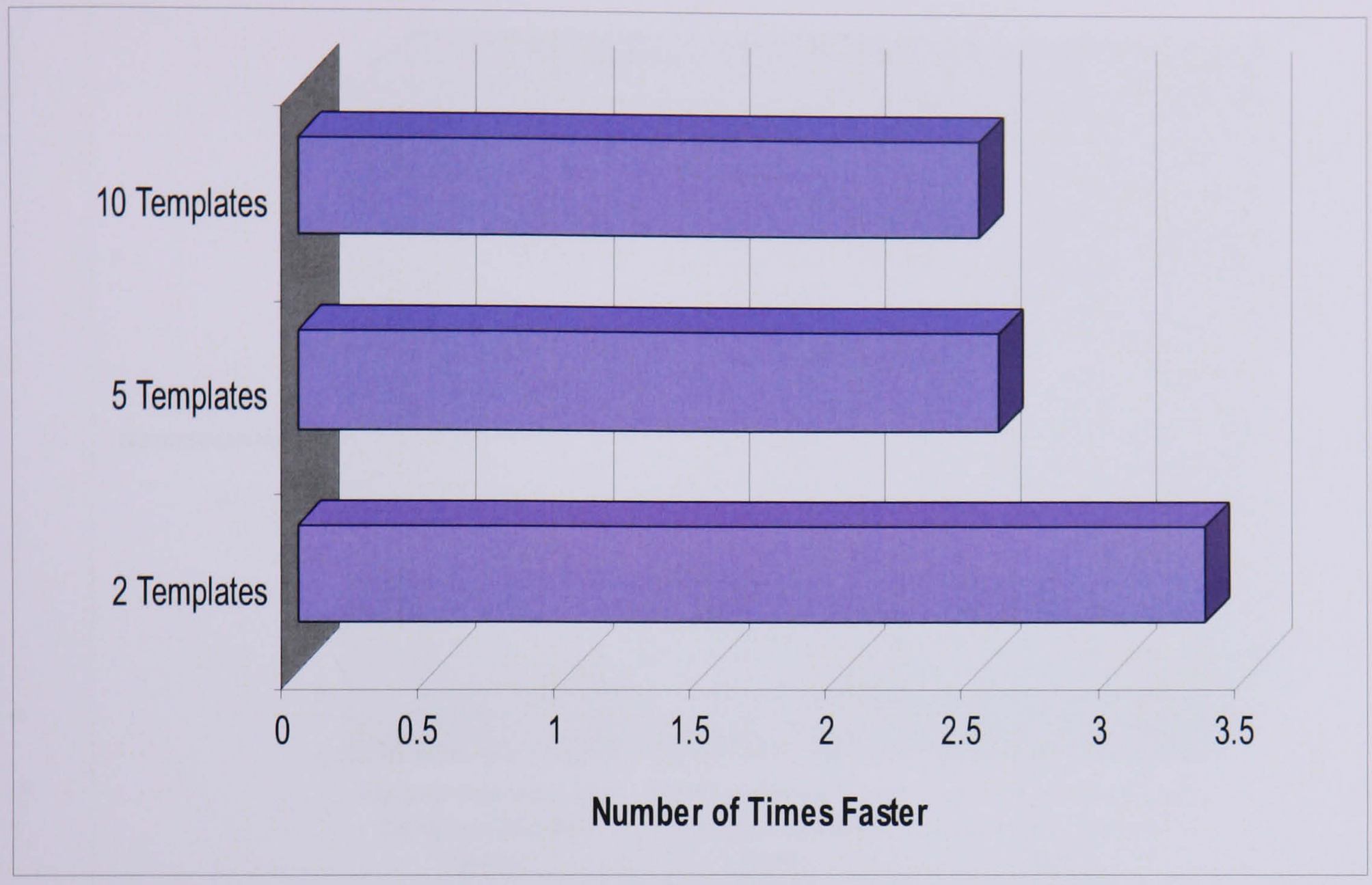


Figure 7.49: How Much Faster the Decider is for Each Case.

Moreover, the bar chart in Figure 7.50 is used to show the reduction in the time cost in each of the cases. The light bars represent the time cost of the No Decider method for Melting Templates and the dark bars represent the time cost of the With Decider method for Accessing the Previously Melted Templates. With respect to the CQs that have similar scenarios to previously processed scenarios, the time cost of Melting Templates is replaced by the time cost of Accessing previously Melted Templates. In the case where the number of templates is respectively 2, 5 and 10 the time cost of Melting Templates is reduced from respectively 16, 32, and 47 to respectively 4.8, 12.5, and 19.1 nanoseconds which is the time cost of Accessing previously Melted Plans. The Decider results in a reduction of 70%, 61%, and 60% in the time cost of without it in the case where the number of templates is respectively 2, 5, and 10. On the average, the Decider results in a reduction of 64% of the time cost of without it. The cost of melting all the templates at the beginning of execution is equal to 72 (24 ns × 3 plans), 1488 (48 ns × 31 plans), and 80817 (79 ns × 1023 plans) nanoseconds in the case were the number of templates is respectively 2, 5, and 10.

