Enhancing Database Interface Support for Link Analysis

Mathew N. Smith

A thesis submitted in fulfilment of the requirements for the degree
of Doctor of Philosophy in the University of London.

Submitted November 2003
Birkbeck College

Abstract

Link Analysis (LA) is a visual data analysis technique originally developed for analysing crime related data. The technique enables the user to gain a better understanding of the connections between objects of interest in the problem domain by displaying the connections in a form of network diagram referred to as a link chart. A link chart is often altered during its lifetime, as part of the exploratory nature of the knowledge discovery process, to reflect new information, and to increase the level of comprehensibility.

To provide the necessary flexibility for accessing and manipulating large volumes of data the data collected in an investigation is often stored in a database. Permitting link charts to be constructed from this data is of great value, as LA is not adequately supported by the database query systems currently available. This is because the record-based data models they often use are inappropriate for modelling connections between objects, the query languages they typically provide can only retrieve connections between objects if the way connections may be derived are specified in the query, and their data visualisation facilities generally do not allow the results of a number of queries to be integrated and edited.

This thesis concerns the Exploratory Database View Constructor (EDVC), an experimental visual database interface for supporting LA. The results obtained from a user evaluation of EDVC indicate that the system may be used by individuals with no experience of interacting directly with a database management system (DBMS). This is a consequence of the style of interaction supported, which allows the data stored to be browsed without having to possess explicit knowledge of the database schema. Such knowledge is typically a prerequisite for using a database query language and can prevent productive use of such a language by an inexperienced DBMS user.

Table of Contents

Acknowledgements

The research documented in this thesis would not have been possible without the guidance and encouragement received from a number of people, of whom there are too many to mention in a restricted space. Thanks go to the staff and students at the School of Computer Science & Information Systems for their continued support and assistance, and for making the period that the research was conducted both productive and enjoyable. Particular gratitude is extended to be my supervisor Professor Peter King, a constant source of direction and useful and insightful comments throughout my studies.

Others whose contributions made this research possible include the Engineering and Physical Sciences Research Council, and IBM, who provided funding; Lazysoft Limited and Xanalys Limited for the use of their software products for research and evaluation purposes; and those volunteers who took part in the user evaluation, which allowed the techniques developed to be validated in a realistic manner. And last, but certainly by know means least, to my family. I thank you for your encouragement throughout this and every other period; it is always appreciated.

Chapter 1 - Introduction

1.1 Introduction

Data is often best comprehended when presented as a network if the connections between the objects of interest in the problem domain are of as great an importance, if not more so, than the information known about the objects. For example, if we have the information that David is a friend of Andrea, who is the sister of Paul, who frequents the King George public house, as does David, Fiona and Edward, who works at Daltons bank, where he has a colleague called Christine, who is married to Brian, who is the brother of Angela, who is Fiona's friend, then this information may be presented diagrammatically as in figure 1.1.1.

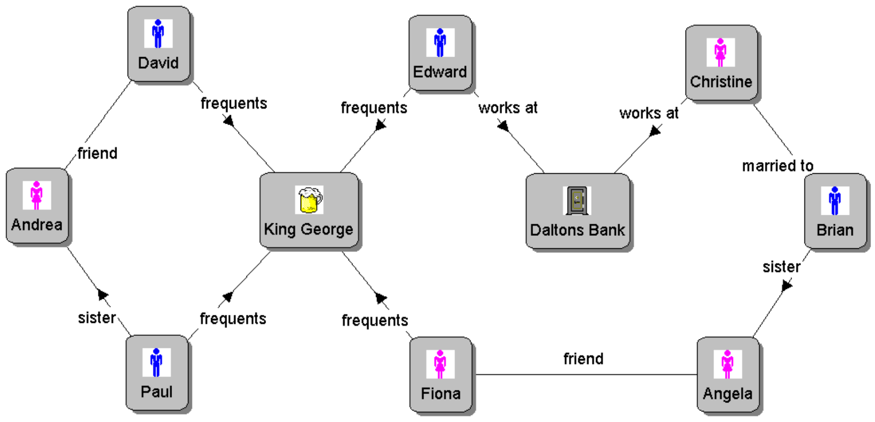


Figure 1.1.1. The connections between a set of people displayed as a network.

Presenting such data in this manner increases its comprehensibility, providing the data set displayed is not too large, and enables the user to establish visually if objects are connected. This is important in *Link Analysis* (LA), a diagrammatic technique for analysing data relating to criminal activities, which is of wide applicability.

LA was developed in response to a need in criminal investigations to fully understand the information that had been gathered, as many important facts can be hidden due the amount of information. LA is used by law enforcement agencies and in other fields such as forensic accounting [FORE]. The fundamental approach is to visualise a relevant subset of the information using a diagram similar to that in figure 1.1.1, which is often referred to as a *link chart*. The objects and links in a link chart, which may have attribute information stored about them, are added to the chart in stages as an investigation proceeds.

Although it may be the case that only particular types of connections are of interest to an investigator, such as determining if two individuals are either linked via shared ownership of a company or through a series of financial transactions, often the nature of the connections is the information required. For this reason, displaying the paths that connect two objects, and displaying the objects connected to an object, along with adding an object to a chart, are the fundamental data analysis operations used to construct a chart [SOUTH92]. These operations are not primitive; being able to add an object to a chart and display the objects directly linked to an object would allow all such operations to be constructed incrementally.

Typically when constructing a link chart the information is obtained and added to the chart in a step-by-step manner. This is because the user is exploring the data in stages, and at each stage attempting to comprehend the information that has been gathered thus far. Therefore, constructing a link chart is necessarily an incremental process. A chart is also altered during its lifetime to increase comprehensibility. Common modifications include removing objects and links no longer deemed relevant, altering the shapes and icons used to represent objects, moving objects to better positions, and adding labelled boxes or text notes, as in figure 1.1.2.

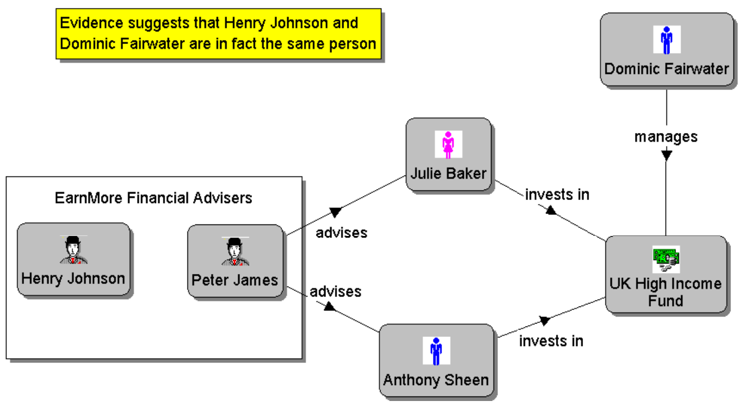


Figure 1.1.2. A link chart containing visual elements in addition to those representing objects and links. The text note displays information not yet stored in the database.

These alterations are of great practical importance in assisting understanding and are not just performed after all objects and links have been added to a chart but as the chart is being constructed.

1.2 Software Support for Link Analysis

Several commercial products provide support for the construction and analysis of link charts. The leading products are descendants of the *Intelligence Analyst Workbench* (IAW) [ICL94], a product that was produced as a result of a series of workshops with law-enforcement professionals in the UK, Hong Kong and Australia. IAW was designed to support the Anacapa [ANAC] charting method. Prior to IAW link charts were typically drawn by hand. However, the development of relatively cheap graphics workstations and the fact that much of the data used in an investigation may already be stored in database systems prompted the provision of further software support.

IAW was initially designed to assist an investigator in the drawing of charts. Facilities for saving a chart for subsequent display and further development were provided, although the system was not designed to interact with a database and the data displayed was initially entered manually, even when the data was obtained from another computer system. Similar products followed IAW, such as Watson [XANA] (described in section 2.2.1) and *The Analyst's Notebook* [I2], which have been have been developed further in response to the needs of their user communities, and are used by law enforcement agencies worldwide. These products provide a wide range of data visualisation and analysis facilities enabling several styles of chart to be developed, such as the timeline displayed in figure 1.2.1, or an association matrix such as that in figure 4.5.1.

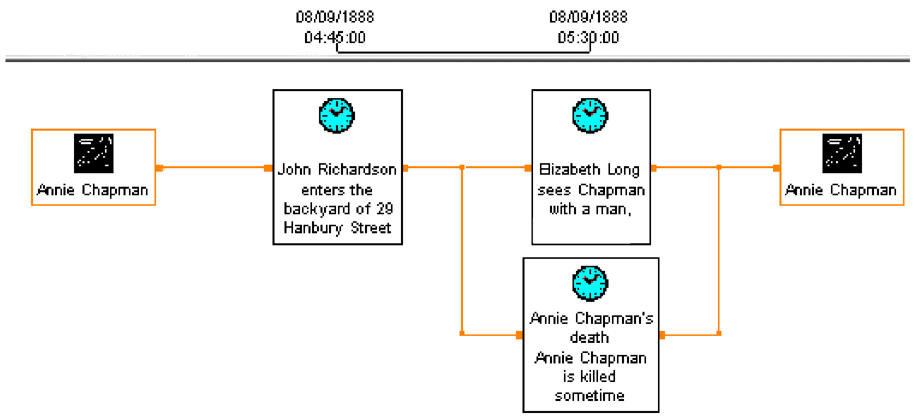

08/09/1888
04:45:00
08/09/1888
05:30:00
Annie Chapman
John Richardson enters the backyard of 29 Hanbury Street
Elizabeth Long sees Chapman with a man,
Annie Chapman's death Annie Chapman is killed sometime
Annie Chapman

Figure 1.2.1. A timeline showing the order in which a sequence of events occurred.

The data gathered in investigations, and general intelligence information, are typically stored in databases in order to manage the large amounts of data involved. Thus, LA products also now provide facilities for importing data into a chart from an already existing database, via a SQL interface. This SQL interface can limit the queries that may be formed for importing data.

Data analysis facilities for highlighting the paths in links charts that connect two objects are normally provided in tools such as Watson. However, such facilities generally only consider the data already displayed in the chart due to the limits imposed by SQL and the ancestry of such tools, which as described above were initially specialist drawing-packages with no database facilities.

The data model of a relational database differs from the model of data used in LA to find connections between objects. Thus, LA products must be configured to the schema of each database queried in order to access the data stored. This increases setup and maintenance costs, as the software must be reconfigured, usually manually, whenever the schema is modified so as to reflect the semantics of the problem domain.

This conversion between data models can also adversely affect performance. Facilities for retrieving paths connecting two objects may have to execute several SQL queries to find a single path, which can be impractical for large databases. This prevents an operator that finds the paths that connect two objects being integrated with the operators of first order logic, on which SQL is based. Thus, the set of useful queries that may be formed is limited, as the set of data analysis facilities supported in the products cannot be extended.

1.3 Database Support for Link Analysis

As described in section 1.1, a link chart is normally constructed during an investigation by performing a number of search and retrieval operations on several existing databases, by manually adding the information that is collected during the investigation itself, and by modifying the chart and its layout to increase the comprehensibility of the data displayed. If a link chart is used in a more limited way to simply display the results retrieved from a particular database, it may be considered to reflect the results returned by a number of queries that have been integrated and edited to increase their comprehensibility, where each query has been formulated in the light of the knowledge gained from the results of the preceding queries. This style of database interaction is not well supported by database query and browsing systems.

Most database query systems use a record-based data model [KENT79], such as the relational data model. Record-based data models are not well suited for modelling data such as that in figure 1.1.1. The support they provide for explicitly modelling objects, links and link attribute information is not adequate. A single record may contain information relating to several objects, or the information concerning a particular object may be contained in several separate records with no semantic connection. Additionally, whereas some links are explicitly represented others may be implicitly modelled as record attributes or as separate record types indistinguishable by the database management system (DBMS) from those modelling objects. This also affects object-oriented [CATT00] data models, which make no provision for modelling links that may have attribute information stored about them.

Query languages such as SQL and OQL assume the schema of the database queried is known and understood by the user and that a query can be expressed as a single expression. The set of queries that may be expressed in such languages is often restricted to those that may be expressed in first order logic extended with a fixpoint operator [AHO79] and operators for grouping values and for generating new values, so that the rapid query execution times required in data processing applications can be obtained. They are most suitable for expressing queries such as *retrieve the names of customers that placed orders of £10,000 or more in the last financial year*, where information that can be well defined is required.

A query language for supporting LA must allow queries that retrieve paths of an undefined length and structure to be specified, such as *display paths connecting David with employees of Daltons Bank*. Such queries cannot be expressed in a language that is based upon first order logic [AYRE95]. As illustrated in the logic formula given below, they require quantification over predicates, which is a property of a second order logic.

```
connected(x, y) ← ∃P (P(x, y) ∨ P(y, x))

path(x, y) ← connected(x, y)
path(x, y) ← ∃z (connected(x, z) ∧ path(z, y))
```

The majority of the domain experts in fields where LA is used would not have experience of directly interacting with a DBMS. Thus, a textual language that requires a strict syntax and knowledge of the database schema is not appropriate for use by these individuals. This may not be the case with a *visual database query system* [CATA97], where a query is constructed, at least partially, by directly manipulating visual elements. This higher-level description of a query may de-emphasize syntax issues and provide a higher-level of abstraction [PAPA95], simplifying query formulation even for professional programmers.

However, the majority of visual database query systems support languages based upon first order logic, even those designed to provide support for network traversal, such as GraphLog [CONS90], which is described in section 2.3.1. Several systems do provide more expressive languages that support general computation and database access, for example, Hyperlog [POUL01] and Spider [RODG97], which is described in section 2.3.4. However, these languages are either not declarative or do not explicitly support network traversal, and are not well suited for LA due to the complexity of expressing queries for retrieving the connections between objects, and their possible inefficiency for evaluating such queries for large volumes of data.

Additionally, visual database query systems tend to focus upon query specification, not the visualisation of their results [CATA97]. Those systems that do provide some visual representation of the data retrieved tend to use a visual representation conceptually similar to the data model used in the underlying DBMS, such as scrollable lists for the relational data model [SANT97]. As discussed above, these data models are often inappropriate for LA. This would also render the visual representations unsuitable, even if facilities for editing the results to increase their comprehensibility were supported, which they are not.

These query systems are also unsuitable for LA because they restrict the data they display to that which is returned by a single query, which may only be formed either by using a query language or via a browsing style of interaction [CATA97]. The lack of interaction between these two database query paradigms also restricts the level of support that these systems would provide for LA. For reasons of efficiency and usability, some LA operations may best be supported via a specialised browsing facility, such as a facility for displaying the paths that connect two objects.

However, others may require a more flexible means of expression. Displaying all males aged between 25 and 30 and between 178 and 182 centimetres tall connected to an employee of Daltons bank via one or more links representing a personal relationship, is a difficult operation to support via a specialised browsing facility, but not necessarily via a query language. As database query systems typically provide little or no interaction between a query language and a browsing query paradigm a single chart would have to be constructed using only the query facilities supported by one of these interaction paradigms, which often proves limiting.

1.4 The Aims & Objectives of the Thesis

The previous sections highlight the shortcomings of the products designed to support LA and emphasize why LA cannot be adequately supported by the existing visual database query systems. Products designed to support LA such as Watson provide powerful data visualisation facilities for editing the data displayed in order to make the data more comprehensible. However, they do not provide query languages with enough expressive power to form queries that filter the objects in an existing database on the basis of their connections to other objects, where the length and structure of these connections cannot be specified.

The use of an SQL interface to an existing database in LA products means the set of queries that may be formed to import data from the underlying database is essentially restricted to that which can be formed using first order logic. SQL was not designed for retrieving connections between the objects of interest in the problem domain modelled in a database. The system architecture of LA products also makes them less maintainable as they must be reconfigured whenever the schema of the database queried is modified.

Some visual database query systems do provide query languages that can express queries that find connections between two objects where the structure or length of a connection is not specified in the query, such as the graph rewriting language Spider. However, the data models used in these systems are not appropriate for modelling objects, links and link attribute information. Also, the data visualisation facilities provided are too limited for LA, and do not allow interaction between the browsing and query language styles of database interaction, or for the results from several queries to be integrated and displayed in a single chart and edited.

Improving the level of database interface support for LA was the aim of the research reported in this thesis. The objectives were to design, implement and evaluate an experimental system using the following design criteria:

- An appropriate data model should be used that provides constructs for modelling distinct types of objects and links that occur in the problem domain, and for directly modelling object and link attribute information. The data model should assist the construction of link charts by storing default values for the visual properties of the different types of objects and links modelled, such as the icon that is to be used by default for all instances of a particular type of object.

- An interface should be provided that allows link charts to be constructed by combining the results returned from a number of browsing operations, queries, or both. The interface should provide facilities for increasing chart comprehensibility, and allow visual elements that do not represent objects and links in the database, such as labelled boxes and text notes, to be added to a chart. The interface should also support facilities for storing link charts for subsequent display and further development.

- The set of browsing operations provided should directly support the fundamental data analysis operations of LA discussed in section 1.1, and others considered of general value in problem domains where LA may be used. The set of browsing operations supported should also be extendable so as to allow the interface to support problem domains where LA is of value but which require specialised forms of data analysis that for reasons of practicality cannot be supported in a generic database interface.

- A query language should be provided that allows the user to express queries that not only allow the objects in the database to be filtered according to their attribute information but also according to their connections to other objects. The user should be able to specify that the connections should represent certain types of relationship in the problem domain, which may include conditions that relate to the attribute value of objects and links. They should also be able to just specify that a connection should exist, which is the case if the nature of the connection is the information required.

- The system should allow users with no previous experience of interacting directly with a DBMS to productively use the system, as this would seem representative of the level of previous DBMS experience possessed by the majority of the intended user community. Such users should not have to rely solely on browsing operations directly supported in the system. They should also be able to form useful queries using the query language in an ad-hoc manner.

- The system should not have to be configured to the schema of the database queried, nor reconfigured whenever the schema of the database is modified, since incremental schema development is necessary in databases used in investigative work.

1.5 Structure of the Thesis

Chapter 2 evaluates relevant systems and research and explains why query systems designed to support related fields of interest to LA do not adequately support it. Chapters 3, 4 and 5 present the experimental system implemented, the Exploratory Database View Constructor. Chapter 3 describes the data model used by the system. Chapter 4 introduces the interface in the system for constructing link charts, and describes the browsing operations and the data visualisation facilities supported within the interface. Chapter 5 describes the query language. Chapter 6 presents a user-evaluation of the system. Chapter 7 describes the design choices made during the implementation and the lessons learnt. The thesis concludes with a summary of the contribution of the research and by outlining possible areas of future research.

Chapter 2 – Previous & Related Work

2.1 Introduction

The design criteria for the system that was to be implemented during our research, specified in section 1.4, were constructed from an evaluation of previous systems and related work. The objectives of the evaluation were to assess: the level of support provided for LA by previous work, discussed in section 2.2; which facilities supported by conventional database interfaces could be of value in the interface to be implemented, discussed in section 2.3; and if software designed for related fields of interest to LA can support the method, discussed in section 2.4.

2.2 An Evaluation of Previous Work

2.2.1 The Visual Data Analysis Tool Watson

Watson [XANA] is a visual data analysis tool where the problem domain is modelled as a set of object classes and link classes. Object classes are used to model objects of interest in the problem domain that have an independent existence and would have attribute information stored about them. The information that may be stored about each object is represented by the set of attributes defined for the object class to which the object belongs. The user must define a non-empty subset of the attributes for each object class as the key attributes. For each instance of an object class, the values of the key attributes are used to uniquely identify the object. Link classes are used to model the relationships between objects in the problem domain, may also have an associated set of attributes, and are also required to have a set of key attributes.