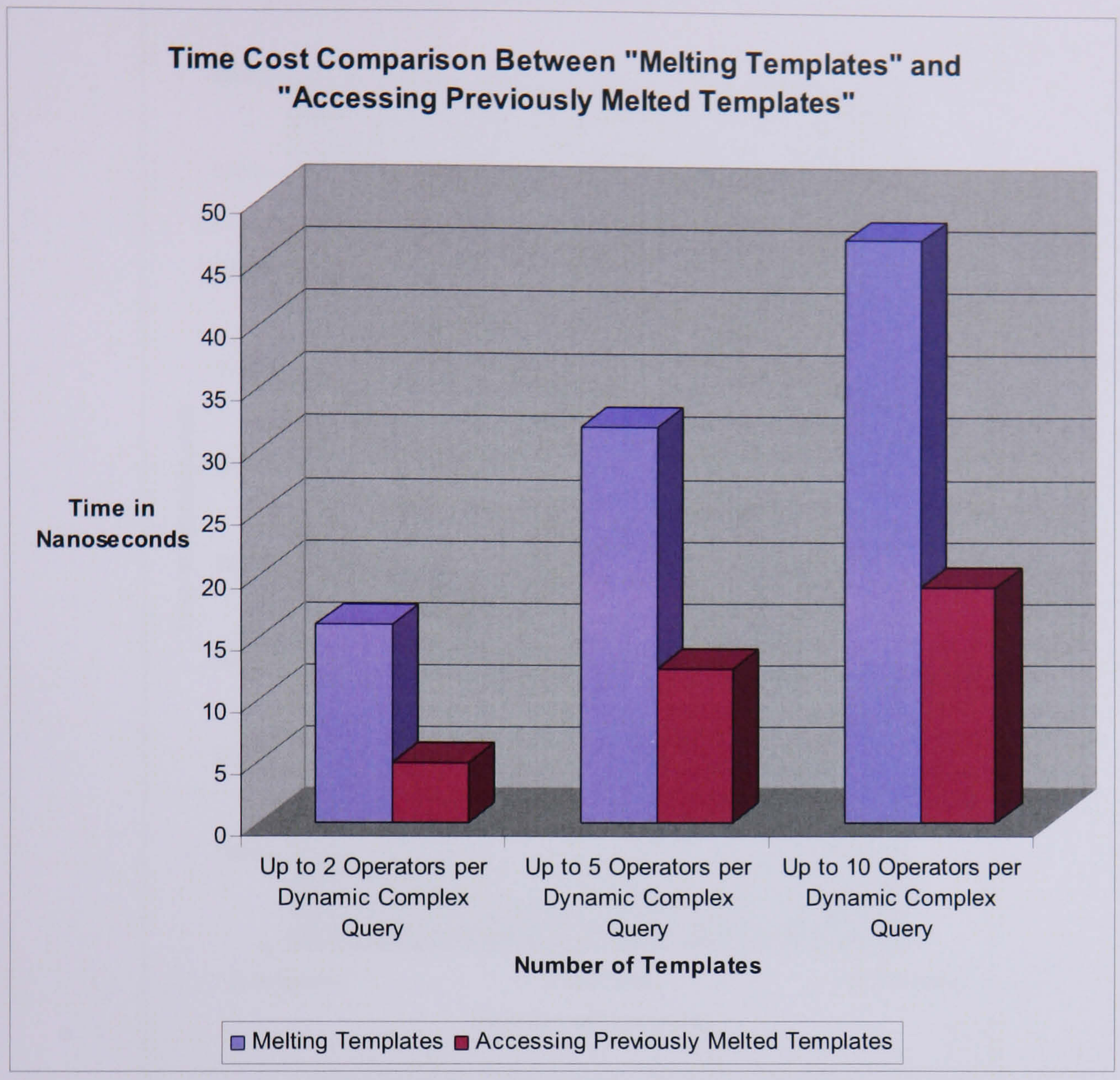


Figure 7.50: The Reduction of the Time Cost of No Decider when Using Decider.

Finally, the area chart in Figure 7.51 visualizes the costs of melting all the templates plans once at the beginning of execution of the Decider. By doing so, it is guaranteed that all the invoked dynamic complex queries produce the fastest results, i.e., the most time cost optimized results, by eliminating their time cost of generating and storing a new plan and accessing previously generated plans of melted templates.

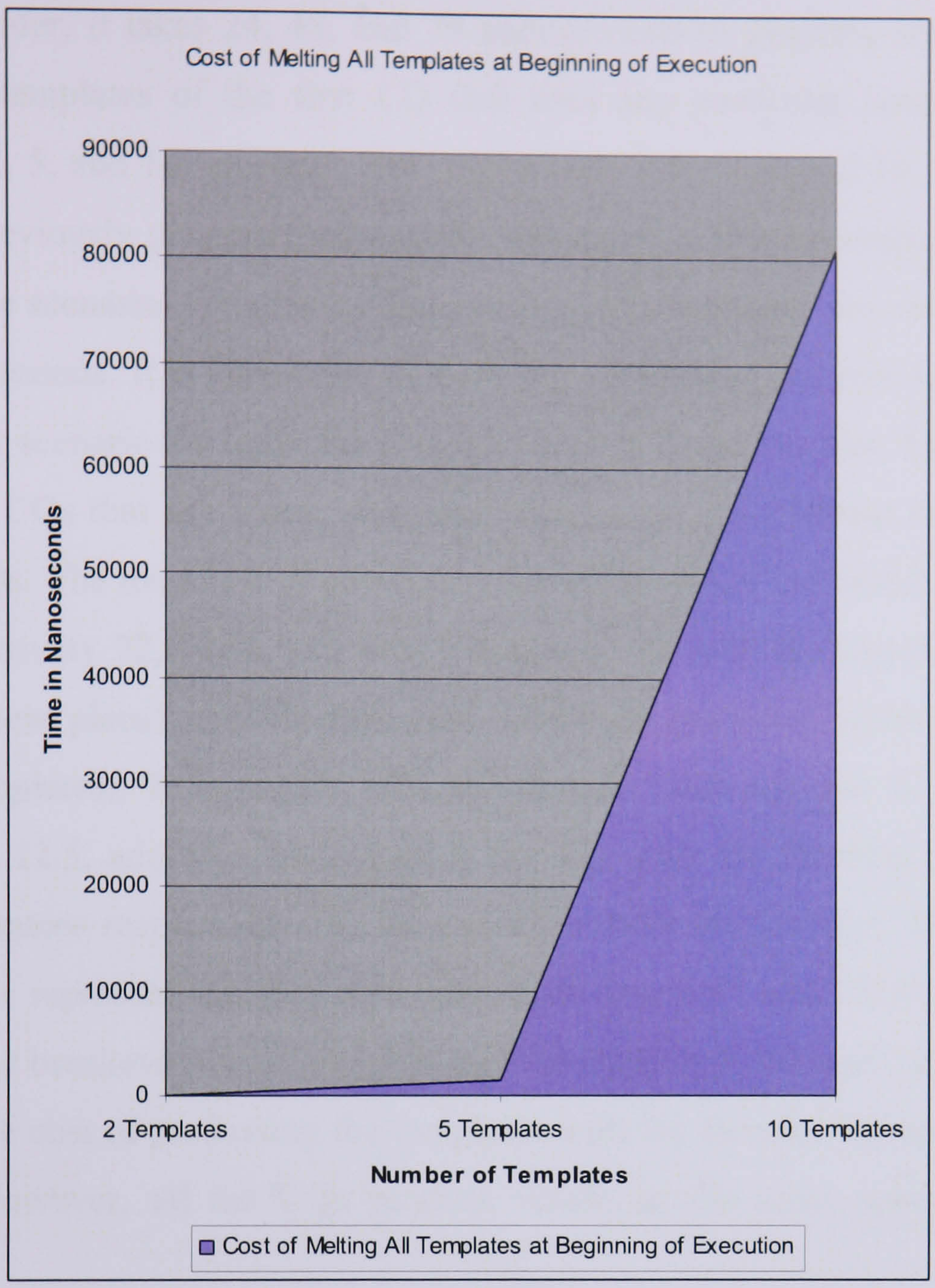


Figure 7.51: The Cost of Melting All Templates Plans at Beginning of Execution.

7.4.5 Results Discussion

After having analyzed the results of the experimental studies, a discussion of the analysis leads to draw conclusions and reach an evaluation of the whole system. The cost of the Melting Templates process, cost of Generating and Storing a New Plan process, and cost of Accessing an Old Plan process, have the same values across the number of complex queries (CQ) that varied from 1, 2, 5, 10, 20, 50, to 100. This means that the number of CQs does not affect the execution time of any of these processes. Their values are independent of the number of CQs. It is clear that varying the number of templates affects the time cost of the three processes. The more templates there are to melt the higher the time cost. This is an obvious fact to consider in the case where extra templates are added to the existing ones.

With the Decider, it takes 24, 48, and 79 nanoseconds to generate and store a new plan for the templates of the first CQ that uses any particular scenario of up to respectively 2, 5, and 10 templates and respectively 4.8, 12.5, and 19.1 nanoseconds to access a previously generated plan of the templates of the successive CQs that are using the same scenario. Without the Decider, any CQ templates are melted in 16, 32, and 47 nanoseconds. It is concluded that for any CQ except the first one that uses a new particular scenario the time cost is much lower with the Decider. More generally, the first 2^t -1 CQs that use a new particular scenario produce slower results than the rest of the CQs. The total cost of processing the templates of the first 3, 31, and 1023 CQs is respectively 72, 1488, and 80817 nanoseconds with the Decider (generating and storing new plans) and 48, 992, and 48081 nanoseconds without the decider (Melting Templates). It is higher with the Decider than without it. After that, it becomes 4.8, 12.5, and 19.1 nanoseconds per CQ with the Decider (accessing old plans) and remains respectively 16, 32, and 47 without the Decider. The cumulative cost lines that represent the two methods "With Decider" and "With No Decider" intersect at the breakeven point which is the CQ number 5, 56, and 2196. After that, the cumulative cost of processing the templates with the Decider becomes lower than without it. However, all the CQs produce results at the same speed without the Decider.

The Decider is 3.33, 2.57, and 2.5 times faster than without it in the case where the number of templates is respectively 2, 5, and 10. On the average, it is 2.8 times faster than without it. In the case where the number of templates is respectively 2, 5 and 10 the time cost is reduced from respectively 16, 32, and 47 to respectively 4.8, 12.5, and 19.1 nanoseconds. It is concluded that the Decider results in a reduction of respectively 70%, 61%, and 60% in the time cost of without it. On the average, it results in a reduction of 64% of the time cost of without it. The cost of melting all the templates at the beginning of execution is equal to 72 (24 ns × 3 plans), 1488 (48 ns × 31 plans), and 80817 (79 ns × 1023 plans) nanoseconds in the case were the number of templates is respectively 2, 5, and 10. In order to be fair to everybody especially the first complex queries that use a particular scenario, it is suggested to melt the templates of all the plans at the beginning of the execution of the Query Melting Processor. Thereby, it is guaranteed that all the complex queries produce results at the same optimized time cost.

7.5 Conclusion

This chapter provides an integration of the IVQL, Query Melting paradigm, Query Optimization, and Time Cost Optimization. Its emphasis is to explore the Query Melting Processor on the Proximity Analysis (k-Nearest-Neighbours and Buffer) queries in the context of dynamic complex queries. It is implemented using a Tourist Application for mobile users using the map of Paris. The user formulates the complex query which is sent to the geographic information system ArcGIS server. The query is processed and its result map is sent back to the user.

The implementation of the system achieved the aims that were set for the case study. The first aim is the implementation of the IVQL using a mobile phone emulator in order to allow the visual query formulation of dynamic complex queries. The second aim is the Query Melting Paradigm which is implemented using the Query Melting Processor. QMP performs common sub-expression elimination CSE by eliminating all the repetitions of the same process in different plans. This task is performed by melting the templates. It allows sharing object of interest between multiple queries by setting the object once and using it for many queries. When two queries share the same spatial area or when an area is included in another, the QMP uses the shared area for multiple queries. It allows sharing an interval of time if it is included or equal to the interval of another query with the same object of interest. The previous three tasks are performed by melting the values and objects of the queries. Sharing the intermediate results is done through using the same underlying space (map) for all the queries of the same user. This is done through executing the plan of each user as a thread which is performed by the Plan Executor.

The third aim Query Optimization is achieved through decomposing the dynamic complex queries, eliminating the sub-expressions, generating the Global Evaluation Plan, and evaluating the plan. The last aim of this case study is Time Cost Optimization. It is to investigate and analyze the time cost of the Query Melting Processor. Two methods of processing were compared. The first is "With No Decider" where the complex queries templates are melted for each complex query. The second is "With Decider" where the templates are melted once per scenario and

stored in RAM for later retrieval by similar successive scenarios. A comparison between the time costs of the two processing methods is done in order to conclude which of them is cost effective.

In order to be able to achieve the purpose of the case study, an explicit scenario, using three users, was presented to show the user interface and query melting in action in order to show how the system works and an experimental evaluation was carried out to evaluate the cost effectiveness of the system. The experimental quantitative results were reported and analyzed using tables, line charts, bar charts, and area charts. A comparison between the time costs of the two methods is studied with the aim to conclude which of them is cost effective. The analysis of the results of the experimental study leads to a number of significant conclusions. The number of complex queries does not affect the execution time of any of the processes and their values are independent of the number of complex queries. Also, the more templates there are to melt the higher the time cost. For any complex query except the first one that uses a new scenario the time cost is much lower with the Decider. More generally, the first $(2^t -1)$ complex queries that use a new particular scenario produce slower results than the successive complex queries. However, all the complex queries produce results at the same speed without the Decider.

The Decider is 3.33, 2.57, and 2.5 times faster than without it in the case where the number of templates is respectively 2, 5, and 10, and on average it is 2.8 times faster than without it. It results in a significant reduction of respectively 70%, 61%, and 60% in the time cost of without it, and on average it results in a significant reduction of 64% of the time cost of without it. In order to get efficient results it is suggested to melt the templates of all the plans at the beginning of the execution of the Query Melting Processor. Thereby, it is guaranteed that all the complex queries produce results at the same optimized time cost. As a final conclusion, it can be said that the Decider has proven to be significantly cost effective.

Chapter 8 – Conclusions and Future Work

8.1 Conclusion

The primary aims and contributions of this thesis are the design, implementation, and evaluation of a new visual query language and novel query optimization strategies for Mobile GIS. The work which was carried out to achieve these aims, as well as the results of the work, was reported in the previous chapters of this thesis. In this section the achievements and conclusions which have been previously drawn will be summarized.

The research began by a thorough investigation into existing visual query languages in order to examine their user interface and query building process. The investigation started by discussing and comparing the specification of which aspects of VQL were evaluated, determining the evaluation method, discovering the type of queries that were handled, and their advantages as well as deficiencies with respect to current Mobile GIS requirements. It also highlighted a suitable query formulation process, user interface, and evaluation method for the implementation of the proposed visual query language.

Similarly, the thesis has investigated existing query optimization strategies in GIS in order to determine their various approaches and examine how they are applied in various domains. The investigation started by discussing and comparing the approaches of query execution, sharing common aspects, evaluation method, and their advantages as well as deficiencies with respect to current dynamic complex queries. Moreover, it investigated a suitable propagation ruler mechanism and the existing time cost optimization strategies in order to specify how they are implemented with different type of queries. Also, the thesis highlighted a suitable paradigm for the implementation of the Query processor as well as a suitable decision making mechanism for the time cost optimization.