The data to be analysed must be held in a set of files or in a relational DBMS accessed by configuring the software to the schema of the database in the manner discussed in section 1.2. Objects and links stored in the database may be visualised using a number of charting techniques. The most commonly used is the link chart, an example of which is displayed in figure 2.2.1.1, but others are provided, such as that which allows the data stored to be visualised in a manner that emphasizes the order in which a sequence of events occurred, as in figure 1.2.1.

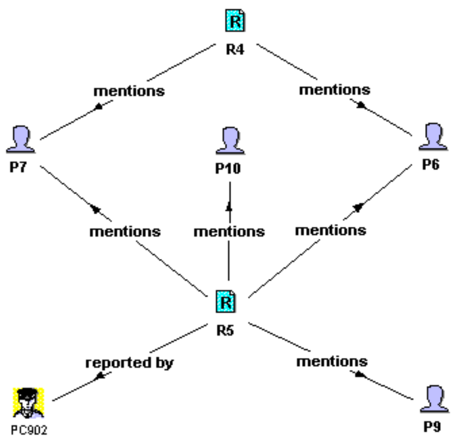
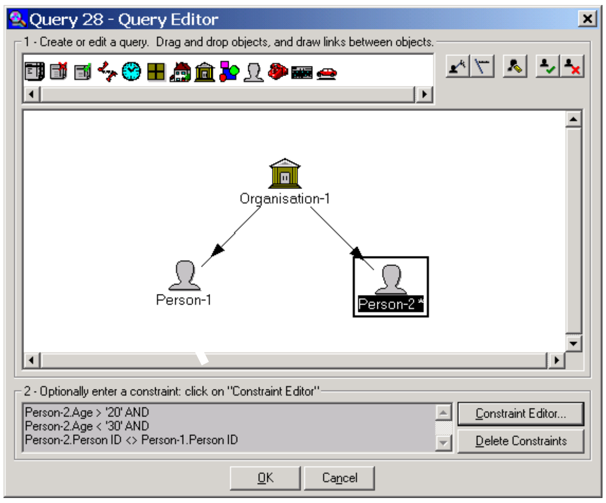


Figure 2.2.1.1. A link chart where the objects are incident reports, people and police officers.

The user begins exploring the data stored by displaying the objects and links returned by a visual query. A visual query is a pattern that filters the objects and links stored so that only those objects and links that match the pattern are displayed, and consists of a set of object constraints and link constraints. An object constraint may match all objects belonging to a specified object class or all objects belonging to a specified class that match a set of attribute value constraints. A link constraint connects two object constraints and specifies that the objects matched by the object constraints must be linked; see figure 2.2.1.2 for an example.


Query 28 - Query Editor
1 - Create or edit a query. Drag and drop objects, and draw links between objects.
Organisation-1
Person-1
Person-2 *
2 - Optionally enter a constraint: click on "Constraint Editor"
Constraint Editor...
Delete Constraints
OK
Cancel

```
Person-2.Age > '20' AND
Person-2.Age < '30' AND
Person-2.Person ID <> Person-1.Person ID
```

Figure 2.2.1.2. A visual query that displays all pairs of people linked to the same organisation where at least one person is aged between 20 and 30. The organisations and the connecting links would also be displayed.

The data displayed by a visual query may be added to and analysed using a range of data analysis facilities. Objects may be highlighted according to the number of links they have to other objects in the chart, or by specifying the object class to which the highlighted objects must belong, and if required one or more attribute value constraints that the highlighted objects must satisfy.

The connections between the objects displayed may also be analysed. The objects that are linked to a particular object may be displayed. It is also possible to ascertain if two objects are connected. If the connection is an indirect connection then the shortest connecting path(s) already displayed in the chart are highlighted.

Watson has many powerful data visualisation facilities that allow the data retrieved from the underlying data store to be effectively comprehended. However, the data visualised is stored in a database using an inappropriate data model for supporting LA. Therefore, the system has to be configured to the schema of the database queried, which increases maintenance costs and can adversely affect performance.

The query facilities provided are not sufficiently powerful enough to support LA. The query language has an expressive power equivalent to that offered by combining the selection, projection and Cartesian product operators of the relational algebra. As discussed in section 1.3, this is not powerful enough to express queries such as those in section 2.2.2. The limited range of data analysis facilities provided are chart focused and in general do not consider the data stored but only that already displayed in the chart. This can limit their usefulness.

2.2.2 The Database Programming Language Hydra

Hydra [AYRE96] is a computationally complete database programming language based upon the functional programming paradigm that uses a data model that takes a functional view of a binary-relational data model. Relationship types are represented as functions that form part of the data model. As shown in figure 2.2.2.1, functions, which are uniquely identified by their name, may either be *single-valued* or *multivalued*, and map an instance of their *domain type* to zero or more instances of their *codomain type*.

```
entity student;
entity course;
entity lecturer;

primary age           :: student  -> integer;
primary student_name  :: student  -> string;
primary tutored_by    :: student  -> lecturer;
primary takes         :: student  -> [course];

primary lecturer_name :: lecturer -> string;
primary teaches       :: lecturer -> [course];
```

Figure 2.2.2.1. A Hydra database schema modelling data relating to the set of courses taken by a set of students. Functions defined with square brackets surrounding their codomain types are deemed multivalued, those that are not are deemed single-valued.

Single-valued functions return at most one instance of their codomain type, whereas multivalued functions return zero or more instances of their codomain type.

Data may be retrieved from a database by applying functions defined in the data model to entities, which are represented by surrogates visible to the user. The data retrieved may be filtered and manipulated using the facilities provided for general computation, such as in the query given in figure 2.2.2.2.

```
count [ x | x <- ~takes Java_Programming | x.age > 30 ];
```

Figure 2.2.2.2. A query for the database whose schema is defined in figure 2.2.2.1 that retrieves the number of students taking the Java programming course whose age is greater than 30. A function whose name starts with ~ represents the inverse of the function defined in the data model with the same name excluding the tilde, such functions are automatically provided by Hydra.

Data may also be retrieved using functions provided for querying the connections between data items. Such functions are referred to as *associational primitives* [AYRE95], and are discussed below.

The associational primitive *like* may take either an entity or an entity type as a parameter and returns a list of all the entities stored that are instances of the entity type that the parameter is an instance of, or the entity type which is supplied as the parameter.

The associational primitives *from* and *to* may also take an entity or an entity type as a parameter. The function *from* returns a list of the functions defined in the data model that have the entity type that the parameter is an instance of, or the entity type supplied as a parameter, as their domain type; see figure 2.2.2.3.

```
> from Peter_Jones;

[ml.age, ml.student_name, ml.tutored_by, takes];
```

Figure 2.2.2.3. A query for the database whose schema is defined in figure 2.2.2.1 that returns a list of the functions defined in the data model that have the entity type that Peter_Jones is an instance of as their domain type. Single-valued functions are automatically prefixed by *ml.* to convert them into multivalued functions so that the list returned conforms to typing restrictions.

The function *to* returns a list containing all functions defined in the data model that have the entity type that the parameter is an instance of, or the entity type that is supplied as a parameter, as their codomain type.

The associational primitive *trail* takes an integer and two entities as parameters and returns a list of lists, where each list contained in the list returned represents a path through the database that connects the two entities and whose length is less than or equal to that of the integer; see figure 2.2.2.4.

```
> trail 2 Peter_Jones Dr_Wilson;

[[Peter_Jones,  takes,  DM01,  ~teaches,  Dr_Wilson],[Peter_Jones,  tutored_by,
Dr_Wilson]]
```

Figure 2.2.2.4. A query for the database whose schema is defined in figure 2.2.2.1 that retrieves the paths through the database of length two or less that connect the entities Peter_Jones and Dr_Wilson.

The paths through the database returned by trail are filtered so that paths containing loops or instances of a base type (system defined data types such as string or integer) are not returned. As shown in figure 2.2.2.5, the data retrieved by applying associational primitives may be filtered and manipulated using the facilities provided for general computation.

```
[ last_element path | y <- like ?student | path <- trail 2 David_Broome y ];
```

Figure 2.2.2.5. A query for the database whose schema is defined in figure 2.2.2.1 that retrieves all students connected to the entity David_Broome by a path through the database of length two or less. Entity types are referred to by preceding the name of the entity type with *?*.

Hydra offers a great deal of expressive power and can be used to form queries of great complexity that retrieve paths through a database, but nonetheless is not intended to be an end-user tool. Functional programming is a skill that can be difficult to acquire, even for experienced computer programmers [WADL98]. Thus, the scope for a user familiar with the problem domain but not familiar with functional programming and functional databases to extract meaningful information is somewhat limited.

The data model used in Hydra is too simplistic for supporting LA. Links for which attribute information may be stored cannot be directly modelled; they have to be modelled as separate entity types. This results in a break down between how data is modelled and the user's view of the problem domain and may lead to an increased level of complexity for expressing queries.

Hydra does not provide for integrating and editing the results returned by multiple queries, and is also affected by a lack of meta-query facilities. It is not possible to directly ascertain what the type of an object is when a query is being evaluated, which prevents conditions that relate to the attribute values of the objects from being specified. This proves limiting for expressing queries that filter the types of paths returned to the user.

2.2.3 The Visual Query Interface for Hydra, VisualQ

VisualQ [ABRE95] is an experimental visual query interface for Hydra that allows the user to display the associations between a subset of the entities stored in a specified database as a link chart; see figure 2.2.3.1. The browsing facilities provided in the system allow the user to add entities directly to a chart, display the information stored directly about a selected entity in a separate window, display paths containing five or less links connecting two selected entities, and display in a separate window the entities connected to a group of selected entities by paths containing three or less links.

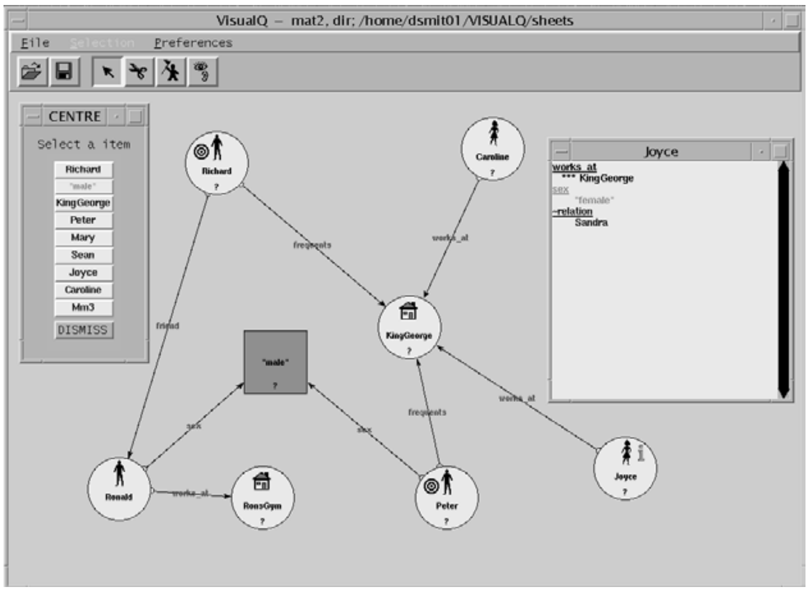


Figure 2.2.3.1. VisualQ.

A chart may be modified by removing chart elements, associating icons with those elements representing entities, and by moving elements to better positions. The charts constructed may be stored for further development at a later point in time.

VisualQ may be used effectively for comprehending the associations between the entities in a small database, but is not suitable for supporting LA over non-trivial databases. As the data model of Hydra is used, the deficiencies of this data model for supporting LA discussed in section 2.2.2 also detrimentally affect VisualQ. The system also has a restricted range of standalone browsing facilities whose usefulness is limited by the restrictions placed on the upper path length. As these facilities are not integrated with the query operators traditionally found in database query languages, such as those of the relational algebra, they may not be used to form queries such as that in figure 2.2.2.5.

The range of data visualisation facilities provided restricts the usefulness of the system for LA. Entity surrogates in practice do not correspond to meaningful pieces of information such as the names of people, which may be duplicated, but to unique identifiers which often have little meaning. Additionally, function names of are often abbreviated to comply with the syntax of Hydra. Displaying these to an end-user is inappropriate; the user could be confused or misinterpret their meaning unless they have knowledge of the database and its schema. This is inappropriate as there is no schema browsing or data dictionary facility provided.

Several users may wish to refer to the same entity using several different names and require several different concurrent views of the same database. As the label of an object in a chart is restricted to the surrogate of the corresponding entity and only a single chart may be viewed at any one instance, this is not supported within the system. The comprehensibility of a chart is also restricted by not being able to add elements other than those representing the entities and associations stored.

2.2.4 The Database Management System Sentences

Sentences [LAZY] uses the associative model of data [WILL02][1], a form of binary-relational data model where there are two basic modelling constructs: entity types and association types. Entity types in the associative model of data are similar to the entity types in a binary-relational data model except that each entity type may have an associated system-defined data

[1] This data model is essentially the same data model as used in the FACT system [MCGR80].

type such as text or number. If so, the entity type is referred to as a *value type*. Value types are equivalent to base types in the data model of Hydra, which is described in section 2.2.2.

Association types model directed associations in the problem domain. Association types may be defined from any type (including other association types) except a value type, to any type. Association types and entity types may form type hierarchies similar to those that may be defined for classes in object-oriented systems.

Each entity in a database must have an associated name. This name is given by the user and need not be unique within a database as each entity has a unique system generated identifier. Each association has a unique system generated identifier.

Sentences provides two distinct styles of query interface for analysing the data in a database, one which is based upon a browsing style of interaction, the *explorer*, which as illustrated in figure 2.2.4.1 is divided into two panes: the *schema pane* and the *data pane*. The schema pane, located on the left hand side, displays the database schema. The data pane, located on the right hand side, displays the instances of the type currently selected in the schema pane. Both the schema and data panes use a tree structure to display information.

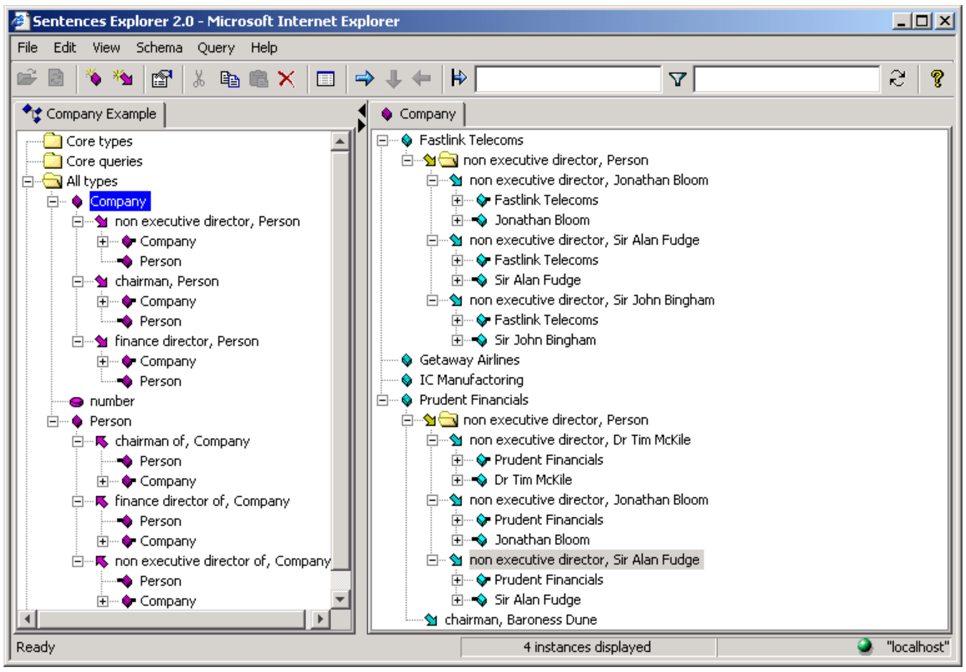
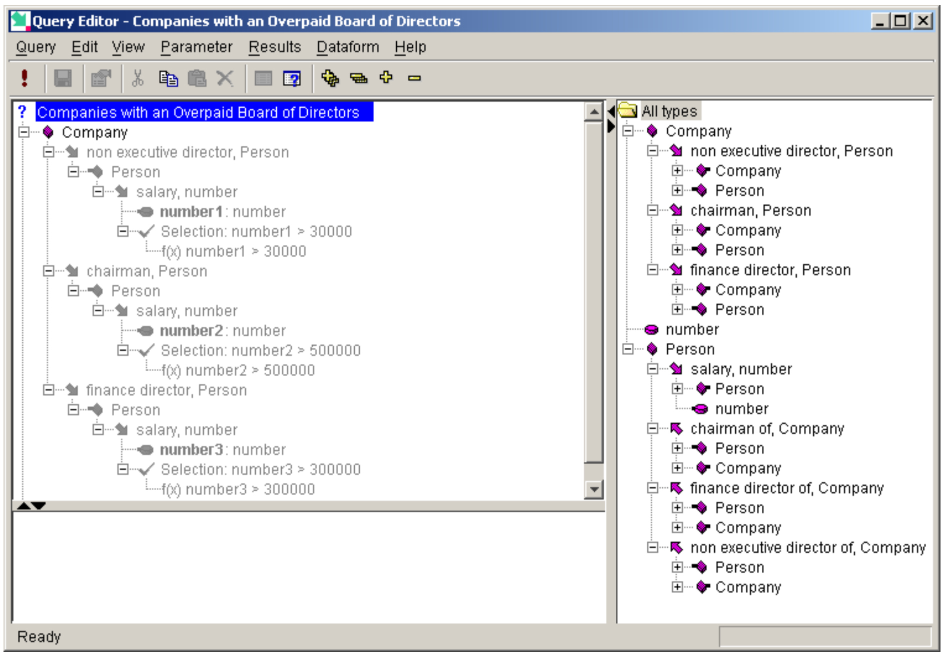


Figure 2.2.4.1. An explorer.

In the schema pane tree nodes represent entity types and association types. The root of the tree represents the database schema, its children represent entity types. Nodes that represent types that are the *source* or *target* of an association type (the types that an association type is defined from and to) will have children. Each such node will have one child for each association type that has the corresponding type as a source or target. Each such child will in turn have children representing the source and the target of the corresponding association type. This structure is recursive. The tree structure in the data pane is analogous except that the tree nodes are not entity types and association types, but instances of these types.

As shown in figure 2.2.4.2, queries are also represented using a tree structure. The root node, referred to as the *data request node*, must represent an entity, an association, an entity type or an association type. Its children may represent types whose source or target is the same as, or a supertype of, the type associated with the data request node. This is true for the children of all nodes representing elements in the database or the database schema.


Query Editor - Companies with an Overpaid Board of Directors
Query
Edit
View
Parameter
Results
Dataform
Help
Ready

```
Companies with an Overpaid Board of Directors
Company
    non executive director, Person
        Person
            salary, number
                number1: number
                Selection: number1 > 30000
                    f(x) number1 > 30000
    chairman, Person
        Person
            salary, number
                number2: number
                Selection: number2 > 500000
                    f(x) number2 > 500000
    finance director, Person
        Person
            salary, number
                number3: number
                Selection: number3 > 300000
                    f(x) number3 > 300000
```

```
All types
    Company
        non executive director, Person
            Company
            Person
        chairman, Person
            Company
            Person
        finance director, Person
            Company
            Person
    number
    Person
        salary, number
            Person
            number
        chairman of, Company
            Person
            Company
        finance director of, Company
            Person
            Company
        non executive director of, Company
            Person
            Company
```

Figure 2.2.4.2. A query that retrieves the companies having a non-executive director paid more than 30,000, a chairman paid more than 500,000, and a finance director paid more than 300,000. Sub-trees needed to specify filter conditions but not required to return instances are coloured light grey.

The instances returned by nodes representing elements in the database schema may be filtered in several ways. The most common method is to add a *selection node* with an associated filter condition as a child of such a node. As illustrated in figure 2.2.4.2, the selection node will filter the instances returned by its parent so that only instances that satisfy the filter condition are returned.