The identified deficiencies of the visual query languages have been addressed in this thesis by developing the Iconic Visual Query Language (IVQL) which is able to handle visually dynamic complex queries for Mobile GIS. The proposed IVQL is based on using smiley icons to represent operators, values, themes, and objects. The language constructs, architecture and query formulation process have been explained and the efficiency of the user interface and query formulation have been demonstrated in the implementation and evaluation of the language. Both the user testing and the user satisfaction have been chosen in order to evaluate the expressive power of the smiley icons, their level of recognition, the ease of use, the user interface, the query building and formulation, and the expressive power of the IVQL. The results showed that the subjects understood quickly the smiley icons, user interface, query building and formulation, and proved to be satisfied with the aspects of the expressive power of the language. They also found that the user interface was very good, the query building and formulation easy, and that the IVQL language had a very good expressive power. The advantage of the evaluation implemented in this work over the evaluations implemented in the review of literature was that all the aspects of usability of IVQL were evaluated whereas in each of the other reviewed evaluations one aspect of usability only was evaluated such as query writing, user interface, and expressiveness of icons and language.

The results of the analysis showed that the subjects found that the smiley icons had a good expressive power, a high level of recognition, and are easy to use. They showed that people from different backgrounds like programmers and non-programmers are expected to perform the same when using the IVQL visual query language. They do not need to have any programming experience in order to be able to use easily IVQL to formulate queries or understand its visual language. It is important to note at this point that as a result of the testing some icons were completely changed and improved according to the suggestions proposed by the subjects who undertook the evaluation. It was found also that the icons 2 (Hotel), 20 (Taxi), and 30 (Ice Skating) were better recognized by programmers than by non-programmers. However, the results showed they may simply be random anomalies. Finally, the results proved that there is some evidence for thinking that the programmers perform better than the non-programmers. A comparison between IVQL icons evaluation with the ones conducted in the reviewed visual query languages showed that other work evaluated the expressive

power of the language by user testing whereas it was done in IVQL by user satisfaction. The results that were reported by other work showed that their icons had a 65% expressive power whereas the IVQL evaluation results showed that the icons had a 90% expressive power, hence, reflecting a higher expressive power in favour of the IVQL icons.

With respect to the query formulation, it was noted that the programmers performed better than the non-programmers in the first 5 simple queries and the last 5 complex queries whereas the non-programmers performed better than the programmers in the intermediate ones. Moreover, it was clearly noted that the programmers performed better than the non-programmers in 24 questions of the user satisfaction where a difference in the results was observed, with the probability of this happening by random chance is around 1 chance in 17 million. Hence, it was concluded that there is therefore a strong indication that the programmers perform better than the non-programmers though the magnitude of this difference is small. When comparing the results of the evaluation of IVQL query formulation with the ones reviewed in the literature it was noted that in IVQL the programmers performed in general as good as the non-programmers. This fact proved that IVQL is equally understood by people from different backgrounds. In the evaluation done other work the programmers performed better than the non-programmers due to the fact that the visual query resembled a lot the form of an SQL statement or GIS users performed better than non-GIS users due to the fact the GIS users had a good background in querying GIS applications. Hence it was concluded that the query formulation in IVQL could be more easily understood by the generic public than the ones that were reviewed in the literature. Moreover, it was important to note that none of the reviewed visual query languages used the user satisfaction in order to evaluate the aspects that were evaluated in IVQL. This fact was considered as an advantage of the evaluation of IVQL over the others.

Moreover, the novel aspects of the developed query optimization strategies have been demonstrated in the implementation of the Query Melting Processor that handles dynamic complex queries for proximity analysis. This started with an in-depth investigation of the GIS operators and their corresponding execution plans in order to determine the commonalities between them, which common parts could be shared and

which ones re-ordered. The investigation covered all types of operators namely the static operators, dynamic operators with one-predicate, and dynamic operators with multiple predicates. The investigated execution plans have proved to have significant commonalities hence the sharing paradigm was employed in processing them. The proposed Query Melting paradigm is based on the sharing paradigm, push-down approach, traditional query optimization, sharing the global execution plans (GEP), and a new melting ruler mechanism that acts like a sliding ruler to allow sharing the functions of various templates, input data, intermediate results, underlying space maps, spatial areas, time intervals, objects of interest, and query results. The deficiency of the sharing sub-plans strategy has been addressed in this thesis using a new query optimization paradigm, called Sharing Global Execution Plans (GEP), based on sharing totally a previously generated GEP by multiple users who formulate similar scenarios of queries, where a new Decision Making Mechanism called time cost optimizer (TCOP) has been employed to optimize the time cost.

The design, implementation, and theoretical evaluation of the new Query Melting Processor were carried out using the query melting paradigm, query optimization, and time cost optimization. The query melting paradigm includes sharing the functions, objects, spatial areas, and time intervals, the query optimization includes decomposition of the queries, common sub-expression elimination, generation of global execution plan, and evaluation of the plan, and the time cost optimization is performed through sharing the global execution plans. The design of the Query Melting Processor was done using a number of visual diagrams such as the Use case Diagram used to visualize the activities and tasks of a system showing at the same time the actor/actors associated with each activity and the Time Sequence Table Diagram showing the order as well as the duration in time of each task or activity including the user, and the Architecture of the Query Melting Processor Diagram illustrating the input tables with their attributes, the variables, fields, or arrays that are stored in the memory, the processes or procedures that are executed, and the output of the processor. The algorithm of each process of the Query Melting Processor has been presented and the computational cost estimated using the Big-Oh notation. Similarly, the cost effectiveness of QMP has been evaluated and proved by conducting a theoretical evaluation that was able to quantify the cost and profit of each phase.

Finally, a case study involving proximity analysis multiple dynamic complex queries was experimentally evaluated using an integration of the IVQL, Query Melting Paradigm, and Time Cost Optimization strategy. The implementation involved a Tourist Application for mobile users using the map of Paris. The prototype of the visual language that was installed on a mobile phone emulator to generate the queries that are sent to the ArcGIS server has proven the effectiveness of the complex queries formulation. The query is processed by the QMP that generates an optimized global execution plan, executes it, and sent the resulting map to the user. An explicit scenario was presented using three users in order to show how the system works. The experimental evaluation was conducted to evaluate the time cost of the QMP as well as the new introduced approach, sharing global evaluation plan, based on the new decision making mechanism (TCOP). The developed system was able to prove the concept of optimizing the global evaluation plan and the time cost optimization strategy.