Support for querying hierarchies is provided. A node representing an association type whose target is a subtype of the source may be declared *recursive*. When a query is evaluated such associations are applied transitively until closure is achieved.

While Sentences uses an appropriate data model for supporting LA the browsing facilities, query language and data visualisation facilities supplied to the end-user are inappropriate for this purpose. The explorer interface is limited as the data displayed reflects the results of a single query that has been constructed in a piecemeal fashion. This restricts the information that may be displayed.

Additionally, the hierarchical display mechanism is not well suited for supporting LA as important information may be obscured by a single entity appearing more than once inside a single data pane. Consider the data pane displayed in figure 2.2.4.1. Jonathan Bloom and Sir Alan Fudge are both non-executive directors of Prudent Financials and Fastlink Telecoms; because these entities are duplicated within the data pane this connection may go undetected.

The query language is more expressive then those provided in LA products such as Watson (described in section 2.2.1), although not directly comparable due to the difference in data model and associated algebraic operators [WILL02]. However, queries retrieving connections where the nature of the connections is not specified in the query cannot be expressed.

2.3 The Contribution of Related Work

2.3.1 Hy$^+$ & the Visual Query Language GraphLog

Hy$^+$ [CONS94] is a visualisation system for data represented as a Hygraph, which as shown in figure 2.3.1.1, is a form of graph where nodes may contain one or more other nodes. The data visualised may be restricted by specifying a filter pattern using the visual query language GraphLog [CONS90], which has an expressive power that is a subset of that offered by first order logic extended with a transitive closure operator. As illustrated in figure 2.3.1.2, a filter pattern is expressed as one or more query graphs.

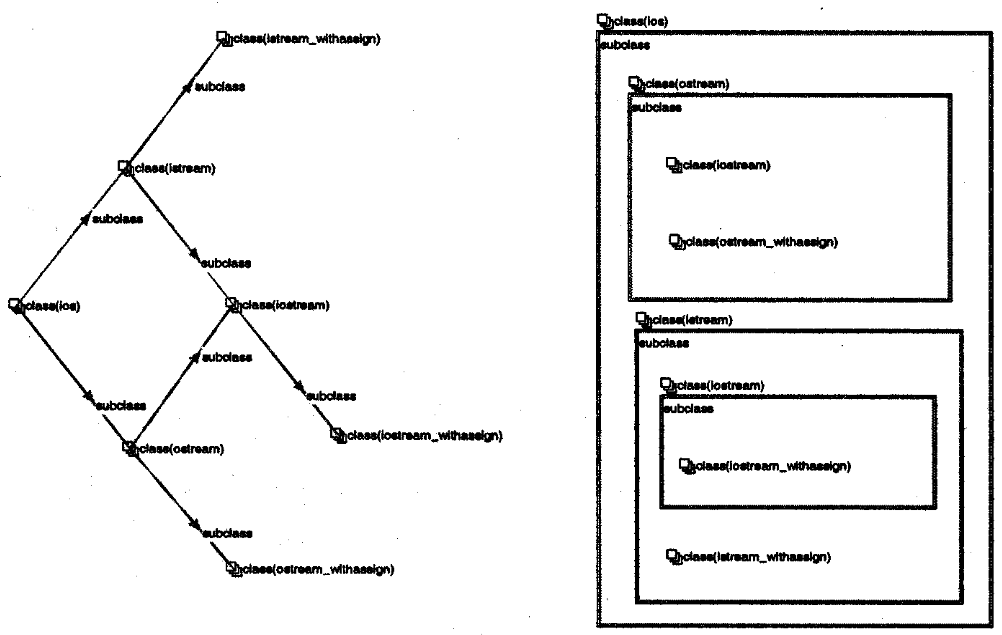


Figure 2.3.1.1. The Hygraph located on the left hand side of the figure represents the same set of problem domain relationships as the Hygraph located on the right hand side of the figure. Whereas the Hygraph located on the left represents the relationships using a set of edges the Hygraph located on the right represents the relationships using containment.

Each query graph node matches zero or more nodes in the database graph, must be connected, and labelled with either a constant or with the name of a variable. Each query graph edge must be labelled with a regular expression or a label that represents another query graph. An edge labelled with a regular expression matches the paths, if any, between the nodes in the database graph currently being considered for the start and end points of the edge that satisfy the expression. The user may select the graph displayed in Hy$^+$ as the database graph, which allows the information displayed to be filtered incrementally.

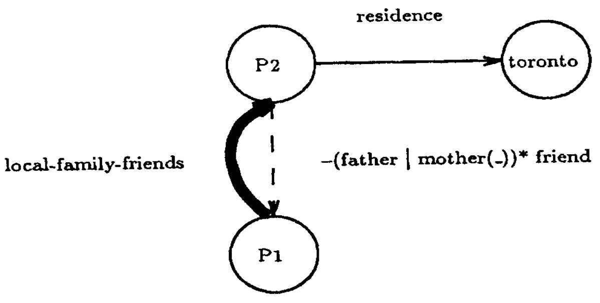


Figure 2.3.1.2. A filter pattern that retrieves people that have one or more friends resident in Toronto, or one or more ancestors having one or more friends resident in Toronto; the friends, ancestors and the connecting edges are also displayed. The black line labelled local-family-friends signifies that this query graph may be nested within another query graph by labelling an edge with local-family-friends.

While the data model used in Hy$^+$ is simple yet powerful, the inability to directly represent information about problem domain relationships makes the data model unsuitable for use in many LA application areas for the reasons discussed in the penultimate paragraph of section 2.2.2 for the data model of Hydra. Additionally, the ability to present problem domain relationships using edges or containment negatively impacts upon the effectiveness of the data visualisation facilities for comprehending the connections between graph nodes.

The expressive power of GraphLog compares favourably with many database query languages, and being able to incrementally filter the dataset displayed allows queries to be formed that cannot be expressed using a single complex query by an individual who has knowledge of the underlying structure of the data. However, queries retrieving paths through the database where how the connections may be derived are not specified in the query cannot be expressed.

The regular expressions that may be specified on query graph edges cannot just contain the closure operator. Therefore, the user must specify how potential connections may be derived, which requires a full knowledge of how the underlying data is structured and may prove impractical for inexperienced users. Additionally, little support is provided for the comparison operators typically provided in database query languages, or for combining them.

While Hy$^+$ and GraphLog are not appropriate for supporting LA, features from these tools have been included in the system implemented during the course of our research. The user may filter the information displayed in a link chart incrementally by selecting the chart as the database. Also, the query language in the system (discussed in chapter 5) allows the user to constrain the paths matched using a form of regular expression, although as discussed in section 2.3.2 such expressions are not specified in the same manner as they are in GraphLog.

2.3.2 SWYN: A Visual Representation for Regular Expressions

Although the form of regular expression (RE) provided for matching paths in GraphLog is not powerful enough to match paths of an undefined length and structure, REs can be used as the basis for pattern matching languages capable of providing the necessary expressive power. The majority of these languages are textual and there exist several issues regarding their level of usability that stem from the terse notation used and the lack of a mental model of how expressions are evaluated [BLAC01]. As part of the SWYN research project [BLAC01] several alternative yet equivalent notations for representing REs were evaluated in order to access how readily understandable they were. It is this aspect of the project that is discussed below.

Four notations were evaluated. The first, an example of which is given in figure 2.3.2.1, was a constrained subset of that conventionally used to express REs. The closure operator, which is often represented by the + symbol, was used instead of Kleene closure, typically represented by the * symbol, as this symbol may confuse novices as the null string is also matched.

```
(0|0044)1223[356][0-9]+
```

Figure 2.3.2.1. A regular expression in the first notation that matches telephone numbers.

The second notation was also textual, but not declarative; control characters such as + and | were replaced with English instructions and a strict order of evaluation that could be followed by the user was defined; see figure 2.3.2.2.

```
Find one of the following:
       a) either the sequence “0”
       b) or the sequence “0044”
followed by the sequence “1223”
followed by any one of these
characters: “3” or “5” or “6”
followed by at least one, possibly more,
of the following:
       -any one of these characters: any
       one from “0” to “9”
```

Figure 2.3.2.2. A regular expression in the second notation equivalent to that in figure 2.3.2.1.

The third notation, an example of which is given in figure 2.3.2.3, was declarative, but used graphical cues in place of control characters.

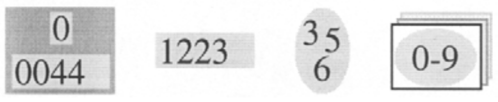


Figure 2.3.2.3. A regular expression in the third notation equivalent to that in figure 2.3.2.1.

All participants in the evaluation were given a legend of the graphical cues used in the third notation, which are given in figure 2.3.2.4.

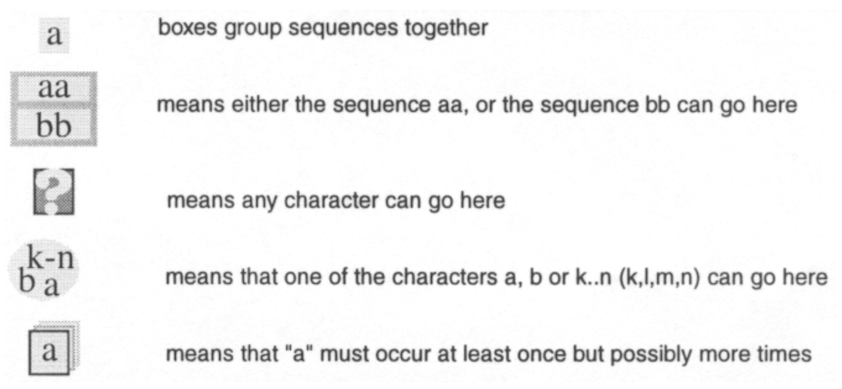

a
boxes group sequences together
aa
bb
means either the sequence aa, or the sequence bb can go here
means any character can go here
k-n
b a
means that one of the characters a, b or k..n (k,l,m,n) can go here
a
means that "a" must occur at least once but possibly more times

Figure 2.3.2.4. A legend for the graphical cues used in the third notation.

The fourth notation, an example of which is given in figure 2.3.2.5, was both graphical and procedural and similar to that used in flow charts and state transition diagrams, which are commonly used to express finite state automatons.

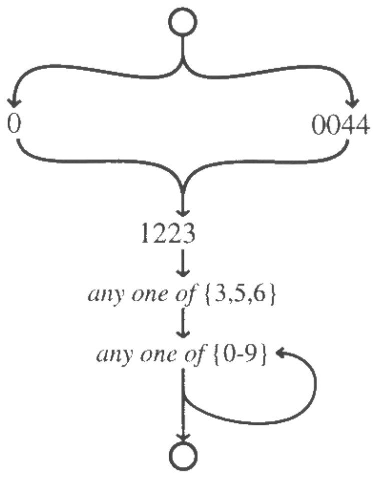


Figure 2.3.2.5. A regular expression in the fourth notation equivalent to that in figure 2.3.2.1.

A legend for the graphical cues used in the fourth notation is given in figure 2.3.2.6.

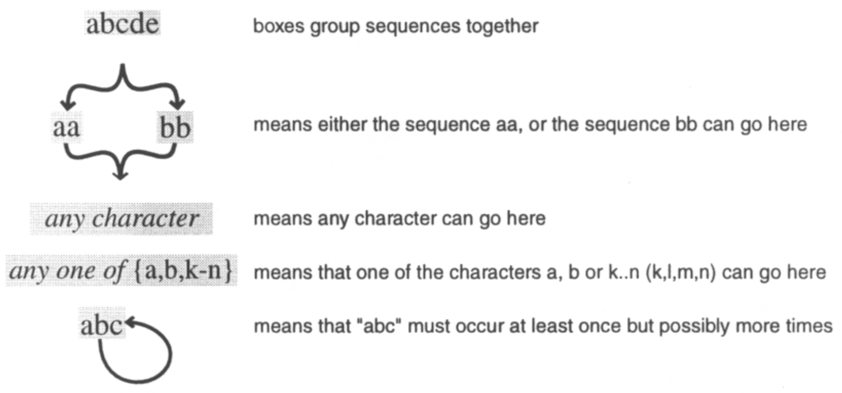

abcde
boxes group sequences together
aa
bb
means either the sequence aa, or the sequence bb can go here
any character
means any character can go here
any one of {a,b,k-n}
means that one of the characters a, b or k..n (k,l,m,n) can go here
abc
means that "abc" must occur at least once but possibly more times

Figure 2.3.2.6. A legend for the graphical cues used in the fourth notation.

39 participants took part in the evaluation, all postgraduate students studying a wide range of arts and science disciplines; none were previously familiar with REs. Each participant was provided with a booklet describing the evaluation, the four different notations and 12 tasks, which focussed upon identifying commonly encountered strings such as telephone numbers.

Each task involved studying a RE presented in one of the four notations and marking 5 text strings to indicate if they matched the expression. The amount of time taken to complete each task was noted. The tasks were divided into 6 six pairs. Distinct notations were used within each pair and the structure of the expressions were made as similar as possible so as to allow a comparison to be made between the performance of the participants on similar tasks using different notations. The assignment of notational formats to pairs and to tasks was varied as was the order in which the notations were presented.

The mean time taken to complete the tasks for each notation and the percentage of incorrect answers given are provided in table 2.3.2.1.

	Mean Time	**Incorrect Answers**
1st Notation	117.6	60%
2nd Notation	106.7	48%
3rd Notation	86.1	48%
4th Notation	86.2	38%

Table 2.3.2.1. Performance of the participants; the mean time to complete tasks is measured in seconds and the figures for incorrect answers are percentages.

An investigation comparing the performance of all participants for the first task encountered and the last six tasks was also conducted; the results are given in table 2.3.2.2.

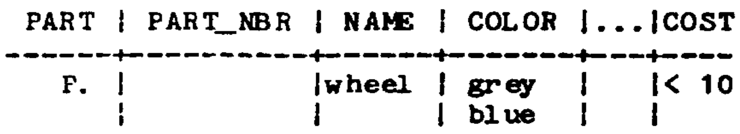
	Mean Time 1st Task	**Incorrect Answers 1st Task**	**Mean Time Last 6 Tasks**	**Incorrect Answers Last 6 Tasks**
1st Notation	198	64%	104	47%
2nd Notation	207	44%	82	37%
3rd Notation	110	25%	77	47%
4th Notation	123	9%	71	27%

Table 2.3.2.2. Performance of the participants in the first task and in the last 6 tasks; the mean time to complete tasks is measured in seconds and the figures for incorrect answers are percentages.

The results of the experiment show that the participants when using graphical notations produced significantly fewer errors and required noticeably less time to complete tasks than with the textual notations. They also illustrate that the fourth notation was initially the most accurate to use, almost the quickest notation to use initially, and proved the most usable of the four notations for the last six tasks. Thus, this notation would seem best suited for use by inexperienced and experienced users for matching paths through a database. It is for this reason that a similar notation is used for filtering such paths in the query language provided in the system implemented during the course of our research.

2.3.3 The Visual Database Language G-WHIZZ

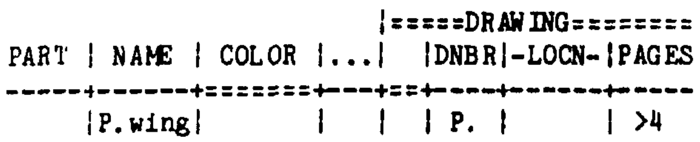
G-WHIZZ [HEIL85] is a visual language for databases using a binary-relational data model similar to that used in Hydra (described in section 2.2.2) that is based on the work of Zloof [ZLOO77]. The primary visual construct is the table. A table represents an entity type defined in the data model. The columns of a table represent the functions defined in the data model that have the associated entity type as their domain type and the rows typically represent the retrieval conditions specified by the user; see figure 2.3.3.1.



```
PART | PART_NBR | NAME  | COLOR |...|COST
-----+----------+-------+-------+---+----
  P. |          |wheel  | grey  |   |< 10
     |          |       | blue  |   |
```

Figure 2.3.3.1. A query that retrieves the parts whose name is wheel, whose cost is less than ten and whose colour is either grey or blue.

A column that represents a function whose codomain type is an entity type that is the domain type of one or more functions defined in the data model may be expanded. When such a column is expanded the columns that would appear in a table for the entity type that is the codomain type of the associated function are nested inside of the expanded column. As illustrated in figure 2.3.3.2 the nested columns are collectively labelled with the name of the column that was expanded. Support for querying hierarchies is also provided.



```
                          |=====DRAWING========
PART | NAME | COLOR |...|  |DNBR|-LOCN-|PAGES
-----+------+=======+---+==+----+------+-----
     |P.wing|       |   |  | P. |      | >4
```

Figure 2.3.3.2. A query that retrieves the name and associated drawing numbers of the parts whose name is wing and have an associated drawing of more than four pages.

The data retrieved by a query is displayed in a table whose columns correspond to the columns in the query table that were selected for inclusion in the query result. Such columns must contain a variable, such as the PART column in the query displayed in figure 2.3.3.1.

G-WHIZZ is not well suited for supporting LA. As discussed in section 2.2.2 for Hydra, the data model used is too simplistic for effectively modelling data that may be stored about problem domain relationships. The expressive power of the query language is equivalent to that offered by extending the relational algebra with a transitive closure operator, and hence not well suited for querying the connections between entities, which is also true of the tabular method of query result visualisation.

However, the form-based method used in G-WHIZZ for combining retrieval conditions allows such conditions to be combined declaratively without having to explicitly use the Boolean connectives AND and OR. This has the potential to reduce the number of incorrect queries formed as the use of these connectives can lead to errors when constructing queries [GREE90]. It is for this reason that a similar method of combining retrieval conditions has been incorporated in the query language in the system implemented during the course of our research.

2.3.4 The Visual Database Programming Language Spider

Spider [RODG97] is a computationally complete graph rewriting language for databases using a binary-relational data model similar to that described in section 2.2.2 for Hydra. A query is regarded as a transformation step that does not alter the state of the database. As shown in figure 2.3.4.1 a transformation step, is specified as a sequence of graph rewrites. Each graph rewrite consists of a pair of graphs; one for matching sub-graphs in the database, the other for specifying the structure of the graph matched after transformation.

After a transformation step has been selected each graph rewrite in the sequence defined for the step is tried in turn until one is matched against one or more sub-graphs in the database. The sub-graphs matched are then transformed using the rewrite, which does not necessarily indicate the end of computation. Further transformation steps may be applied if the rewrite introduced one or more application nodes into the graph. These specify that the corresponding transformation steps must to be applied in the order the nodes were introduced. No permanent loss of information is caused by a graph rewrite that deletes part of the database sub-graph that was matched; this only affects the query result.

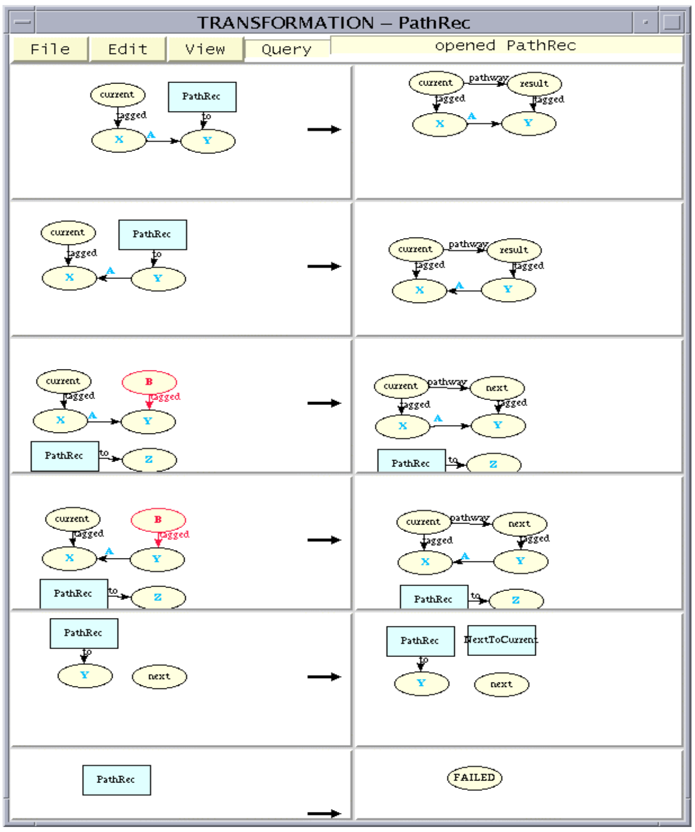


Figure 2.3.4.1. A transformation step. Each row represents a graph rewrite, the match graph is located in the in the left column, the graph specifying the structure of the graph matched after transformation in the right column. Each match graph may have an application node that represents the transformation step and is labelled with the name of the transformation step. Application nodes are rectangular.

A query that retrieves paths through a database where the length and structure of the paths to be retrieved is undefined in the query can be formed in Spider. However, forming such a query is a non-trivial process. The fundamental concept of graph rewriting is relatively simple and intuitive in principle. However, our experience with using Spider indicates that in practice the method is complex to apply, even for forming simple queries. For example, figure 2.3.4.1 displays just some of the transformations required to retrieve all the paths connecting two entities. For the reasons discussed in section 2.2.2, the complexity of applying the method increases further if information needs to be stored about relationships in the problem domain that link entities.

Spider, unlike the majority of database query languages, is not declarative. Therefore, a user cannot concentrate solely on how to specify what information should be retrieved; they must also specify how it should be retrieved. This is impractical for inexperienced database users, and may result in suboptimal query evaluation times for all classes of user. For these reasons the query language provided in the system implemented during the course of our research is declarative and only contains a relatively small number of concepts so that individuals who may have little or no experience of interacting directly with a DBMS interface can form useful queries that may be evaluated efficiently.

2.4 An Evaluation of Systems Designed for Related Fields

2.4.1 Systems for Supporting Social Network Analysis

LA is frequently used to comprehend the connections between a set of people, which is also one of the aims of social network analysis (SNA) [MOLI01, WIDM99]. Several systems support SNA [ANAL, BATA97, CYRA], the majority by visualising the network analysed in some manner [FREE00]. One of the more sophisticated of these systems is Pajek [BATA97].

Pajek is an experimental system for analysing *large* networks. The computational complexity of the analysis routines provided is generally sub-quadratic providing that the entire network can be loaded into main memory. A network may either be loaded from a text file or constructed interactively. Nodes are uniquely identified by a number but may have a label that need not be unique. Edges may either be directed or undirected and can also have an associated numeric value. There is no support for data typing.

Support for analysing the paths in a network is provided through the provision of routines for finding the shortest path(s) connecting two nodes and the *K-neighbours* of a node, which are the nodes connected by K or less edges to a particular node. The nodes that are the same number of edges away from a selected node may be considered members of a *cluster*, which is a group of nodes deemed structurally similar to one another.

Clusters may also be formed according to the number of edges connecting nodes to other nodes, or the ability to reach other nodes in the same cluster via a path through the network. This includes the ability to detect cliques, where every node has to be directly connected to every other node in the same cluster, or K-cores, where every node has to be connected to at least K other nodes in the same cluster.

The results of analysis routines may be represented as a separate network that may subsequently be displayed, or as some form of matrix. Where sensible to do so the results of one routine may be used as the input for another, thus allowing the network analysed to be filtered incrementally. Networks may also be combined using the set operations union, intersection and difference.

The network analysed, or a network produced as the result of an analysis routine, may be displayed using a number of data visualisation facilities. These include facilities for laying out a network with a minimal number of edges crossing (see figure 2.4.1.1), for rotating the network displayed, and for altering the appearance of nodes and edges, such as editing their size or colour.

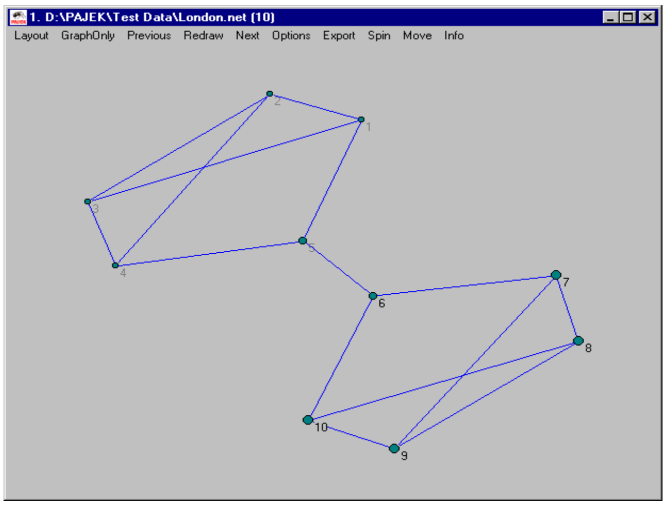


Figure 2.4.1.1. An example of a social network displayed in three-dimensional space.

Systems supporting SNA generally provide a great deal of support for analysing the structural properties of relatively small homogeneous networks. However, they are not designed for exploring the large heterogeneous networks often encountered in LA.

Systems supporting SNA typically provide little or no support for data typing; the edges in the network analysed are generally assumed to represent the same type of relationship, and the nodes have no associated type. This limits the range of queries that may be posed, and no support is provided for query languages such as those described in sections 2.2 and 2.3.

Although support is often provided for combining and visualising the results of analysis routines the range of facilities typically provided and the manner in which results are combined, limits their usefulness for LA. Additionally, the data visualisation facilities in these systems are very limited when compared to systems such as Watson (described in section 2.2.1), which can have a significant impact on the comprehensibility of the data displayed.

2.4.2 Systems for Querying Semi-Structured Data

Retrieving paths of an undefined length and structure through the data stored, possibly without any knowledge of the underlying structure of the data, is one of the characteristics of querying semi-structured data such as XML. Several systems for querying semi-structured data provide visual interfaces [CERI99, ERWI03, MUNR00]. One of the more powerful systems is Xing [ERWI03], a form-based visual language for querying and restructuring XML data sources that need not have a Document Type Definition[2], like that in figure 2.4.2.1.

```
<bib>
       <book year = “1988”>
              <title>Concrete Mathematics</title>
              <author>Graham</author>
              <author>Knuth</author>
              <author>Patashnik</author>
       </book>

       <article year = “1998”>
              <title>Linear Probing and Graphs</title>
              <author>Knuth</author>
              <journal>Algorithmica</journal>
       </article>
</bib>
```

Figure 2.4.2.1. Bibliographic data expressed as an XML document.

A query is expressed as a *document rule*. A document rule consists of an *argument pattern* and possibly a *result pattern*. The argument pattern specifies the conditions that the data retrieved should satisfy, the result pattern species how the data retrieved should be structured in the query result. If no result pattern is specified then the result pattern is deemed to have the same structure as the argument pattern, as in the query in figure 2.4.2.2.

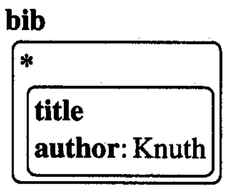


Figure 2.4.2.2. A query for a data source such as that in figure 2.4.2.1 that retrieves the publications authored by Knuth, and the associated bibliography and title.

An XML element is represented within a document rule as a labelled box, which may be nested to represent structural relationships. Boxes labelled with a regular expression may match any type of element, provided that their labels satisfy the pattern specified. Queries across several data sources may be formed. Query elements may be explicitly linked as in the query in figure 2.4.2.3, or joined using user-defined variables as in a QBE [ZLOO77] query.

[2] A Document Type Definition specifies the structure that all valid XML documents belonging to the corresponding data source must comply.

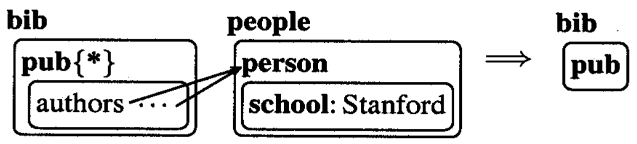


Figure 2.4.2.3. A query that retrieves the publications, and the associated bibliography, authored by members of Stanford University. The element labelled pub is used in the argument pattern and in the result pattern, which is specified to the right of the directed arc.

Support for matching elements at an arbitrary depth in a hierarchy is provided. As illustrated in figure 2.4.2.4, preceding the label of a query element by diagonal ellipses represents that the element will be matched against any sub-element of the element currently matched by the parent element.

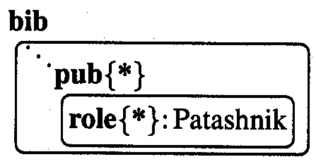


Figure 2.4.2.4. A query that retrieves the publications in which Patashnik was involved. The label role followed by the regular expression enclosed in curly brackets represents any element that is a child of that represented by the label pub which has the value *Patashnik.*

Languages for querying semi-structured data generally provide powerful operators for querying paths through the data stored, typically expressed using some form of regular expression. However, these languages are intended for querying the sub-elements in a hierarchy. Their usefulness for finding paths through more general forms of graph, where the elements are not necessarily all connected, is limited as the set of paths matched is dependant on the structure of the data queried. Consider figure 2.4.2.5.

```
<IncidentReport id = ″R1″>
        <heading>Knife Fight</heading>

        <mentions>
                <Person id = ″P1″>
                        <name>David Cole</name>
                        <age>25</age>
                </Person>
                <Person id = 'P2'>
                        <name>Ian Jones</name>
                        <age>27</age>
                </Person>
        </mentions>
</IncidentReport>
```

Figure 2.4.2.5: A XML document representing a police incident report.

If David Cole and Ian Jones were suspects in an investigation in would be helpful to see if there was any connection between the two. The operators for querying sub-elements in a hierarchy provided in languages such as Xing are not well suited for finding such paths. The elements representing these people are defined in different sub-trees of the incident report document. Thus, no connecting path from either the element representing David or that representing Ian would be found as neither element is a sub-element of the other.

Some textual languages for processing XML, such as XPath [WWWC], do provide operators for traversing up and down a hierarchy as required, possibly with no knowledge of the underlying structure of the data. However, these languages are not intended as end-user tools and are not suitable for use by problem domain experts with little or no knowledge of the underlying theory as forming expressions to represent the connections between nodes can be complex. Additionally, a rooted tree representation is still assumed.

The focus on querying hierarchies is also reflected in the range of data visualisation facilities provided in these systems, which are typically akin to those provided in the Sentences explorer (described in section 2.2.4). Hence, they are generally unsuitable for comprehending the connections between nodes in a network, a prerequisite for supporting LA.

2.5 Discussion

The facilities in the systems evaluated in section 2.2 allow varying degrees of support to be provided for LA. The level of support that could be provided by each system is dependant on the data model used, the query language supported, the browsing facilities provided and the data visualisation facilities. As discussed in section 2.2, no one system meets all the design criteria specified in section 1.4. However, our experimentation with these systems contributed towards the research reported in this thesis as the design criteria described in section 1.4 were constructed from an evaluation of the systems.

The criteria in section 1.4 relating to the data model were included due to the success of the data model used in Watson for modelling real-world LA data. Watson also contributed to the criteria relating to the facilities that should be provided to increase the comprehensibility of a link chart, as did VisualQ, as this system allows the results of multiple queries to be integrated in a single link chart. Our use of VisualQ also demonstrated that supporting fundamental data analysis operations via browsing facilities allows the facilities to be provided in a usable and efficient manner.

The criteria relating to the set of queries that should be able to be expressed in the query language were included as a result of our experimentation with Hydra. Our experimentation with the DBMS Sentences, and a prototype of the system we implemented, illustrated how a DBMS using a data model directly supporting the model of data used for finding connections in LA allows the database interface to be used without having to configure it to the schema of the database queried.

Our experimentation with the systems described in section 2.3 was less valuable in terms of its contribution towards the system design criteria. However, it helped us clarify why problem domains supported by such software require specialised query systems, and allowed us to appreciate where allowing interaction between a system for supporting LA and these systems may be beneficial. This is discussed further in section 4.5.

Chapter 3 – The Data Model of EDVC

3.1 Introduction

LA products such as Watson (described in section 2.2.1) were developed as a result of a practical need in many law enforcement areas to analyse the connections between objects of interest in an investigation, and was developed independently of DBMS and data modelling concepts. Nonetheless, the tools designed for supporting LA that provide access to the data stored in a database all use essentially the same data model. This model uses two basic modelling constructs, one for modelling different types of objects and another for modelling different types of directed links. A set of attributes may be defined for object or link types, which are equivalent to the columns of a table in a record based data model and represent attribute information particular to the object or link type in question.

This data model has been successfully used in tools like Watson to model the data analysed in LA. However, the simplicity of the model does have limitations. Useful connections may exist between objects in an investigation due to common attribute information. Consider a criminal investigation where a witness overheard a muffled conversation between suspects where the number 24 had been of significance. This number may be the age of a person, a house number or the quantity of an illegal substance.

When attribute information is stored in a record structure, such investigations in general can only by answered by forming several queries, providing the user has total knowledge of the database schema. Values containing the value 24 may be duplicated many times as there may be several attributes storing pieces of text, and links between values and objects cannot be represented in the model. Thus, to extract the required information the structure of the data model must be known by the user, which often proves problematic for inexperienced database users as they can find the concept of database schema difficult to understand [CATA00].

Additionally, type hierarchies are not supported. Link type hierarchies tend not to be of use in LA. Usually more general relationships such as *family member* are used instead of more specific relationships like *brother*, although such information may be modelled as a link attribute value. However, we believe that object type hierarchies could prove of value in LA and are a useful tool for incorporating additional problem domain information into a database schema.

The motivation for the design of the data model of EDVC was to preserve the usefulness and simplicity of the data model used in LA products while adding type hierarchies for object types and solving the issue of how to find useful connections via common object attribute information, at least from the perspective of the data model.

3.2 The Data Model

3.2.1 Object Types

Objects types have no internal structure and model objects of interest in the problem domain that would normally have attribute information stored about them. Each object type must have an associated name consisting of a sequence of letters, digits and white space that is unique within the set of object type names defined for a database schema.

An object type may be declared a *subtype* of one or more other object types (its *supertypes*) providing that cycles are not introduced into the type hierarchy. An instance of a subtype is considered an instance of its supertypes, and each object has a single most specific type. An object type may be declared *abstract*, which means that no object can have this object type as its most specific type.

3.2.2 Value Types

Value types are system-defined types that have no internal structure and model objects of interest in the problem domain that are directly displayable and self explanatory, and would not normally have information stored about them. The following value types are currently supported in EDVC: date, text, number, time, timestamp, truth and picture[3].

3.2.3 Link Types

Link types model directed relationships in the problem domain. A link type must be defined from an object type to an object type, or from an object type to a value type, and must have an associated name consisting of a sequence of letters, digits and white space. The name of a link type must be unique within the set of link types defined from the object type from which the link type is defined; this allows link type names to be overloaded.

[3] The value types truth and picture represent Boolean values and images respectively.

Each link type must be declared single-valued or multivalued. Each instance of the object type from which a single-valued link type is defined can have zero or one instances of that link type defined from it. Each instance of the object type from which a multivalued link type is defined can have zero or more instances of that link type defined from it.

A link type may have a set of attributes defined. Each attribute defined for a link type is a name/value-type pair modelling a piece of information that may be stored directly about an instance of the link type. Each attribute defined for a link type must have a name that is unique for the set of attributes defined for that link type.

3.2.4 Discussion

As described in section 3.1, the inclusion of object type hierarchies in the data model of EDVC was seen as a useful tool for incorporating additional problem domain information into a database schema. The manner in which object attribute information is modelled is heavily influenced by the way in which it is modelled in the binary-relational data model of Hydra, which is described in section 2.2.2.

As illustrated in figure 3.2.4.1, modelling objects and values as atomic data values with no internal structure, and modelling object attribute information as links defined from objects to values, proves useful when searching for meaningful connections between objects using common attribute information, providing that a query language that supports the necessary operators for finding the connections is used to analyse the data stored.

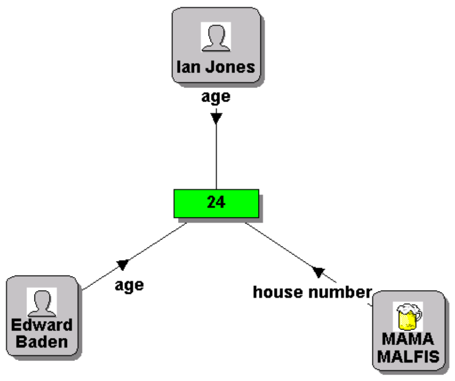


Figure 3.2.4.1. Illustrating how connections between objects via values may be explored.

Thus, these simple enhancements to the data model typically used by tools for supporting LA provides a platform for any query language that may be used to find useful connections via object attribute information, and supports the object type hierarchies that may occur in some problem domains.

3.3 A Database from the Perspective of a User

From the perspective of a user each database that is accessed via EDVC primarily consists of instances of the types defined in the database schema, where instances of objects types, value types and link types are referred to as *objects*, *values* and *links* respectively.

Each object is declared an instance of a non-abstract object type, which is its single most specific type, and is uniquely identified by a user visible surrogate consisting of a sequence of letters, digits and white space referred to as the *object identifier*. User visible surrogates are used in EDVC as the unique identifiers for objects of interest in the problem domain tend to contain information that may be of use to a problem domain expert [KENT79], even those which are considered to be meaningless. Thus, using such identifiers allows objects to be uniquely identified when listed while providing potentially useful information to the user.

A value is uniquely identified by its value and associated type. Each link represents a directed relationship between two objects, or from an object to a value, and like an object has an identifier. However, link identifiers are system-generated and are not visible to the user. A link can be an instance of an *object link type*. An object link type is defined from an object to an object type, or from an object to a value type. An instance of an object link type represents information stored directly about a particular object that may only be stored for that object.

3.4 The Visual Properties of Database Schema Elements

3.4.1 Introduction

Although there is no formally defined universal standard for displaying information in a link chart there are commonly used conventions that are intended to make the data displayed in a chart more comprehensible [PETE98]. The shape of an object is often used to indicate the type of thing in the problem domain that the object represents; objects representing people are often displayed as circles whereas objects representing organisations are often displayed as squares. An icon displayed within the boundary of the shape of an object may also be used to indicate the type of the object, usually a more specific type than the shape indicates, such as an icon indicating that a particular person is a burglar.

The visual properties of links are also often used to convey information to the user. Links where the investigator thinks that the strength of the link is weak, such as when two people are associates but only casual associates, may be displayed with a dotted line, whereas links deemed strong, such as when a large quantity of drugs has been sent from one supplier to another, may be displayed with a solid line. The thickness or colour of a line may also be used to indicate the specific type of a link [FREE00] or information stored about a link.

For these reasons, each type in an EDVC schema has an associated set of visual properties that determine the default visual properties of an instance of the element when the element is initially displayed in a link chart, and also the appearance of the element when displayed in a database schema diagram; see figure 3.4.1.1.