The implementation of the system has achieved the aims that were set for the case study namely, the implementation of the IVQL using a mobile phone emulator in order to allow the visual query formulation of dynamic complex queries, and the Query Melting Paradigm implemented using the Query Melting Processor to perform common sub-expression elimination to allow sharing object of interest between multiple queries, sharing common spatial areas, and sharing an interval of time. The previous three tasks have been performed by melting the values and objects of the queries, using the same underlying space (map) for all the queries of the same user. Moreover, the experimental evaluation has demonstrated that the Time Cost Optimization strategy proved to have significant cost effectiveness by conducting a comparison between the two methods "With No Decider" where the complex queries templates are melted for each complex query and "With Decider" where the templates are melted once per scenario and stored in RAM for later retrieval by similar successive scenarios. The analysis of the results have shown that the first complex query that uses a new scenario has a lower time cost with the Decider, more generally, the first $(2^t - 1)$ complex queries that use a new particular scenario produce slower results than the successive complex queries. However, all the complex queries produce results at the same speed without the Decider. Finally, it was concluded the new paradigm sharing global execution plans by employing the new decision making

mechanism strategy has proven to be significantly cost effective because it is faster than without it and results in a significant reduction in the time cost of processing the dynamic complex queries that use similar scenarios.

8.2 Future Work

In this section we give suggestions about how the work presented in this thesis can be carried out further. The future work which can presently be seen in this field may be classified into a number of categories.

The first category for future research is related to the fact that the current implementation IVQL does not support the queries that are based on previous queries results, such as to find the nearest hotel to the nearest restaurant to the nearest university. This can be achieved by using some kind of connector icons which can be investigated and integrated into IVQL. The connector should be studied and analyzed starting with connecting two simple queries, then three, and so on. Many different connector icons might be proposed in order to cover all possible combinations of operators in the queries. For example, making a simple query about the nearest hotel followed by another query about all theatres located within 500 meters from the hotel would produce automatically all theatres next to the nearest hotel. An example of a new connector that might be used for such an operation is a 'Filter' Operator that connects two or more simple queries instead of the 'and' operator. Simple queries that are connected with the 'and' operator produce the union of the results of all the queries on one single map, whereas the ones that are connected with the 'Filter' operator produce the results of the first simple query, apply the second simple query on the results of the first one, and so on until all the simple queries are executed. Hence, the results that are produced on the map actually represent the last simple query applied on the results of its previous ones.

The second category is related to the extension of IVQL. As was seen in the evaluation of the language a number of icons have been redesigned which has clearly demonstrated how easy to extend and re-adapt the language. Extending IVQL can be considered in future work by designing and adding new icons based on new themes, categories, and functionalities which can be easily integrated. Moreover, although

IVQL has been designed with mobile devices in mind it can also be applied to palmtops, laptops, desktops etc.

The third category is related to the optimization strategies. The experimental evaluation of the case study has indicated the possibility of generating all the global evaluation plans at the beginning of the execution of the query melting processor which can be considered in future work. An experimental evaluation could be carried out to compare it with the existing Decider based approach in order to identify which of them is cost effective in terms of time and memory space. Also, a number of other potential combinations of the strategies might be worthwhile.

Finally, the research area that can be contributing to future work is the application and integration of Data Mining techniques. This gives the system the ability to suggest to the user of the mobile device extra answers to his queries, based on the patterns of previous queries. For example, when a mobile user formulates a query to find the nearest theatre, the system would automatically suggest to him the nearest restaurant to that theatre. Some investigations should be carried out in order to determine the factors that should be included in the processor to make it handle Query processing and Data Mining.

References

[Abd05a] Abd Rahman S., Bhalla S., and Hashimoto T., Query-By-Object Interface for Information Requirement Elicitation in M-Commerce. Proceedings of the Seventh IEEE International Conference on E-Commerce Technology, CEC, 2005.

[Abd05b] Abd Rahman S., Bhalla S., and Hashimoto T., Query-By-Object Interface for Dynamic Access and Information Requirement Elicitation, Proceedings of the Fourth International Conference on Mobile Business, 2005.

[Abd05c] Abd Rahman S., and Bhalla S., Spatial QBE Interface for Web GIS, Proceedings of the Fifth International Conference on Computer and Information Technology, CIT, 2005

[Abd05d] Abd Rahman S., and Bhalla S., Supporting Spatial Data Queries for Mobile Services, Proceedings of the 2005 IEEE International Conference on Web Intelligence, WI, 2005.

[Abd06] Abd Rahman S., and Bhalla S., A Mobile Device User Level Interface for Dynamic Access to Spatial Data, Proceedings of the Sixth IEEE International Conference on Computer and Information Technology, CIT, 2006.

[Abr74] Abrial J. R., Data Semantics, In J.W. Klimbie & K.L. Koffeman (Eds.), Data Base Management (pp. 1-59). North-Holland & American Elsevier Publishing Companies.

[Afe98] Afework A., Beynon M. D., Bustamante F., Demarzo A., Ferreira R., Miller R., Silberman M., Saltz J., Sussman A., and Tsang H., Digital

Dynamic Telepathology – The Virtual Microscope, In AMIA 98, American Medical Informatics Association, November 1998.

[All83] Allen J. F., Maintaining knowledge about Temporal Intervals, Communications of the ACM, Vol. 26, n°11, pp 832-843.

[And00] Andrienko N., Andrienko G., and Gatalsky P., Visualization of Spatio-Temporal Information in the Internet, In Proceedings of the 11th International Workshop on Database and Expert Systems Applications (DEXA'00), IEEE Press, 2000.

[And01] Andrade H., Kurc T., Sussman A., and Saltz J., Efficient Execution of Multiple Query Workloads in Data Analysis Applications. Proceedings of SC2001, Denver, USA.

[And02] Andrienko N., and Andrienko G., Interactive Visual Tools for Spatial Mutlicriteria Decision Making, In M. De Marsico, S. Levialdi, E. Panizzi (Eds.). In Proceedings of the Working Conference on Advanced Visual Interfaces, AVI 2002, Trento, Italy, May 22-24, 2002, ACM press, 2002, pp.129-132.

[And02a] Andrade H., Kurc T., Sussman A., Borovikov E., and Saltz J., On Cache Replacement Policies for Servicing Mixed Data Intensive Query Workloads, Proceedings of the 2nd Workshop on Caching, Coherence, and Consistency, held in conjunction with the 16th ACM International Conference on Supercomputing, 2002.

[And02b] Andrade H., Kurc T., Sussman A., and Saltz J., Scheduling Multiple Data Visualization Query Workloads on a Shared Memory Machine, Proceedings of the 2002 International Parallel and Distributed Processing 2002.

[And02c] Andrade H., Kurc T., Sussman A., and Saltz J., Multiple Query Optimization for Data Analysis Applications on Clusters of SMPs. In

Proceedings of the 2nd International Symposium on Cluster Computing and the Grid, 2002.

[And02d] Andrade H., Kurc T., Sussman A., and Saltz J., Active Proxy-G: Optimizing the Query Execution Process in the Grid, Proceedings of the 2002 ACM/IEEE Supercomputing Conference, 2002.

[And02e] Andrade H., Kurc T., Sussman A., and Saltz J., Processing Large-Scale Multi-dimensional Data in Parallel and Distributed Environments, Parallel Computing, 28(5), 827-859, 2002.

[And03] Andrade H., Aryangat S., Kurc T., Slatz J., and Sussman A., Efficient Execution of Multi-Query Data Analysis Batches Using Compiler Optimization Strategies, Proceedings of the 16th International Workshop on Languages and Compilers for Parallel Computing, LCPC, 2003.