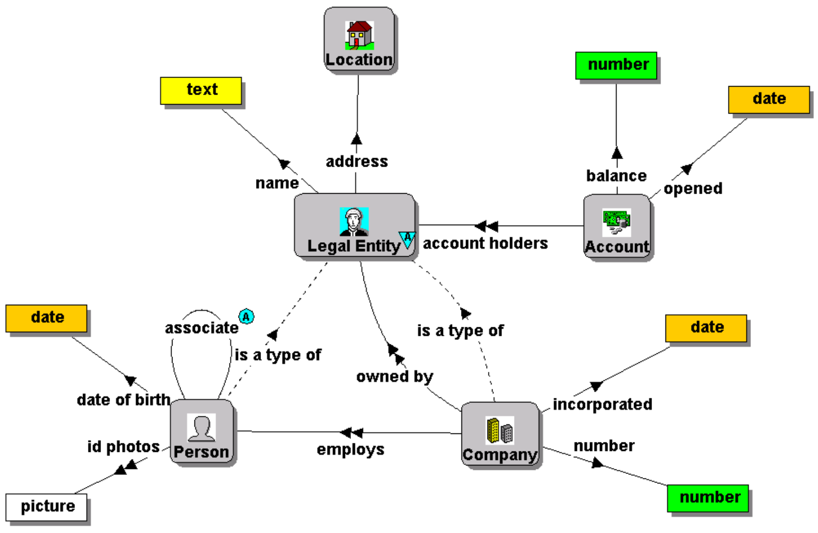


Figure 3.4.1.1. A graphical representation of a database schema. The dashed directed arcs labelled *is a type of* represent subtype relationships. The letter A enclosed within a triangle next to the label of the object type *legal entity* signifies that the object type is abstract, and the letter A enclosed within a circle next to the label of the link type *associate* indicates that this link type has attributes defined.

The set of visual properties defined for each schema element in EDVC were chosen to allow the common conventions used in LA to be supported, although there is no guarantee all such conventions can be supported, as there is no common formal definition for these conventions.

3.4.2 Object Types

The visual properties that may be defined for each object type are: shape, background colour, if there should be a shadow effect, outline colour, an optional associated icon, label colour, label font and a list of *label generation instructions*. The list of label generation instructions associated with an object type specifies the default label initially given to an instance of the type when added to a link chart by a query operation.

A label generation instruction may specify that the object identifier should be used, a new-line character, a sequence of given characters, or the set of values linked to the object by links whose associated link type is specified and defined from the object type or a supertype. Instructions defined in the list are executed in turn to generate the label for an instance of the type when added to a link chart. If an empty label is generated the object is labelled with its identifier. By default, an object type is given a single instruction specifying that the identifier of an object should be used.

The text displayed in the label of an object in a link chart is not typically the identifier of the object in the database. More often than not the label contains a subset of the object's attribute information, such as a person's name, which may not necessarily be unique within a database but probably is within a chart. Automatically displaying this information in the label of an object allows the user to identify who or what the object represents as soon as an object is added to the chart and prevents the user having to modify the labels of objects manually.

3.4.3 Link Types

A link type also has a list of label generation instructions defined. These allow a user to see relevant link attribute information as soon as instances of a link type are displayed in a chart. A link type label generation instruction can specify that a new-line character should be used, a sequence of given characters, the name of the link type, or the value of one of the attributes defined for the link added. If the label generated for a link is empty the link is labelled with the name of the associated link type. By default, a link type is given a single instruction specifying that the name of the link type should be used.

The other visual properties that may be defined for a link type are: colour of connecting arc, thickness of the connecting arc, if directed arrows should be used, label colour, label font, the background colour of the label and outline colour of the label background. When instances of object link types are displayed in a chart they are given a default set of visual properties.

3.4.4 Value Types

Each value type has an associated system-specified background colour, unique for the set of value types defined in the system. When a value is added to a link chart it is automatically given the background colour associated with the corresponding value type, which may not be changed. This allows the user to clearly differentiate between different types of values when displayed in a chart, and elsewhere in the user interface.

The set of colours associated with the value types currently defined are: yellow for text, green for number, blue for truth, pink for timestamp, orange for date, magenta for time and white for picture. The colours were not chosen according to any existing convention or empirical results, as attribute information is not modelled in this manner in existing LA tools and is not typically displayed in a link chart.

3.5 Discussion

The modelling constructs of the EDVC data model, and the set of visual properties associated with schema elements do not differ significantly from those used in existing LA products, although as described in section 3.2.4, we believe the features of the data model additional to those of that traditionally used in LA will prove of value. An added advantage of the model, illustrated previously in figure 3.4.1.1, is that a database schema has a natural graphical representation similar to that of a link chart. Displaying such a graphical representation of the schema of the database browsed may prove of use to experienced database users wanting to gain an appreciation of how data is modelled, and is supported in EDVC.

Chapter 4 – The Explorer Interface

4.1 Introduction

As described in section 1.1, in LA a chart is constructed using three fundamental data analysis operations: add an object to the chart, display the paths connecting two objects and display the objects directly or indirectly connected to an object. This is not to say that supporting these operations would be all that is required for every problem domain where LA is of value. Some domains may require more specialised operations, such as finding all the groups that a particular person in the chart is a member of. Others problem domains may require the fundamental LA data analysis operations to be applied to groups of objects that meet given criteria concerning their attribute information or the connections between them, such as displaying males aged between 25 and 35 weighing approximately 200 pounds that have previous form for robbery and are connected to a member of Daltons Bank.

Thus, while certain LA data analysis operations may be supported in a database interface by specifically designed browsing operations, others need to be supported using a flexible query language that should provide for queries that may include conditions that relate to the connections between objects, as well as conditions that relate to object attribute information. Both of these styles of database interaction may be of value when constructing a single link chart, as may altering the chart to improve comprehensibility and adding elements such as text notes and labelled boxes that do not represent elements stored in the database.

Products for supporting LA, such as Watson (described in section 2.2.1) provide powerful data visualisation facilities for increasing the comprehensibility of a chart, but do not provide the necessary range of query facilities. The opposite is true for database query systems. Whereas some do provide powerful query languages capable of general computation as well as database access, they often use inappropriate data models, and the facilities they provide for visualising the results of queries are not powerful enough for supporting LA. They do not allow the user to construct a link chart in stages by integrating the results, or a subset of the results, returned by a number of queries formed using either a browsing style of interaction, a query language or both. This is limiting, especially as they only support a fixed set of browsing operations.

Providing database interface support for exploring the connections between objects of interest in the problem domain by constructing a link chart using an appropriate set of browsing operations, a query language with the necessary expressive power, data visualisation facilities for increasing the comprehensibility of a chart by editing elements, and by adding elements not stored in the database was the motivation for the design and implementation of the explorer interface, an example of which is given in figure 4.1.1.

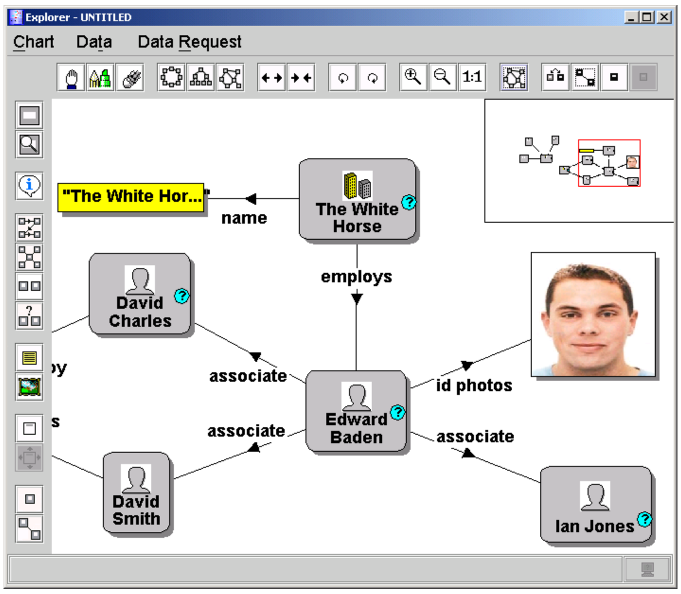


Figure 4.1.1. An example of an explorer interface. Objects displayed with a question mark enclosed in a circle are involved in one or more links stored in the database that are not displayed in the chart.

The interface provides a number of browsing facilities for adding objects and links to a chart, and for exploring the connections between objects. Each facility is designed to either support one of the fundamental data analysis operations used in LA to construct a link chart, or was included as a result of our experience with analysing crime related data such as that used in the tutorial session for the user evaluation of EDVC (described in section 6.2.11), as these data analysis operations may be of value in other problem domains.

The data analysis operations supported by the browsing and data visualisation facilities supported in the interface are discussed below using the database displayed in figure 4.1.2 to provide relevant examples. The query language that may also be used to add objects and links to a link chart, or to filter the objects and links displayed in an existing chart, is described separately in chapter 5.

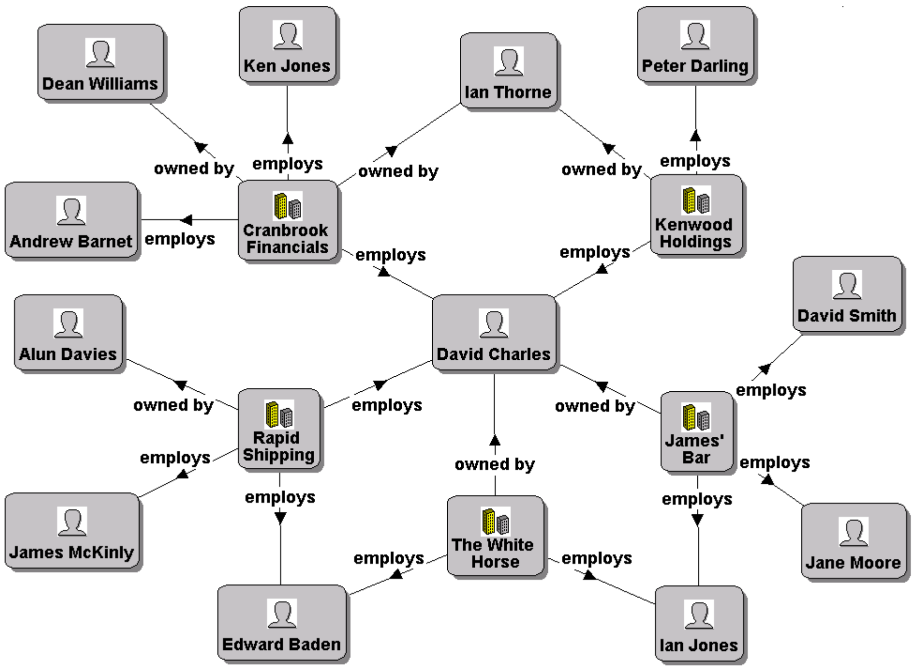
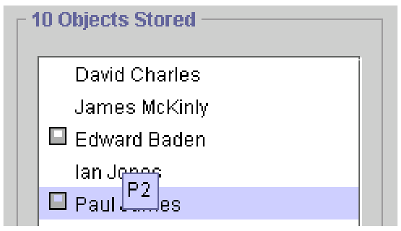


Figure 4.1.2. A fictional database modelled upon the schema given in figure 3.4.1.1 that is used in the examples provided in sections 4.2 and 4.3.

4.2 The Fundamental Data Analysis Operations Supported

4.2.1 Adding Objects & Values Directly to a Link Chart

An object may be added directly to a chart from a list of those stored. The list may be filtered by restricting the type of objects displayed and by using wildcard pattern matching similar to that used for files in the UNIX operating system. As in all places within the interface where objects are listed, the labels displayed are those generated using the label generation instructions for the associated object types (described in section 3.4.2). As illustrated in figure 4.2.1.1 the object identifiers may be obtained as a tool tip.


10 Objects Stored

David Charles
James McKinly
☐ Edward Baden
Ian Jones
☐ Paul James

P2

Figure 4.2.1.1. Displaying the identifier of an object as a tooltip. All dialogs that list objects and values place an icon next to those already displayed in the chart.

Values may be added directly to a chart in a similar manner to that described above for adding objects. However, when dates and times are listed they may not necessarily be the same as those stored in the database. The user may set a time zone for the values in the database and another for the interface, which by default are the same.

If the time zone set for the interface is different from that for the database then when time and timestamp values are displayed their value will be different from those stored. The dates and times displayed are those in the time zone set for the interface that are equivalent to those stored in the database, which are assumed to be in the time zone set for the database. We believe this feature could be of value when examining data from distinct regions or countries. For example, consider an international fraud investigation where money was being transferred between accounts in New York and London, and the data generated from the accounts in New York was to be analysed by investigators in London. This facility could be used to offset the time difference, increasing comprehensibility and minimising the work required.

When picture values are displayed in a link chart their dimensions may not be the same as the images stored in the database. Similarly for text values, not all of the characters in the values stored in the database may be displayed. Each link chart has a set of display options defined that allows the dimensions of picture values to be constrained, and the maximum number of characters in a text value to be restricted. If a picture value exceeds these limits then when the value is displayed it is scaled to satisfy them. As illustrated in figure 4.1.1 if a text value needs to be truncated three periods follow the characters that have been displayed to indicate this to the user. These limits prevent these types of values obscuring other elements when displayed or becoming incomprehensible due to their size.

4.2.2 Displaying Information Stored Directly About an Object or Link

Objects and values linked to a specified object may be displayed in a separate window then added to the chart. Similarly, the attribute values of a link in a chart may be listed in a separate window, but cannot be added to the chart; a discussion of how this may be supported is discussed in section 8.2.6. The objects and values, and associated links, represented in the window and currently displayed in the chart are highlighted. Such windows persist until cancelled and the visual cues are updated to reflect modifications to the chart.

For example, the window in figure 4.2.2.1 displays the objects and values linked to the object representing Edward Baden. The type of link that an object or value is linked by is indicated using the label of the associated panel and that of the tabbed pane. A tabbed pane labelled with the identifier of the object selected lists the objects and values linked to the selected object by instances of object link types; see section 3.3.

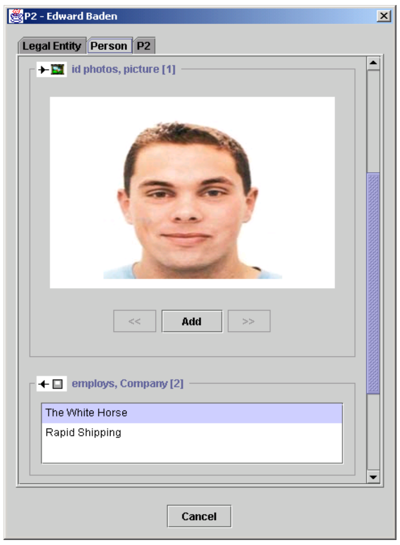


Figure 4.2.2.1. A window displaying information stored directly about the object representing Edward Baden, which has the object identifier P2. The object labelled 'The White Horse' is highlighted as it is already displayed in the same chart.

4.2.3 Displaying the Objects Connected to a Specified Object

The objects connected to a specified object, with their connecting paths, may be added to the chart. The connecting paths to be added may be constrained by specifying query parameters: a set of link types LTS requiring that each link in a valid path must be an instance of a link type LT $\in$ LTS, the maximum number of links in a valid path, and if every link in a valid path must follow the direction in which it is defined or alternatively must go against this direction.

Paths containing loops are not displayed since generally they supply no more information and can result in a large number of redundant paths being displayed. Paths which pass through links defined to instances of value types are also not displayed as they may reflect that two objects in a path have common membership of a set; such as two objects representing people that are a member of the set of people aged 23[4].

For example, if James' Bar had been added to an empty chart and the objects connected to this object by a path containing 3 or less links, all representing a company employs a person, were also added, as were the connecting paths, then we see from figure 4.1.2 that the chart would be that in figure 4.2.3.1.

[4] These restrictions were first implemented in Hydra, which is described in section 2.2.2.

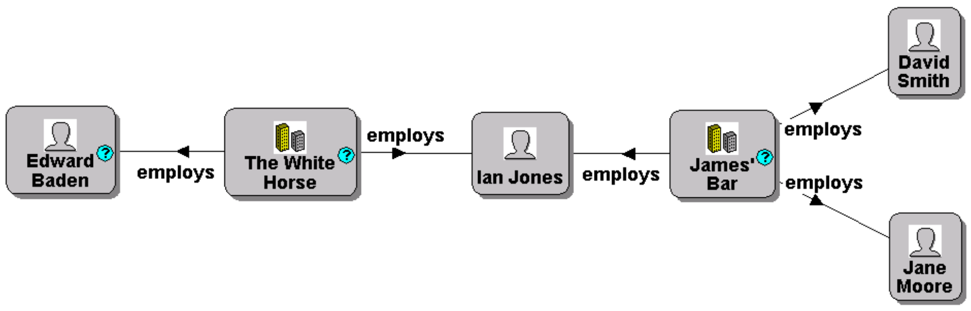


Figure 4.2.3.1. The contents of a link chart after the object representing James' Bar has been added and the objects connected to that object by a path containing 3 or less links, all representing a company employs a person; the connecting paths are also displayed.

4.2.4 Displaying the Paths Connecting Two Specified Objects

The paths in the database connecting two specified objects may be added to the chart. The paths considered valid may be constrained by specifying query parameters: a set of link types LTS requiring that each link in a valid path must be an instance of a link type LT $\in$ LTS, the maximum number of links in a valid path, and if every link in a valid path must follow the direction in which it is defined or alternatively must go against this direction.

For the reasons described in section 4.2.3, paths do not contain loops or links defined to instances of value types. However, they may be closed circuits, as this is of value in problem domains such as transport networks. For example, if objects represented airports and links flights between airports then such paths would represent trips that start from and return to the same location.

If Edward Baden and David Smith were suspected of being significant figures in a financial fraud it would be useful to see the connections between the two. One such path is already displayed in the chart in figure 4.2.3.1, to ascertain if any more meaningful connections exist the paths containing 4 or less links will now also be added. The chart, after modifications to its layout to increase comprehensibility now looks like that in figure 4.2.4.1.

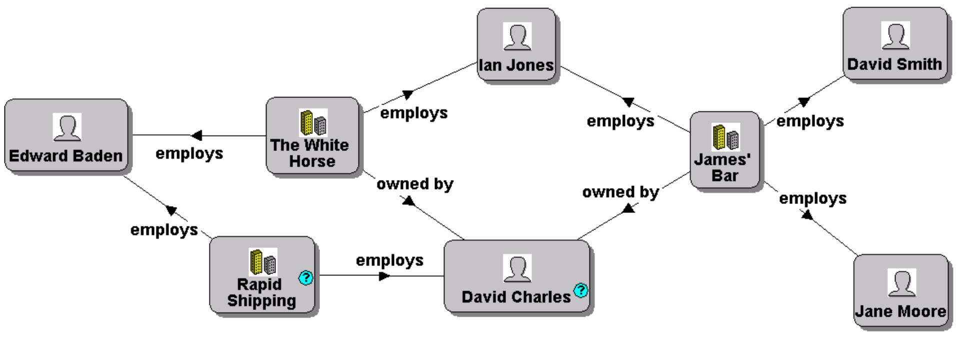


Figure 4.2.4.1. The chart displayed in figure 4.2.3.1 with paths containing 4 or less links that connect Edward Baden and David Smith added, and modifications to its layout to increase comprehensibility.

4.3 Additional Data Analysis Operations Supported

4.3.1 Displaying the Objects Connected to Several Specified Objects

The objects connected to two or more specified objects may be listed in a separate dialog and then added to the chart, as may the connecting paths. The objects listed may be filtered by selecting a set of object types OTS so that each object displayed O must be an instance of an object type OT $\in$ OTS. The paths connecting O to the objects specified which are considered valid may also be constrained using additional query parameters: a set of link types LTS requiring that each link in a valid path is an instance of a link type LT $\in$ LTS, the maximum number of links LEN in a valid path, and if each link in a valid path must follow the direction in which it is defined or alternatively must go against this direction.

The objects displayed do not have to be connected to all of the objects specified; they have to be connected to N or more of these objects, where N is a positive integer specified as a query parameter and is less than or equal to the number of objects selected. If all five query parameters are specified then each object that is displayed will be an instance of an object type OT $\in$ OTS, be connected by a set of paths P to N or more of the objects specified where the number of links in each path in P is less then or equal to LEN, and each link in each path in P is an instance of a link type LT $\in$ LTS and is defined in an appropriate direction.

This facility may be of use in gang related criminal investigations. Consider figure 4.1.2, if David Charles, Andrew Barnet and Ian Thorne were suspects in a financial fraud where it was thought likely that others were involved, displaying the objects representing people connected to these people would be of value. If a chart initially contained these three objects then after adding the objects representing people connected to at least two of the three objects by a path in the database containing 2 or less links, and the connecting paths, the chart would be that in figure 4.3.1.1.

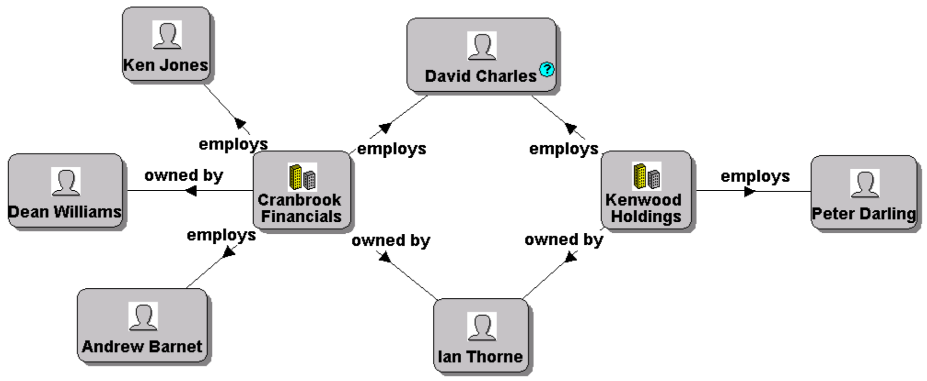


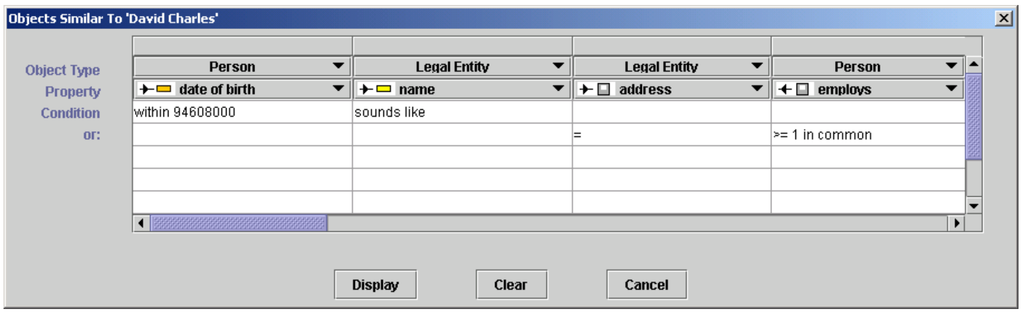
Figure 4.3.1.1. A link chart that initially contained the objects representing David Charles, Ian Throne and Andrew Barnet with the objects representing people connected to at least two of these objects by paths containing 2 or less links added, and the connecting paths.

4.3.2 Displaying the Objects Linked to a Number of Objects

The objects in the database may be listed in a separate dialog and then added to the chart. The objects listed may be filtered. A set of object types OTS may be selected so that each object displayed O must be an instance of an object type OT ∈ OTS. A condition (N C I) may be specified where C ∈ (>, >=, =, <=, <), I is a non-negative integer, and N is the number of links between objects involving object O such that each object displayed O must satisfy (N C I). The set of links between objects involving O counted for N may be restricted by specifying a set of link types LTS requiring that each link counted is an instance of a link type LT ∈ LTS, and also by specifying if each link must be defined from O or alternatively to O. This facility may be of use for detecting significant objects, which are often highly connected.

4.3.3 Displaying the Objects Similar to a Specified Object

Instances of the same type as that of a specified object that are deemed similar to it according to a set of given similarity conditions, may be displayed in a separate window and then added to the chart. As shown in figure 4.3.3.1, similarity conditions relate to information stored directly about instances of the object type and are grouped in a grid.



Object Type	Person	Legal Entity	Legal Entity	Person
Property	date of birth	name	address	employs
Condition	within 94608000	sounds like		
or:			=	>= 1 in common

Figure 4.3.3.1. Similarity conditions matching all instances of the object type person that represent people whose date of birth is within three years (specified as seconds) of David Charles' and have a similar sounding name; or have the same address as David and are employed in at least one of the same companies as David.

Each grid cell may contain at most a single condition, formally defined in appendix B. The information that a condition relates to is specified by the object type and link type selected in the associated column. Only instances of the same object type that the specified object is an instance of and which satisfy all conditions in at least one of the rows in the grid are listed in the window displayed. Boolean connectives are not explicitly used as their use can lead to errors when constructing queries in database query languages [GREE90].

For example, identifying people who have similar characteristics to that of a known criminal, or a criminal profile, may prove productive in identifying previously unknown identities or for creating a list of possible suspects. If David Charles was a prime suspect in several investigations this may indicate that he is a significant criminal and uses aliases. The information used by criminals in their aliases tends to have a degree of similarity, such as the same first name or date of birth, to allow the criminal to manage different identities. Thus, a set of conditions such as those specified in the condition grid in figure 4.3.3.1 could be useful for identifying aliases that David uses.

4.3.4 Displaying the Common Properties of Two Specified Objects

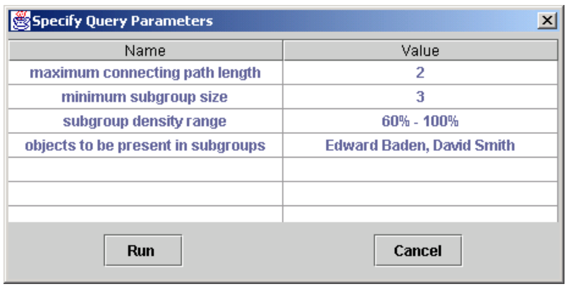
The objects and values linked to two specified objects by links that are instances of the same link type may be displayed in a separate window then added to the chart. For every object or value displayed OV there will be at least one link L1 involving the first object selected O1 and at least one link L2 involving the second object selected O2 such that L1 and L2 are instances of the same link type, and the following condition is satisfied: L1 is defined from OV to O1 and L2 is defined from OV to O2, or L1 is defined from O1 to OV and L2 is defined from O2 to OV. This facility may be used to ascertain how similar two objects are.

For example, if two people were deemed similar using the facility described in section 4.3.3 then this facility could be used to determine their level of similarity.

4.3.5 Extending the Operations Supported

The set of browsing operations supported in the explorer interface is not fixed as there are queries that may be of value in LA that are not currently supported, such as finding closely connected subgroups. Supporting operations such as this in a general-purpose database query interface can be problematic as the most appropriate algorithms and query parameters to provide are often dependent on the characteristics of the data analysed. It is for his reason the range of operations supported in the explorer interface can be extended without the need for further system development.

A programming framework is provided that allows software components implementing query operations, which may be parameterised, to display the results returned in a new or existing link chart. Each additional component must provide a human readable description of the operation implemented and of the required parameter values, whose format is also defined within the component. This allows the user to choose operations from a list of those available and for meaningful feedback to be provided when invalid parameter values are specified, either by selecting elements in a chart or via a form-based interface like that in figure 4.3.5.1.



Name	Value
maximum connecting path length	2
minimum subgroup size	3
subgroup density range	60% - 100%
objects to be present in subgroups	Edward Baden, David Smith

Figure 4.3.5.1. The form-based interface for specifying query parameters.

4.4 Data Visualisation Facilities Supported

4.4.1 Adding Elements Not Stored in the Database to a Link Chart

The comprehensibility of a link chart may be increased with the addition of elements that do not represent objects, values or links stored in the database. Several types of elements may be added. As illustrated in figure 4.4.1.1, text notes representing information as free text can be added, as can boxes for grouping objects. Images in a suitable graphical format[5] may also be added.

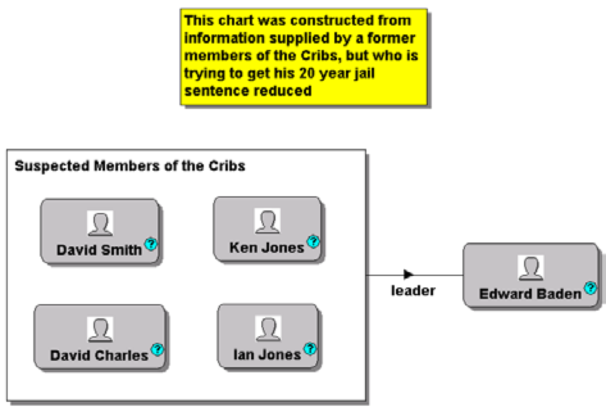

This chart was constructed from information supplied by a former members of the Cribs, but who is trying to get his 20 year jail sentence reduced
Suspected Members of the Cribs
David Smith
Ken Jones
David Charles
Ian Jones
leader
Edward Baden

Figure 4.4.1.1. A link chart containing a text note, a labelled box and a link not stored in the database.

Objects and links that do not represent any of those currently stored in the database can also be included in a chart. This may allow the user to determine what effect the addition of these elements would have on the network analysed without altering the view other users have of the data stored; a subject discussed further in section 8.2.3. Each of these types of elements, including those discussed above, has a distinct set of visual properties. They may be set when adding the element and edited.

4.4.2 Removing & Editing Link Chart Elements

When constructing a link chart using the database browsing facilities described in sections 4.2 and 4.3, the facilities for adding elements not representing objects and links stored in the database described in section 4.4.1, and the query language described in chapter 5, not of all the objects and links displayed may be of relevance. Therefore, elements may be removed from a chart at any stage during the lifetime of a chart, which neither affects the data in the database nor that displayed in other charts.

Charts elements may also be modified to increase comprehensibility. Each object and link in a link chart has an associated set of visual properties. These correspond to those defined for object types and link types in the database schema, discussed in section 3.4. They provide a default set of properties when objects and links are added to a chart by a browsing facility or the query language.

[5] The formats currently supported are the Graphics Interchange Format (GIF) and Joint Photographic Experts Group (JPEG) format.

The properties of individual objects and links can be modified, and the values to which these properties can be changed are not constrained unnecessarily. This allows two or more objects in a chart to use the same label, which may be of use if the names of people were used to label objects. Information not currently stored in the database may also be displayed in the labels of objects. Additionally, an object in the database that is displayed in several charts can have different labels. This is of use in criminal investigations as criminals often have aliases and this allows the most appropriate name to be used in a given context.

4.4.3 Automatic Chart Layout

While the layout of a chart may be modified to increase comprehensibility by dragging the elements to better positions, automatic chart layout facilities are provided that may reposition elements in a circular manner, hierarchically or to minimise the number of links that cross. The later facility would generally be most frequently used as networks with this property have a desirable quality that allows connections to be more clearly comprehended [HERM00].

Circular layouts are also common. In telephone call analysis, often utilised in the analysis of data relating to organised crime, telephones are typically represented as objects and calls as links, with attribute information concerning the time and duration of a call. As illustrated in figure 4.4.3.1, arranging the objects in a chart in a circular manner allows the telephones that made or received a relatively large number of calls to be determined visually.

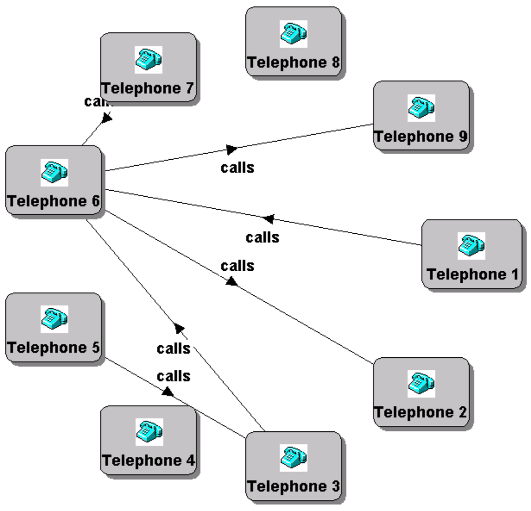


Figure 4.4.3.1. A link chart where the objects displayed are arranged in a circular manner.

Hierarchical layouts are of particular use where a subset of the data analysed may, but not necessarily, be structured in this manner. In organised criminal gangs there is often some form of implicit social hierarchy. As illustrated in figure 4.4.3.2, displaying the personal relationships between the members of such organisations in a chart and then using this facility may allow such social hierarchies to be detected.

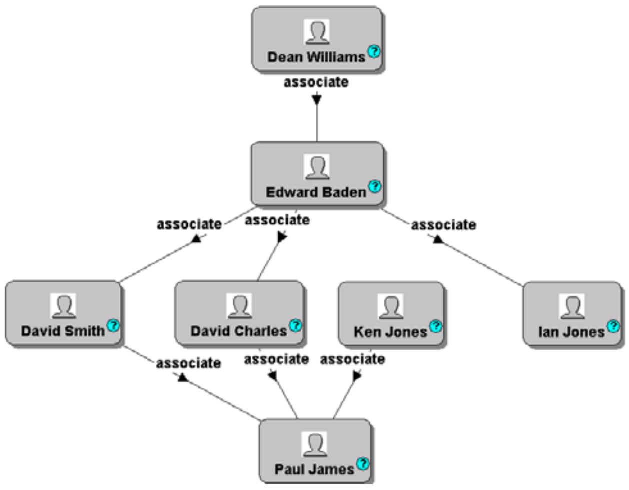


Figure 4.4.3.2. A link chart where the objects displayed are arranged in a hierarchical manner.

Several factors have an effect on the ease with which a graph can be understood, such as the number of links crossing and the spacing of nodes [COLE96]. The desired spacing between objects in a chart may vary between problem domains depending on the information included in the labels of objects. Therefore, each link chart has a set of layout parameters associated with it that controls this spacing when the layout of the chart is automatically rearranged. For the circular layout these control the circle radius, for the hierarchical the gap between levels.

4.5 Supporting Social Network Analysis

The data analysed using LA often represents some form of social network. As discussed in section 2.4.1, existing software for supporting Social Network Analysis (SNA) focuses more upon analysing the structural properties of relatively small homogenous networks rather than exploring the connections between the actors in very large heterogeneous networks. The data in a link chart can represent such a small homogenous network.

While the analysis facilities provided in software tools supporting SNA could be supported, this would represent unnecessary duplication, and the supply of a probably inferior range of facilities as many freely availably tools [ANAL, BATA97] are in continuous development. Therefore, there are no facilities provided for performing such analysis. However, there are facilities for generating data sets representing the network formed by the objects and links in a chart that may be analysed, possibly after some editing, using these tools.

Software supporting SNA commonly provides facilities for importing the data analysed from a text file representing an *adjacency matrix*, a form of matrix that represents the relationships between a set of actors in a social network. In an adjacency matrix there are the same number of rows as columns, and the contents of the cells represent the relationships between the actors, which by convention are directed. Each actor is associated with one row and one column in the matrix. The fact that there exists zero or more relationships from actor A to actor B is signified by there being a non-negative integer in the cell located in the row associated with actor A and the column associated with actor B; the integer represents the number of relationships.

Consider table 4.5.1. The cell located in the row associated with Edward Baden and the column associated with David Charles contains 1, this indicates that there is one relationship from Edward to David. Similarly, the cell located in the row associated with David Charles and the column associated with Edward Baden contains 0, indicating there is no relationship from David to Edward.

	David Charles	Ian Jones	David Smith	Edward Baden
David Charles	0	0	0	0
Ian Jones	0	0	0	0
David Smith	0	0	0	0
Edward Baden	1	1	1	0

Table 4.5.1. An adjacency matrix representing the social network formed by the links between the objects representing people in figure 4.1.1 displayed in the canvas.

Generating a matrix such as that given in table 4.5.1 where the contents of the cells indicate the number of paths less than or equal to a given length from one actor to another can also be useful for a number of reasons. Multiple paths connecting two actors may indicate a stronger association than a single direct relationship, and greater distances between the actors may suggest that it takes longer for information to diffuse [HANN99].

Adjacency matrices may be generated from the objects and links displayed in a chart using the explorer interface. The user may filter the objects represented in the matrix and the links used to generate the matrix, by restricting the types of objects and links considered to only those that are instances of types included in a list selected from those defined in the database schema.

Matrices representing the paths connecting a group of selected objects may also be generated in a similar manner to that described above for adjacency matrices. The connecting paths considered valid, which may not necessarily be displayed in the chart, may be restricted in exactly the same manner as the paths connecting two objects that may be displayed using the browsing facility described in section 4.2.4.

4.6 Discussion

The style of database interaction supported in the explorer interface differs from that which is supported in database query systems. Allowing the user to construct a single link chart by integrating the results returned by a number of browsing operations and queries, possibly removing and editing chart elements as results are being combined, breaks down the previous distinction made between the browsing and query language styles of database interaction. It also allows the power of the data visualisation facilities that have been provided in tools such as Watson (described in section 2.2.1), which include the ability to add non-database elements to a chart to increase comprehensibility, to be incorporated into a database interface.

This interaction between database browsing and a query language style of interface provides great flexibility while analysing the data stored. Charts can be constructed that could not be formed using just either the browsing operations or the query language supported in EDVC, as the expressive power of either is not a subset of the other. This style of interaction differs from the integration of browsing and querying as in PESTO [CARE96], and from the facility provided in Hy+ (described in section 2.3.1) to iteratively filter query results. However, the Hy+ style of interaction is supported in EDVC as the user may specify that the chart should be regarded as the database when a query is formed using the language described in chapter 5.

The expressive power of the set of browsing operations supported in the explorer interface is an improvement on other systems supporting LA. The fundamental data analysis operations used in LA, and discussed in section 1.1, are supported, and unlike tools such as Watson they consider the data stored in the database not just that displayed in a chart. The programming framework also allows problem domains that require specialised forms of data analysis to be supported that for reasons of practicality could not be directly supported in a generic database query interface.

Further flexibility for constructing link charts is provided by allowing a subset of the objects and links displayed in two or more charts to be combined. Several explorer interfaces may be open at the same time, each displaying a separate chart that may contain a different subset of the data stored, and elements may be freely copied between charts. This also allows large charts to be decomposed into several smaller charts and for distinct lines of investigation to be followed. Link charts may be saved for subsequent display and further development.

When a chart is reopened a number of checks are made as the underlying database may have been updated since the chart was saved. The first check is to verify that all objects displayed in the chart representing those stored in the database are still stored, with similar checks then being made for values and links. If any of these checks are not satisfied the user is informed and given a set of options. The elements representing objects, values and links stored in the database still stored, and those that represent text notes and labelled boxes, may be copied to a new chart and displayed. Alternatively, a new chart may be created and subsequently displayed that contains all those elements from the original chart except those that represent objects, values and links that are no longer stored in the underlying database, which are replaced by similar elements with identical visual properties that only exist in the chart.

Despite the provision of these facilities the explorer interface if used in isolation does not satisfy all the criteria listed in section 1.4 that relate to query facilities that should be provided in a database query interface to support LA. Therefore, a query language is also provided in EDVC that allows the user to form queries that filter objects and links stored in the database, or in a specified link chart, so that only the objects and links that match the criteria specified in the query are added to the chart.

Chapter 5 – Filter Pattern Query Language

5.1 Introduction

The query languages provided in LA products are not well suited for analysing connections between objects as the user has to specify how connections between objects are derived. This is often inappropriate in LA as the user often does not know the nature of the connection, as this is the information that is typically desired. This is also true of the majority of database query languages, even those that are explicitly designed to support graph traversal such as GaphLog (described section 2.3.1).

However, this is not true for Hydra and Spider (described in sections 2.2.2 and 2.3.4, which both use a binary-relational data model. Nonetheless, these languages are inappropriate for supporting LA. The simplicity of the binary-relational data model makes specifying conditions that relate to link attribute information problematic, as links need to be modelled as entity types. This impacts upon the ability to constrain the connections matched to represent particular type(s) of relationship in the problem domain, which is also negatively impacted upon in these languages by the lack of an appropriate range of meta-data query facilities as there is no way of directly determining the type of an entity in a connection when a query is evaluated.