[And03a] Andrienko N., and Andrienko G., Tools for Visual Comparison of Spatial Development Scenarios, In Proceedings of the Seventh International Conference on Information Visualization (IV'03), IEEE Press, 2003.

[And03b] Andrienko N., and Andrienko G., Informed Spatial Decisions through Coordinated Views, Information Visualization, 2(4), 2003, pp.270-285.

[And04] Andrienko N., and Andrienko G., Interactive Visual Tools to Explore Spatio-Temporal Variation, In Proceedings of ACM AVI'04, Advanced Visual Interfaces, May 25-28, 2004. Gallipoly (LE), Italy.

[And06] Andrade H., Kurc T., Sussman A., and Beomseok N., Data Management and Query – Multiple Range Query Optimization with Distributed Cache Indexing, SIGMOD Conference. 2006.

[And07] Andrienko G., Andrienko N., and Wrobel S., Visual Analytics tools for Analysis of Movement Data, In Proceedings of ACM SIGKDD Explorations Newsletter, Vol. (9) 2, ACM.

[And99] Andrienko N., and Andrienko G., Interactive Maps for Visual Data Exploration, International Journal of Geographical Information Science, Vol. 13 (4), 1999, pp.355-374.

[Art96] Artale A., Franconi E., Guarino N., and Pazzi L., Part Whole Relations in Object Centered Systems: An Overview, Data and Knowledge Engineering (DKE) Journal, North Holland, Elsevier, 1996, pp. 347-383.

[Auf95] Aufaure M. A., A High-Level Interface Language for GIS, Journal of Visual Languages and Computing, Academic Press, Vol. 6, n°2, pp 167-182.

[Bee06] Beeharee A., and Steed A., A Natural Wayfinding Exploiting Photos in Pedestrian Navigation Systems, In Proceedings on Human-computer Interaction with Mobile Devices and Services, MobileHCI'06.

[Bee07] Beeharee A., and Steed A., Exploiting Real World Knowledge in Ubiquitous Applications, Personal and Ubiquitous Computing.

[Bet00] Bettini C., Wang X., and Jajodia S., Temporal Semantic Assumptions and their Use in Databases, Proceedings of the IEEE Transactions on Knowledge and Data Engineering, 2000.

[Bet97] Bettini C., Wang X., and Jajodia S., An Architecture for Supporting Interoperability among the Temporal Databases, Temporal Databases, Dastuhl, 1997, pp. 36-55.

[Bla00] Blaser A., and Egenhofer M., A Visual Tool for querying geographic databases, in AVI 2000, ACM Press, Palermo, Italy, 211-216, 2000.

[Blu03] Blueman A., Elementary Statistics, second edition, McGraw Hill, 2003.

[Bon00] Bonhomme C., Trepied C., and Aufaure M. A., Metaphors for Visual Querying Spatio-Temporal Databases, In: Advances in Visual Information Systems, edited by R. Laurini, Proceedings of the 4th International Conference on Visual Information Systems. Springer Verlag, Lecture Notes in Computer Science, pp. 140-153.

[Bon01] Bonhomme C., and Aufaure M. A., Tests psycho-cognitifs de metaphors visuelles pour un language d'interrogation de Systemes d'Information Geographique, Revue d'Interaction Homme-Machine (RIHM), Vol. 3 – n° 2.

[Bon02] Bonhomme C., and Aufaure M. A., Mixing Icons, Geometric Shapes and Temporal Axis to Propose a Visual Tool for Querying Spatio-Temporal Databases, Advanced Visual Interfaces (AVI'2002), Trento, Italy.

[Bon99] Bonhomme C., Trepied C., Aufaure M. A. and Laurini R., A Visual Language for Querying Spatio-Temporal Databases, In Proceedings of ACM GIS'99, 7th ACM Symposium on Advances in Geographic Information Systems, November 5-6, 1999. Kansas City, USA.

[Bri02] Brinkhoff T., A Framework for Generating Network-Based Moving Objects. GeoInformatica, 6(2).

[Che02] Chen J., DeWitt D. J., and Naughton J.F., Design and Evaluation of Alternative Selection Placement Strategies in Optimizing Continuous Queries. Proceedings of ICDE, San Jose, CA.

[Cle93] Clementini E., De Felice P., and Oosterom P., A small set of formal topological relationships for end-user interaction, Proceedings of

Advances in Spatial Databases, 3rd International Symposium, Springer-Verlag, Singapore, 1993, 277-295.

[Cli87] Clifford J., and Croker A., The Historical relational data model (HRDM) and algebra based on Lifespans, In Proceedings of the 3rd IEEE International Conference on Data Engineering, 1987, pp. 528-537.

[Djo96] Djordjevic-Kajan S., Functions and contents of digital maps in process of building GIS in Serbia PTT. Proceedings in GIS/LIS, Budapest, Hungary, June 10-14, 1996, GIS/LIS Editions, 82-91.

[Ege97] Egenhofer M., Query processing in special query by sketch, Journal of Visual Languages and Computing, Vol. 8(4):403-424, 1997.

[Elm05] Elmongui H., Mokbel M., and Aref W., Spatio-temporal Histograms. Proceedings of SSTD, 2005.

[Elm06] Elmongui H., Ouzzani M., and Aref W., Challenges in Spatio-temporal Stream Query Optimization. Proceedings of MobiDE 2006, Chicago.

[Els06a] Elsidani Elariss H., Khaddaj S., and Haraty R., Towards a New Visual Query Language for GIS. IASTED Databases and Applications 2006, 195-202, Austria 2006.

[Els06b] Elsidani Elariss H., Khaddaj S., and Haraty R., An Evaluation of a Visual Query Language for Information Systems. ICEIS (5) 2006, 51-58, Paphos 2006.

[ESR] www.esriuk.com, last visited Nov. 2008.

[Fid07] Fidel R., Scholl H., Liu S., and Unsworth K., Mobile Government Fieldwork: A Preliminary Study of Technological, Organizational, and Social Challenges, Proceedings of the 8th annual International

Conference on Digital Government Research Conference: Bridging Disciplines and Domains, DGO'07, May 2007.

[Gia07] Gianotti F., Nanni M., Pinelli F., and Pedreschi D., Trajectory Pattern Mining, In Proceedings of the 13th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD'07.

[Gus06] Gustafsson A., Bichard J., Brunnberg L., Juhlin O., and Combetto M., Believable Environments: Generating Interactive Storytelling in Vast Location-based Pervasive Games, Proceedings of the2006 ACM SIGCHI International Conference on Advances in Computer Entertainment Technology, ACE'06.

[Gut06] Guting H., De Almeida T., and Ding Z., Modelling and Querying Moving Objects in Networks, VLDB Journal – The International Journal on Very Large Databases, Vol. (15) 2.

[Hor00] Hornsby K., and Egenhofer M., Identity based change: a foundation for spatio temporal knowledge representation, International Journal of Geographical Information Science, 14(3), 2000.

[ISOI96] ISO/IEC JTC1/SC21/N10441, SQL Multimedia and Application Packages, Part 3: Spatial, p.172.

[Joh98] Johnson P., Usability and Mobility; Interactions on the Move. EPSRC workshop on Healthcare Informatics, Abbingdon.

[Kan94] Kang M., Dietz H., and Bhargava B., Multiple-Query Optimization at Algorithm-Level. Proceedings of SSDI 1994. Data & Knowledge Engineering 14, 57-75.

[Kha04] Khaddaj S., and Horgan G., The Evaluation of Software Quality Factors in Very Large Information Systems, Electronic Journal of Information Systems Evaluation, Volume 7 Issue 1, 2004.

[Kim05] Kim P., Gargi U., and Jain R., Event-Based Multi-Media Chronicling Systems, CARPE, 2005, Singapore.

[Kim07] Kim S., Diverdi S., Chang J., Kang T., Iltis R., and Hollerer T., Implicit 3D Modelling and Tracking for Anywhere Augmentation, VRST 2007, Newport Beach, California, November 5-7, 2007.

[Kol99] Kollios G., Gunopulos D., and Tsotras V.J., On Indexing Mobile Objects, Proceedings of PODS.

[Lad05] Ladd A., Bekris K., Rudys A., Kavraki L., and Wallach D., Robotics-Based Location Sensing Using Wireless Ethernet, Proceedings of the 8th ACM International Conference on Mobile Computing and Networking, MOBICOM, Sep. 2002, Atlanta, GA.

[Lan92] Langran G., Time in Geographic Information Systems, London, Taylor and Francis 1992.

[Lau03] Laurini P., Paolino L., Sebillo M., Tortora G., and Vitiello G., Phenomena – A Visual Query Language for Continuous Fields, In Proceedings of ACMGIS 2003 on Association for Computing Machinery, New Orleans, Louisiana, USA 2003.

[Lau07] Laurini R., and Servigne S., Visual Access to City Websites: A challenge for PDA's GUI, Journal of Visual Languages and Computing, 18(3), 339-355, 2007.

[Laz02] Lazaridis K., Porkaew K., and Mehrotra S., Dynamic Queries over Mobile Objects, In EDBT, 2002.

[Laz04] Lazysoft Sentences. Retrieved January 5. 2004: www.lazysoft.com/technology_sentences.htm

[Lba97] Lbath A., Aufrane-Portier M. A., and Laurini R., Using a Visual Language for the Design and Query in GIS Customization, VISUAL'97, 2nd International Conference on Visual Information Systems, San Diego, USA.

[Li05] Li H., Lu H., Huang B., and Huang Z., Two Ellipse-Based Pruning Methods for Group Nearest Neighbour Queries, ACM-GIS, 2005.

[Liu03] Liu X., and Ferhatosmanoglu H. Efficient k-NN Search on Streaming Data Series, SSTD, 2003.

[Lud06] Ludford P., Franlowski D., Reily K., WAilms K., and Terveen L., Because I carry my Cell Phone anyway: Functional Location-based Reminder Applications, Proceedings of the SIGCHI Conference on Human Factors in Computing, CHI'06.

[Mar99] Marcus A., Globalization of user-interface design for the Web, Proceedings 5th Conference on Human Factors and the Web: The Future of Web Applications, Gaithersburg, Maryland, USA, 1999.

[Mcg80] McGregor D. R., and Malone R. J., the FACT Database System, Proceedings of Symposium on Database Systems.

[Mok03] Mokbel M.F., Aref W.G., Hambrush S.E., and Prabhakar S., Towards Scalable Location-Aware Services: Requirements and Reseach Issues. In Proceedings of the ACM Symposium on Advances in Geographical Information Systems, ACM GIS.

[Mok04a] Mokbel M.F., Xiong X., Aref W.G., Hambrush S.E., and Prabhakar S., Hammad M., , PLACE: A Query Processor for Handling Real-Time Spatio-temporal Data Streams. In Proceedings of the VLDB.

[Mok04b] Mokbel M.F., Xiong X., and Aref W.G.. SINA: Scalable Incremental Processing of Continuous Queries In Spatio-temporal Databases. In Proceedings of the SIGMOD.

[Mok04c] Mokbel M.F., Continuous Query Processing in Spatio-temporal Databases. In Proceedings of the ICDE/EDBT PhD Workshop.

[Mok05a] Mokbel M.F., Xiong X., Hammad M., and Aref W.G., Continuous Query Processing of Spatio-temporal Data Streams in PLACE. GeoInformatica, 9(4), 343-365, 2005.

[Mok05b] Mokbel M.F., and Aref W.G., GPAC: Generic and Progressive Processing of Mobile Queries over Mobile Data. In Proceedings of the MDM, Aya Napa, Cyprus.

[Mor02] Morris A. J., Abdelmoty A. I., Tudhope D. S., and ElGeresy B. A., Design and Implementation of a Visual Query Language for Large Spatial Databases, Proceedings of the Sixth International Conference on Information Visualisation (IV'02).

[Mor04] A. J. Morris A. J., A. I. Abdelmoty A. I., D. S. Tudhope D. S., and B. A. ElGeresy B. A., A Filter-flow Visual Query Language and Interface for Spatial Databases., GeoInformatica Vol. 8(2), p. 107-141.

[Mou00] Mountrakis G., Agouris P., and Stefanidis A., Navigating through Hierarchical Change Propagation in Spatio Temporal Queries, Proceedings of the IEEE International Conference on Data Engineering, 2000.

[Mou05] Mountrakis G., Agouris P., and Stefanidis A., Similarity Learning in GIS: An Overview of Definitions, Prerequisites and Challenges, Spatial Databases 2005 pp. 294 - 321.

[Mur00] Murray N. S., Paton N. W., and Goble C. A., Kaleidoquery: A Flow-based Visual Language and its evaluation, Journal of Visual Query Languages and Computing Vol. 11 No. 2, 2000.

[Mur98a] Murray N. S., Paton N. W., and Goble C. A., Kaleidoquery: A Visual Query Language for Object Databases, In Proceedings of Advanced Visual Interfaces, L'Aquila, Italy, 25-27, May 1998.

[Mur98b] Murray N. S., Paton N. W., and Goble C. A., Kaleidoscape: 3D Environment for Querying ODMG Compliant Databases. In Yannis Ioannidis and Wolfgang Klas, editors, Proceedings of Visual Databases 4, pp. 85-101, Chapman & Hall, London, 1998.

[Mur98c] Murray N. S., Paton N. W., Paton, and Goble C. A., A Framework for Describing Visual Interfaces to Databases, Journal of Visual Languages and Computing, vol. 9(4), pp.429-256, August 1998.

[Nur06] Nurminen A., M-LOMA: A Mobile 3d City Map, Proceedings of the Eleventh International Conference on 3d Web Technology, Web3D, 2006.