The motivation for designing of the Filter Pattern Query (FPQ) language was to provide a language that would allow the user to specify declaratively queries for analysing connections between objects without requiring that the user should have to specify how connections are derived. It was also recognised that support should also be provided for constraining the connections matched so that they represent particular type(s) of relationships in the problem domain.

5.2 Usability Design Goals

5.2.1 Introduction

Domain experts in fields where LA is often of value typically have little or no experience of interacting directly with a DBMS. Therefore, for the reasons discussed in section 1.3 it was decided that a FPQ should be specified visually. A number of design guidelines were used to provide ease of use, based on the guidelines discussed by Papatonakis [PAPA95].

5.2.2 Flexibility in Specifying Queries

A FPQ only specifies the information to be displayed, the decision concerning how best to retrieve this information is left to the query processor. Additionally, the sequence of user actions performed while constructing a FPQ has no affect on its semantics. Therefore, the FPQ language is declarative, which decreases the effort required to specify a FPQ and can improve query evaluation times.

5.2.3 As Few Syntax Rules as Possible

The FPQ language uses a minimal syntax, but not an absolute minimum; see section 5.3.8. Controlled redundancy is used to allow FPQs to be specified concisely, and so that new users only have to grasp a small number of concepts to start to form useful queries in the language. This enables the semantics of a pattern to be more easily comprehended.

5.2.4 No Explicit Use of Boolean Connectives or Logical Quantifiers

Many users find Boolean connectives and logical quantifiers such as universal and existential quantification difficult to use when forming queries, and they can contribute to semantically incorrect queries being formed [GREE90, THOM76]. Therefore, the FPQ language does not require the explicit use of such connectives and operators when forming FPQs.

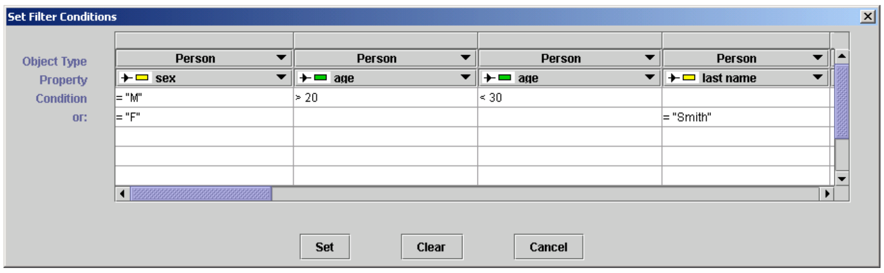
5.3 An Informal Presentation

5.3.1 Introduction

A FPQ is a visual template that is matched against the network formed by the objects and links stored in the database, or displayed in a specified link chart, to determine the sub-networks that match the template so that they may be displayed. Each filter pattern consists of a set of constraints, of which there are several types.

5.3.2 Object Constraints

An object constraint can be used to specify a particular object, instances of a specified object type, or any object, regardless of its type. Object constraints that only match instances of a specified object type may have a set of *filter conditions* (a syntactical definition of which is provided in appendix C) that restrict the objects matched to those satisfying the conditions. To allow object constraint filter conditions to be combined to form complex restrictions, and to increase filter pattern comprehensibility, filter conditions are not displayed in a filter pattern but are displayed separately and combined using a grid like that in figure 5.3.2.1.



Object Type	Person	Person	Person	Person
Property	sex	age	age	last name
Condition	= "M"	> 20	< 30	
or:	= "F"			= "Smith"

Figure 5.3.2.1. A set of filter conditions that match people who are male and are aged between 20 and 30, or are female and have Smith as a last name.

There may be at most one condition in each cell. Only objects satisfying all the conditions in at least one row are matched by the constraint. Object constraints that have filter conditions have a C enclosed within a magenta circle displayed next to their label, as in figure 5.3.2.2.

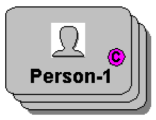


Figure 5.3.2.2. A filter pattern query containing a single object constraint that matches instances of the object type Person and has a set of filter conditions. Constraints matching instances of a specified object type are uniquely labelled with the name of the type followed by a hyphen and an integer.

5.3.3 *Link Constraints*

Link constraints connect two object constraints and specify that the objects matched by the constraints must be linked. Link constraints may only match instances of a specified link type, indicated by the link type name on the arc, and may if required take the direction of links into consideration, indicated by if the arc representing a constraint is directed; see figure 5.3.3.1.

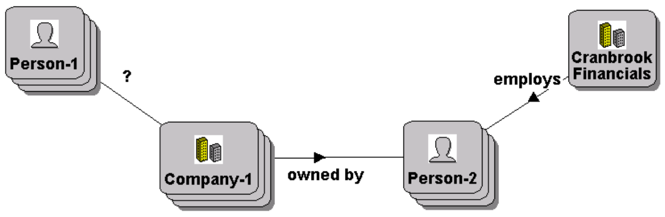
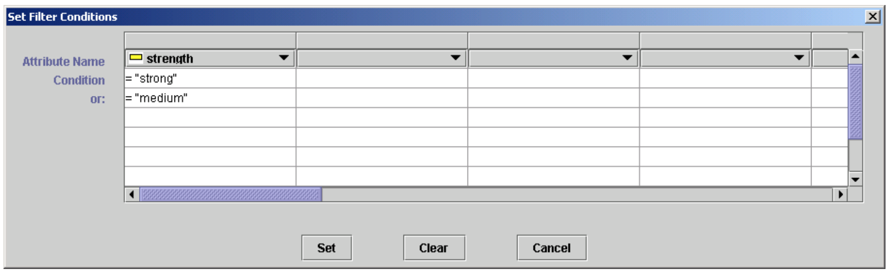


Figure 5.3.3.1. A filter pattern query that displays people linked to a company owned by an employee of Cranbrook Financials, and the associated people, companies and links. A question mark on an arc indicates that any type of link may be matched.

Link constraints that only match instances of a specified link type may have one or more filter conditions that restrict the instances matched to only those whose attribute values satisfy the conditions. Link constraint filter conditions (a syntactical definition is provided in appendix D) are not displayed in a FPQ but may be displayed separately, and are combined and evaluated in a manner similar to those of object constraints; see figure 5.3.3.2.



Attribute Name	strength			
Condition	="strong"			
or:	="medium"			

Figure 5.3.3.2. A set of filter conditions for a link constraint that matches links representing that two people are associated, where the association is either strong or of medium strength.

5.3.4 Path Constraints

A path constraint specifies that the objects matched by two specified object constraints must be connected. The connecting path(s) may be the shortest path(s) connecting the objects, or those containing less than a specified number of links. Path constraints are required for matching connections between objects where the length and structure of a connection cannot be predicted. These are difficult to express using only object and link constraints, especially if the user is unfamiliar with the database schema.

A path constraint is represented in a FPQ in a similar manner to a link constraint, but the arc representing the constraint is undirected, dashed and labelled with a piece of text specified by the user that indicates which paths they believe are matched; see figure 5.3.4.1.

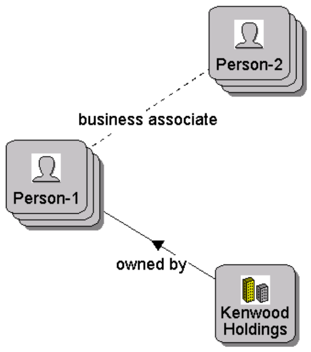


Figure 5.3.4.1. A filter pattern query that displays people connected to an owner of Kenwood Holdings, and the associated objects and links. The label *business associate* is specified by the user.

To increase FPQ comprehensibility the maximum number of links that may be contained in the connections matched by a path constraint is not displayed in a FPQ, but if required may be displayed separately.

The connections matched by a path constraint may be filtered by a *path filter*, which is not displayed in a FPQ but if required may be displayed separately. A path filter is similar to a Finite State Automaton (FSA) [RAYW83] except that states transitions, which match links in a path, and the states, which match objects, may also evaluate information stored directly about objects and links in a path, and their type.

A path filter is represented in a similar manner to regular expressions in the 4th notation used in the experiment described in section 2.3.2. Each path filter consists of a non-empty set of object and link constraints, similar to those described in sections 5.3.2 and 5.3.3 for FPQs, but which may not be excluded. There are two object constraints in each path filter that represent the object constraints connected by the associated path constraint. These constraints are always highlighted and are equivalent to the start and finish states in an FSA.

Consider figure 5.3.4.2. This path filter matches all paths connecting the object matched by the object constraint Person-1 and that matched by the object constraint Person-2 that either represent they have a mutual associate or that the first person is employed in a company that also employs an associate of the second.

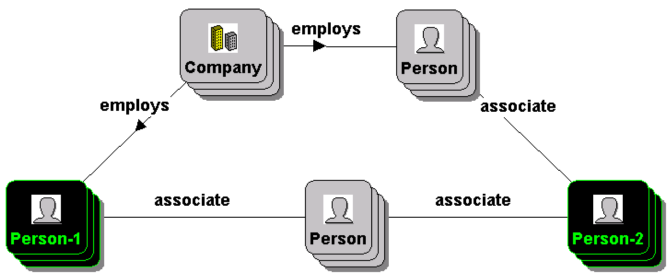


Figure 5.3.4.2. A path filter where each path in the set of paths matched may belong to either one of the two path structures specified.

With the exception of the object constraints that represent the start and finish states the object constraints in a path filter that match instances of a specified object type are labelled with the name of the object type.

Unlike the object constraints described in section 5.3.2 for FPQs, the filter conditions defined for object constraints in a path filter may not be interdependent. For example, in figure 5.3.4.2 a filter condition defined for the object constraint matching instances of the object type

Company cannot refer to information stored directly about an object matched by any of the other object constraints defined in the path filter.

As shown in figure 5.3.4.2, alternate path structures are expressed by specifying two or more link constraints from, or to, an object constraint. The direction of the link constraints only determines which links are matched.

The repetition of a specified structure within a connecting path is represented using a loop. Consider figure 5.3.4.3. This path filter matches all paths connecting the object matched by the object constraint Person-1 and that matched by the object constraint Person-2 that consist only of instances of a link type *associate*, and where all objects in the paths are instances of the object type Person.

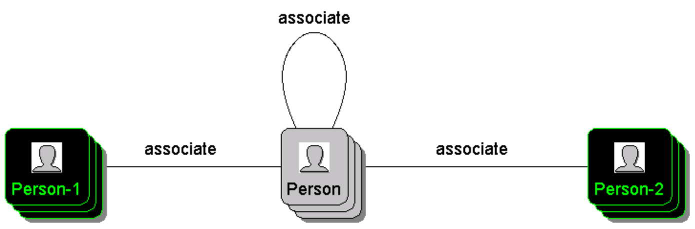


Figure 5.3.4.3. A path filter where a connecting path may contain a specified repeating structure.

A user may specify more concisely that one or more instances of a particular link type are required. Any link constraint in a path filter may be declared *indirect*. This indicates that the objects matched by the object constraints must be connected via a path consisting of one or more links that match the constraint. As illustrated in figure 5.3.4.4, link constraints declared indirect appear with a dashed connecting arc.

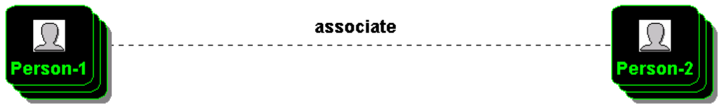


Figure 5.3.4.4. A path filter containing a link constraint declared indirect that matches the same paths as that in figure 5.3.4.3.

With the exception of object constraints that represent the start and finish states, any object or link constraint in a path filter that only matches instances of a specified type in the database schema may be *inverted*. This indicates that only objects or links not satisfying the conditions specified should be matched by the constraint. As illustrated in figure 5.3.4.5, if an object or link constraint is inverted it appears crossed out.

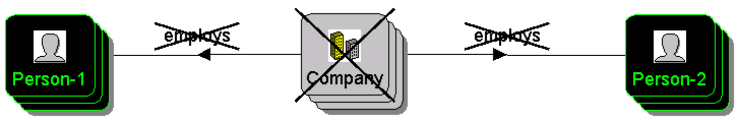


Figure 5.3.4.5. A path filter that matches paths connecting the object matched by the object constraint Person–1 and that matched by the object constraint Person–2 that do not represent that the people are employed in the same company.

In this particular example all the object and link constraints that were added to the path filter have been inverted; however, this need not be the case. For example, the object constraint matching any object representing a company need not be inverted. The path filter would then match any path connecting the object matched by the object constraint Person–1 and that matched by the object constraint Person–2 involving an intermediate object representing a company, and where the links in the path do not represent that a company employs a person.

5.3.5 Object Comparison Constraints

An object comparison constraint connects two object constraints and specifies that the objects matched by the constraints must differ or should be identical. The comparison performed is indicated by the label of such a constraint, as illustrated in figure 5.3.5.1.

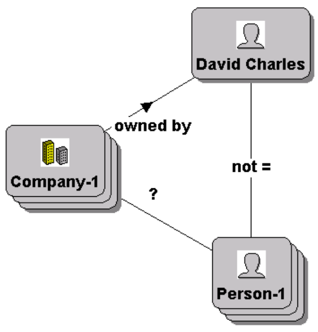


Figure 5.3.5.1. A filter pattern query that displays people linked to a company that David Charles owns, and the associated companies and links. Link constraints that match any type of link are labelled with a question mark; those that do not take the direction of links into consideration have no arrow.

5.3.6 Excluded Constraints

Constraints that match objects, links or paths may be *excluded*. If a constraint is excluded an object, link or path matched by the constraint is not displayed in the chart representing those matched by the pattern, unless it is also matched by another non-excluded constraint or was already in the chart. Excluded constraints are coloured light grey; see figure 5.3.6.1.

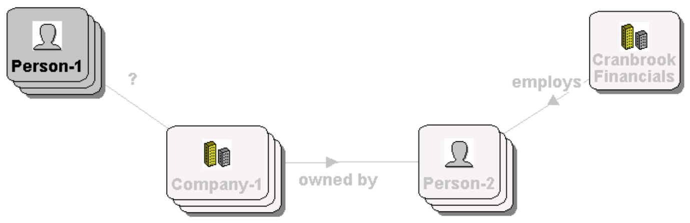


Figure 5.3.6.1. A filter pattern that displays people linked to a company owned by an employee of Cranbrook Financials. This pattern is a modified version of that in figure 5.3.3.1.

5.3.7 Visual Properties of Object & Link Constraints

Object constraints that match a specified object in the database have a set of visual properties similar to those defined for object types in the database schema, see section 3.4.2. When such a constraint is added to a FPQ it is by default given the same values for these properties as set for the object type of the associated object. These properties may be edited.

Object constraints that match instances of a specified object type are given by default the same set of visual properties as set for that object type; these may not be edited. As illustrated in figure 5.3.7.1, object constraints that match any object, regardless of type or information stored directly about the object, are coloured grey, have no label and display an icon containing a question mark.

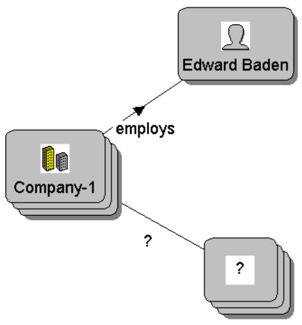


Figure 5.3.7.1. A filter pattern query that displays objects linked to companies of which Edward Baden is employed, and the associated companies and links.

Link constraints have a set of visual properties similar to those defined for link types in the database schema, see section 3.4.3. These properties have default system-defined settings. When instances of a specified link type are matched by a link constraint the visual properties specified for the link type are used.

5.3.8 Redundant Constructs

As illustrated in figure 5.3.4.4, the ability to declare link constraints indirect in the path filter language does not provide any additional expressive power to the language. This is also true of adding link constraints to the FPQ language. The restrictions imposed by link constraints can be specified using path constraints with appropriate path filters. They are only provided so that only a minimum number of concepts have to be learnt to form useful queries.

5.4 The Operational Semantics of a Filter Pattern Query

5.4.1 Introduction

The following subsections define the operational semantics of a FPQ so as to allow the meaning of a query to be determined by the user and to permit the set of queries that may be expressed in the FPQ language to be ascertained. Several methods have been proposed for defining the semantics of a visual language; these include Relation Grammars [CRIM90], Positional Grammars [OREF92] and Picture Layout Grammars [GOLI89]. The semantics of the FPQ language are defined in an operational manner due to the conciseness of the definition produced.

The remainder of this section is organised as follows. Section 5.4.2 abstracts from the graphical representations used for FPQs and path filters presented in section 5.3 to equivalent textual representations. The section then concludes by defining two algorithms, one for evaluating a FPQ, the other for determining if a path matches a path filter; both of the algorithms use the textual representations described in section 5.4.2.

5.4.2 Textual Representations for Filter Pattern Queries & Path Filters

5.4.2.1 A Textual Representation for a Filter Pattern Query

A FPQ may be defined as a 4-tuple (OC, OCC, LC, PC) associated with a database D and a database schema S. The tuple elements are defined below using OBS to represent the set of object identifiers in D, OTS to represent the set of object types defined in S, and LTS to represent the set of link types defined between object types in S:

- OC is a set of object constraints. Each O ∈ OC is a 5-tuple (ID, OID, OT, FC, E). ID is a system generated unique identifier for O in OC. If OID is non-null only the object with identifier OID ∈ OBS will be matched. If OID is null and OT non-null only an instance of the object type OT ∈ OTS will be matched by O, otherwise, any object may be matched, regardless of its type. If OT and FC are non-null, only an instance of OT satisfying the filter conditions FC will be matched by O. Only if E is true is O excluded.

- OCC is a set of object comparison constraints. Each O ∈ OCC is a 3-tuple (OC1, OC2, OP). OC1 ∈ OC and OC2 ∈ OC are the object constraints that O connects. OP ∈ {'=', '≠'} specifies the comparison performed between the objects currently matched by OC1 and OC2.

- LC is a set of link constraints. Each L ∈ LC is a 6-tuple (OC1, OC2, DIR, LT, FC, E). L is defined from OC1 ∈ OC to OC2 ∈ OC. If DIR is true only links defined from the object currently matched by OC1 to the object currently matched by OC2 will be matched, otherwise, links defined in any direction between these two objects will be matched. If LT is non-null only instances of LT ∈ LTS will be matched, otherwise, all links defined in an appropriate direction between these objects, regardless of type, are matched. If LT and FC are non-null, only instances of LT that satisfy the set of filter conditions FC will be matched. Only if E is true is L excluded.

- PC is a set of path constraints. Each P ∈ PC is a 5-tuple (OC1, OC2, LEN, PF, E). P is defined from OC1 ∈ OC to OC2 ∈ OC. If LEN is –1 then only the shortest path(s) connecting the object currently matched by OC1 to the object currently matched by OC2 is matched, otherwise, just paths with LEN or less links are matched. If PF is non-null, only paths that satisfy the path filter PF will be matched. Only if E is true is P excluded.

5.4.2.2 A Textual Representation for a Path Filter

A path filter may be defined as a 4-tuple (OC, LC, SS, FS) associated with the same database D and database schema S as the related path constraint. The tuple elements are defined below using OBS to represent the set of object identifiers in D, OTS to represent the set of object types defined in S, and LTS to represent the set of link types defined between object types in S:

- OC is a set of object constraints. Each O ∈ OC is a 5-tuple (ID, OID, OT, FC, INV). ID is a system generated unique identifier for O in OC. If OID is non-null only the object with identifier OID ∈ OBS will be matched. If OID is null and OT non-null only an instance of the object type OT ∈ OTS will be matched by O, otherwise, any object will be matched, regardless of its type. If OT and FT are non-null, only an instance of OT that satisfies the set of filter conditions FC will be matched by O. Only if INV is true is O inverted.