[Nus04] Nusser S., Goodchild M., Clark K., and Miller L., Geospatial Information in Complex Mobile Field Settings, Proceedings of the 2004 Annual Conference on Digital Government Research, DGO, May 2004.

[Pan02] Pang Y., Multimodal McDrive System, Master Thesis, Delft University of Technology, 2002.

[Pao03] Paolino L., Pittarello F., Sebillo M., Tortora G., and Vitiello G., WebMGISQL – A 3D Visual Environment for GIS Querying. In Proceedings of International Conference on Visual Language and Computing (VLC 2003), Miami, Florida, USA. 2003. pp. 294-299.

[Pao04] Paolino L., and Laurini R., Dealing with Geographic Continuous Fields – the Way to a Visual GIS Environment, In Proceedings of ACM AVI'04, Advanced Visual Interfaces. May 25-28, 2004. Gallipoly (LE), Italy.

[Pao07] Paolino L., Del Fatto V., and Pitarello F., A Usability-Driven Approach to the Development of a 3D Web-GIS Environment, Journal of Visual Languages and Computing, 18(3), 280-314, 2007.

[Pap03] Papadias D., Zhang J., Mamoulis N., and Tao Y., Query Processing in Spatial Network Databases, VLDB, 2003.

[Pap04] Papadias D., Shen Q., Tao Y., and Mouratidis K., Group Nearest Neighbour Queries, ICDE, 2004.

[Pap05] Papadias D., Tao Y., Mouratidis K., and Hui C. K., Aggregate Nearest Neighbour Queries in Spatial Databases, TODS, 30(2), 2005.

[Par88] Park J., Segev A., Using Common Subexpressions to Optimize Multiple Queries. In Proceedings of the Conference on Data Engineering 1988.

[Pre05] Pressman, Software Engineering. Sixth Edition, Mc Graw Hill, 2005.

[Ras06] Rashid O., Mullins I., Coulton P., and Edwards R., Extending Cyberspace: Location Based Games Using Cellular Phones, Computers in Entertainment, CIE Vol. (4) 1.

[Rei81] Reisner P., Formal Grammar and Human Factors Design of an Interactive Graphics System. IEEE Transactions on Software Engineering, SE-7(2), 229-240

[Rep06] Repenning A., and Ioannidou A., Mobility Agents: Guiding and Tracking Public Transportation Users, Proceedings of the Working Conference on Advanced Visual Interfaces, AVI'06.

[Rot06] Roth J., Detecting Identifiable Areas in Mobile Environments, Proceedings of the 2006 ACM Symposium on Applied Computing, SAC'06.

[Sch06] Schultz C., Guesgen H., and Amor R., Computer Human Interaction Issues When Integrating Qualitative Spatial Reasoning into Geographic Information Systems, CHINZ, 2006

[Sel88] Sellis T., Multiple-Query Optimization. ACM Transactions on Database systems, 13(1), 1988.

[Sha98] Shahar Y., and Cheng C., Model Based Visualization of Temporal Abstraction, Fifth International Workshop on Temporal Representation and Reasoning (TIME 1998), Sanibel Island, Florida, 1998, pp. 11-20.

[Shn91] Schneiderman Ben, Visual user interfaces for information exploration. In Proceedings of the 54th Annual Meeting of the American Society for Information Science, pages 379-384, Medford, NJ, 1991, Learned Information Incorporation.

[Shn93] Young D., and Schneiderman B., A Graphical Filter/Flow Representation of Boolean Queries: A Prototype Implementation and Evaluation. Journal of the American Society for Information Science, 44(6), pages 379-384, 1993.

[Smi04] Smith M. N. Enhancing Database Interface Support for Link Analysis., PhD Thesis, Birkbeck College, University of London, Submitted November 2003.

[Smi05] Smith M., King. P., A Database Interface for Link Analysis. Journal of Database Management, Idea Group Publishing, Vol. 16, n°1, pp 60-74. Jan-March 2005.

[Sno95] Snodgrass R. (Ed.). The TSQL2 Temporal Query Language, Kluwer Academic Publishers.

[Son01] Song Z., and Roussopoulos N., K-Nearest Neighbour Search for Moving Query Point, In SSTD, 2001.

[Sto00] Stojanovic Z., Djordjevic-Kajan Slobodanka, and Stojanovic D., Visual Query and Analysis Tool of the Object-Relational GIS Framework, In Proceedings of ACM AVI'00, Advanced Visual Interfaces, CIKM 2000, McLean, VA USA.

[Sto98] Stojanovic Z., Djordjevic-Kajan S., and Stojanovic D., Query language in Telecom GIS, GISPlaNET'98 Conference, 7-11, September 1998, Lisbon, Portugal.

[Sto99] Stoimenov L., Stoimenov A. Mitrovic, Mitrovic D. and Djordjevic-Kajan S., Bridging objects and relations: a Mediator for an OO front-end to RDBMSs, Information and software technology 41(2), pp.57-66.

[Sto03a] Stojanovic Z., Dahanayake A. N. W., and Sol H., Methodology Evaluation Framework for Component-Based System Development, Journal of Database Management, 14(1), 1-26, 2003.

[Sto03b] Stojanovic Z., Dahanayake A. N. W., and Sol H., Methodology Evaluation Framework for Component-Based System Development, Advanced Topics in Database Research, Vol.2, 213-246, 2003.

[Sto06] Stojanovic D., Book Reviews: Web Service-Oriented Project Development from Different Perspectives, IEEE Distributed Systems Online 7(2): 2006.

[Tao02] Tao Y., Papadias D., and Shen Q., Continuous Nearest Neighbor Search,, In VLDB, 2002.

[Tez06] Tezuka T., Kurashima T., and Tanaka K., Toward Integration of Web Search with a Geographic Information System, Proceedings of the 15th International Conference on World Wide Web, WWW'06.

[Xio04] Xiong X., Mokbel M.F., Aref W.G., Hambrush S.E., and Prabhakar S., Scalable Spatio-temporal Continuous Query Processing for Location-Aware Services. In Proceedings of the International Conference on Scientific and Statistical Database Management, SSDBM.

[Xio05] Xiong X., Mokbel M.F., and Aref W.G., Scalable Processing of Continuous K-Nearest Neighbor Queries in Spatio-temporal Databases, In ICDE, 2005.

[Yiu05] Yiu M. L., Mamoulis N., and Papadias D., Aggregate Nearest Neighbour Queries in Road Networks, TKDE, 17(6), 2005.

[Yiu06] Yiu M. L., Mamoulis N., Papadias D., and Tao Y., Reverse Nearest Neighbors in Large Graphs, IEEE Transactions on Knowledge and Data Engineering, TKDE, 18(4), 540 - 553, 2006.

[Yu05] Yu X., Pu K. Q., and Koudas N., Monitoring K-Nearest Neighbour Queries Over Moving Objects, ICDE, 2005.

[Yue05] Yue W., Mu S., Wang H., and Wang G., TGH: A Case Study of Designing Natural Interaction for Mobile Guide Systems, MobileHCI, Austria 2005.