- LC is a set of link constraints. Each L ∈ LC is a 7-tuple (OC1, OC2, DIR, LT, FC, INV, IND). L is defined from OC1 ∈ OC to OC2 ∈ OC. If DIR is true only links defined from the object currently matched by OC1 will be matched, otherwise, links defined in any direction between this object and that currently matched by OC2 will be matched. If LT is non-null only instances of LT ∈ LTS will be matched, otherwise, all links defined in an appropriate direction between these objects, regardless of type, are matched. If LT and FC are non-null, only instances of LT that satisfy the set of filter conditions FC will be matched. Only if INV is true is L inverted. Only if IND is true is L declared indirect.

- SS ∈ OC and FS ∈ OC are the start and end states.

5.4.3 The Operational Semantics of a Filter Pattern Query & of a Path filter

5.4.3.1 The Operational Semantics of a Filter Pattern Query

The operational semantics of a FPQ are defined by the function in figure 5.4.3.1.1. The function takes two arguments: *fpq* represents the FPQ, and *d* the database that *fpq* is to filter. If *fpq* is to filter a link chart LC then *d* represents the objects and links displayed in LC. The function returns a link chart that contains all the objects and links in *d* that match *fpq*.

```
match(FilterPattern fpq, Database d)
{
        chart = new LinkChart()

        for( each set of objects in d that match the set of object constraints fpq.OC )
        {
                matched = true

                for( each occ ∈ fpq.OCC )
                {
                        if( occ is not satisfied )
                        {
                                matched = false
                        }
                }

                if( matched )
                {
                        obs = new Set()
                        ls = new Set()

                        for( each lc ∈ fpq.LC )
                        {
                                s = the set of links in d matched by lc

                                if( s is empty )
                                {
                                        matched = false
                                }

                                else
                                {
                                        Add to ls the links contained in s
                                }
                        }

                        for( each pc ∈ fpq.PC )
                        {
                                s = the set of paths in d matched by pc

                                if( s is empty )
                                {
                                        matched = false
                                }

                                else
                                {
                                        for( each path p ∈ s )
                                        {
                                                Add to obs the objects contained in p
                                                Add to ls the links contained in p
                                        }
                                }
                        }

                        if( matched )
                        {
                                Add to chart the objects and links contained in obs and ls
                        }
                }
        }

        return chart
}
```

Figure 5.4.3.1.1. The variables prefixed by *fpq.* represent the elements in the 4-tuple described in 5.4.2.1 that may be used to represent a filter pattern query.

5.4.3.2 The Operational Semantics of a Path Filter

A path through the network formed by the objects and links in a database may be represented as a list of object and link identifiers, where the list represents the objects and links traversed in the path in order. When represented as such a list, it may be determined if a path is matched by a specified path filter using the recursive procedure defined in figure 5.4.3.2.1.

```
matches(List path, Constraint constraint, Constraint previous)
{
        path element = remove first element from path

        if( constraint does not match path element )
        {
                return false
        }

        else if( (path is empty) and (constraint is the finish state) )
        {
                return true
        }

        else if( path is empty )
        {
                return false
        }

        else
        {
                if( path element is an object identifier )
                {
                        object constraint = (Object Constraint)constraint

                        for( each link constraint lc involving object constraint )
                        {
                                if( lc does not equal previous )
                                {
                                        if( matches(path, lc, object constraint) )
                                        {
                                                return true
                                        }
                                }
                        }

                        return false
                }

                else
                {
                        link constraint = (Link Constraint)constraint

                        oc = the object constraint that link constraint is defined from

                        if( oc equals previous )
                        {
                                oc = the object constraint link constraint is defined to

                        }

                        if( link constraint is declared indirect )
                        {
                                if( matches(path, oc, link constraint) )
                                {
                                        return true
                                }

                                else if( path has 3 or more elements )
                                {
                                        Remove first element from path

                                        return matches(path, link constraint, previous)
                                }

                                else
                                {
                                        return false
                                }
                        }

                        else
                        {
                                return matches(path, oc, link constraint)
                        }
                }
        }
}
```

Figure 5.4.3.2.1. An instance of the type *Constraint* may represent either a link constraint or an object constraint in a path filter.

The procedure should initially be called with the list representing the path in the parameter *path* and the object constraint that is the start state in the path filter in the parameter *constraint*; the third parameter should be the null value.

After the procedure has removed the first identifier from the list representing the path a check is made to determine if this element matches the object or link constraint currently considered, represented by the parameter *constraint*. If not, false is returned. Otherwise, further checks are made to determine if it can be decided given the value of *constraint* and the number of elements left to be processed in *path* that the path matches or does not match the path filter. If so, an appropriate value is returned.

Otherwise, if the path element currently processed is an object, then a recursive procedure call is made for each link constraint defined from, or to, the object constraint represented by *constraint*, providing the link constraint was not the last link constraint processed in the chain of calls made previously to the procedure. These calls represent the alternate routes that may be followed during evaluation. If any return true, true is returned, otherwise, false.

If the path element currently processed is a link, after determining the object constraint to be processed next a check is made to ascertain if the link constraint represented by *constraint* is declared indirect. If so, two recursive procedures calls are made, which allow all possible alternatives regarding how many links are matched by the link constraint to be considered. If any of these calls returns true, true is returned, otherwise, false. If the link constraint is not declared indirect, a recursive call is made that matches the next element in the path against the object constraint that is to be processed next. If this call returns true, true is returned, otherwise, false.

5.5 Expressive Power & Computational Complexity

5.5.1 Expressive Power

The FPQ language is not based upon a formalised algebra or logic from the literature, such as the relational algebra or first order logic. Therefore, the set of queries that may be expressed using the query language does not fit well into classification schemes such as that proposed by Chandra [CHAN88], which are often based upon such algebras and logics. An extended version of Chandra's classification scheme, commonly referred to as *Chandra's hierarchy*, is presented in [CATA97] and is displayed in figure 5.5.1.1.

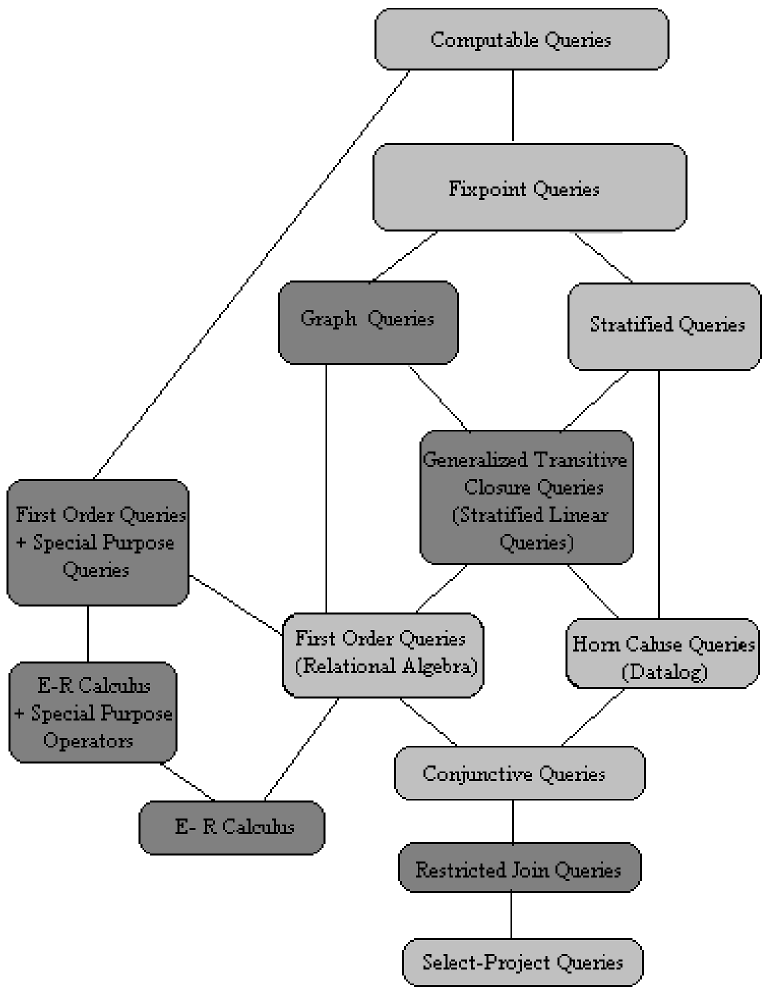

Computable Queries
Fixpoint Queries
Graph Queries
Stratified Queries
Generalized Transitive Closure Queries (Stratified Linear Queries)
First Order Queries + Special Purpose Queries
First Order Queries (Relational Algebra)
Horn Caluse Queries (Datalog)
E-R Calculus + Special Purpose Operators
Conjunctive Queries
E- R Calculus
Restricted Join Queries
Select-Project Queries

Figure 5.5.1.1. The enriched version of Chandra's hierarchy presented in [CATA97]. The classes added to the original hierarchy are shaded dark grey.

As demonstrated in section 5.4.3, the set of queries that may be expressed in the FPQ language is computable. This set overlaps with that which may be expressed using the browsing facilities described in sections 4.2.3 and 4.2.4, as such queries may be expressed more elaborately as a filter pattern. As discussed in section 1.3, queries retrieving connections between objects where how the connection is derived is not specified in the query cannot be expressed in first order logic. Therefore, the only class *appropriate* for representing the set of queries that may be expressed in the filter pattern query language is computable queries.

However, there are several simple queries that can be expressed in first order logic that cannot be expressed as a filter pattern, such as *display the names of companies that supply all blue parts*. Therefore, not all of the queries that may be expressed in query classes ranked below the class representing computable queries can be expressed as a FPQ, a consequence of the FPQ language being specifically designed to support network traversal, not data processing applications.

It is for this reason why a comparison between the expressive power of the FPQ language and other visual database query languages such as GraphLog (discussed in section 2.3.1) and QBD* [ANGE90] is not appropriate. There are queries that may be expressed in these languages that cannot be expressed in the FPQ language, and vice versa. The different data models used also makes such comparisons less valuable.

5.5.2 Computational Complexity

The cost of evaluating the set of queries that may be expressed in a given query class included in the classification scheme in figure 5.5.1.1 is greater than or equal to that of the classes defined below it in the hierarchy. For fixpoint queries, the cost is polynomial with respect to the size of the database. This is also true of the set of queries that may be expressed in the FPQ language, which can be computed using the function defined in figure 5.4.3.1.1.

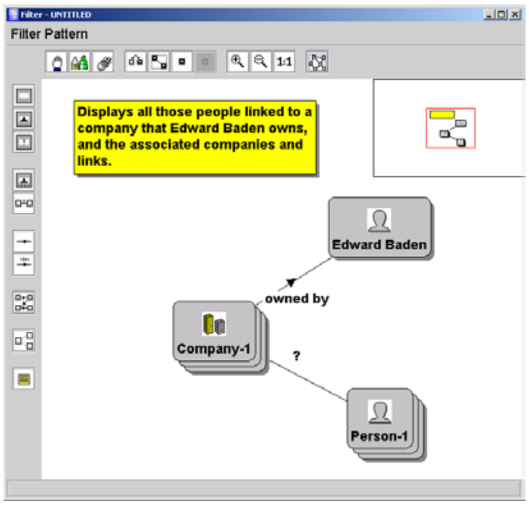
The cost of retrieving the set of values linked to a specified object is directly proportional to the number of links stored in the database. Therefore, iterating over all objects, or instances of a specified object type where information stored directly about the objects meet a set of filter conditions, executes in polynomial time with respect to the size of the database. Thus, the for loop that iterates over all sets of objects that match the object constraints in a FPQ will also execute in polynomial time. This is also true for finding the paths that connect two specified objects which match a given path filter.

The cost of finding the paths connecting two specified objects is polynomial with respect to the size of the database. The cost of determining if a path matches a path filter is polynomial with respect to the size of the path and path filter. A path through a database may be represented as a string of object and link identifiers, and a path filter as a finite state automaton (FSA) where information stored directly about an element in a path is also considered in a transition function. The cost of deciding if a string is accepted by a FSA is polynomial with respect to both the size of the string and automaton [AHO74].

5.6 Discussion

The FPQ language differs from previous database query languages as: it uses a data model that is appropriate for LA, it provides explicit support for analysing connections between objects without having to specify how connections are derived and provides a powerful visual sub-language for constraining these connections if required. This sub-language permits several types of connection to be matched in a single query and conditions that relate to link attribute information to be specified.

The implementation of this query language in EDVC allows FPQs to include text notes. As illustrated in figure 5.6.1, text notes may increase the user's ability to understand the semantics of a filter pattern, particularly when the pattern is supplied by a third-party.



Displays all those people linked to a company that Edward Baden owns, and the associated companies and links.

Figure 5.6.1. A filter pattern editor displaying a filter pattern that contains a text note.

FPQs may be saved for subsequent further development. When a FPQ is reopened a number of checks are made. First, to verify that all object constraints that match a specified object in the database are associated with an object still stored in the database. Second, that all object and link constraints that only match instances of a specified type in the database schema are associated with a type that is still defined in the schema. If there are path constraints in the pattern similar checks are then made for any associated path filters. If any of these checks fails the user is informed that the pattern is not valid for the database currently being analysed and may not be opened.

Chapter 6 – A User Evaluation of EDVC

6.1 Introduction

One of the novel aspects of EDVC is that it provides user interface support for adding objects and links to a chart using a mixture of database interaction styles, either using the browsing operations described in sections 4.2 and 4.3, or using the filter pattern query language, which is described in chapter 5. The level of knowledge required to productively use the browsing operations is less than that which is required to use the query language, as the query language requires a greater understanding of the database schema. Empirical results [CATA00] indicate inexperienced database users have difficulty in understanding database schemas, which would have an impact on how intuitive they find using filter patterns and other database query languages.

The motivation for conducting the user evaluation of EDVC was to assess if the intended user community would find one style of interaction more intuitive than the other, to assess if the system could be productively used by this user community to analyse data such as that would which would be encountered in practice, and to identify areas of weakness in the operations provided and in the style of interaction supported in the system.

6.2 The Experimental Procedure

6.2.1 Form of Evaluation

No previous database query system has provided a data visualisation mechanism appropriate for supporting LA, with both a range of browsing facilities for essential LA operations and a query language with the necessary expressive power to retrieve connections of an undefined length and structure from a database representing some form of network. Thus, a comparison of user performance with EDVC and another system, as in the majority of user evaluations of database query interfaces [CATA95, CHAV97, MURR00], would have been of little value and was not performed.

6.2.2 Facilities Assessed

The explorer interface provides a number of facilities, not all of which could be tested due to restrictions concerning the time and cost of the evaluation. The evaluation focused upon the facilities that allow information to be retrieved from the database, as this is the primary purpose of the interface. The relatively small number of constraints that may appear in a filter pattern allowed the use of all such constraints to be evaluated, to varying degrees. A further objective of the evaluation was to obtain empirical results concerning the use of facilities for making the data retrieved more comprehensible.

6.2.3 The Subjects

Desktop computers are commonly used in many commercial and governmental organisations. Therefore, it may be assumed that the majority of the individuals who would use a system supporting LA, which may not necessarily be for law enforcement purposes, are comfortable, at least at a basic level, with interacting with computerised systems using a graphical user interface. This is not to say that they have previous experience of using a database query language, as the majority will not.

The subjects taking part in the evaluation reflected this. All had experience of interacting with software providing a graphical user interface, but no previous experience of using either a database query language, a programming language or LA. As illustrated in table 6.2.3.1, the subjects were from a mixture of educational backgrounds, ranging from early school leavers to postgraduates students.

As discussed by Peterson in [PETE98], the International Association of Law Enforcement Intelligence Analysts conducted a survey in 1993 of the job descriptions for intelligence analysts at its member agencies in the state of Florida in the USA. 67% of these required a bachelor's degree or four years law enforcement experience that required collating, assembling and analysing pertinent facts, and the ability to present them in a clear and concise written report. However, the standard for intelligence analysts in other countries is not at this level; for example few agencies in either Canada or Australia require their analysts to obtain degrees. Nonetheless, using subjects with the educational backgrounds listed in table 6.2.3.1 would seem representative of those that would use EDVC in practice to analyse criminal data.

Demographic Dimension	Descriptive Statistics
Age	Minimum 18 Years Maximum 61 Years Mean 28.75 Years
Gender	12 Male 8 Female
Educational Background	2 High School 2 Passed Professional Exams 4 Under-Graduate 12 Graduate
Profession	13 Professionals 7 Students

Table 6.2.3.1. Demographic statistics for subjects. The demographic statistics regarding educational background relate to the highest academic level achieved.

Law enforcement officers were not used in the evaluation for reasons of practicality and because any experience that they may have had with LA products could have influenced the results obtained due to familiarity with the interaction style of a particular product. Research also suggests that their acceptance of new technology may be greatly influenced by the opinions of their colleagues [HU03].

In order to obtain reliable results within a reasonable time frame whilst minimising costs twenty subjects were used, all volunteers. This allowed the level of sensitivity that the evaluation results had to minor differences in the experimental environment to be reduced, and prevented any apathy from the subjects influencing the results to any significant degree.

6.2.4 Pre-Evaluation Questionnaire

Each subject was required to fill in a pre-evaluation questionnaire, which like all documents relating to the user evaluation is included in appendix A. The questionnaire requested the following information: contact details, occupation, educational background, familiarity with software providing a graphical user interface, knowledge of database technology and familiarity with procedural programming languages.

The level of intelligence of the subjects in a user evaluation of a query language can have an effect on the results obtained from the evaluation [REIS81]. Therefore, it is important to ascertain what level and type of education a subject has received, as well as their occupation as this may indicate the presence of a certain skill or trait that may influence the results obtained.

As the system provides a graphical user interface ascertaining the level of experience that each subject had with using software providing such an interface was necessary. Particular attention was paid to the level of experience each subject had with using World Wide Web browsers as the browsing style of interaction used within these systems has some similarities with that supported in the explorer interface.

Previous database query interface experience may provide background knowledge that allows a subject to acquire the skills necessary to use EDVC in a shorter timeframe than someone with no such experience. However, there is a significant difference in the level of skill and knowledge required to scroll through the contents of a table in a relational DBMS product and that required to form a query in a language such as SQL. Therefore, if a subject did have previous database experience the nature of this experience was ascertained.

Previous database query language evaluations [CATA95] indicate that subjects who have used procedural programming languages like C and Java find using a declarative query language more difficult than people who are not familiar with these programming languages. Therefore, the level of familiarity that each subject had with such languages was ascertained.

6.2.5 Method of Evaluation

Table 6.2.5.1, taken from [REIS81], lists several known methods for evaluating the ease of use of a query language. Query writing, query reading and query comprehension were chosen for use in the evaluation of EDVC as these methods allow a measure of how proficient a subject is at forming queries to be obtained, as well as an indication whether a subject is answering questions by rote.

Task	Description
Query Writing	Users are given a question stated in English and required to write a query in the given query language
Query Reading	Users are given a query written in the query language and asked to write a translation into English
Query Interpretation	Users are given a query in the query language and a printed database with data filed in. They are asked to find the data asked for by the query
Query Comprehension	Users are given an English 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 English that would solve the problem. The questions should be answerable from the database

Table 6.2.5.1. Methods for evaluating the ease of use of a query language.

6.2.6 Format of the Questions

Each subject answered two groups of questions within an unrestricted time span. One group concerned the explorer interface, the other concerned filter patterns. Each group was subdivided into two subgroups. The first subgroup required query writing, the second, with the exception of the final two questions concerning filter patterns, required query comprehension.

Query reading was used for the final two questions concerning filter patterns, which evaluated the subject's understanding of path filters. The pilot evaluation, described in subsection 6.2.14, indicated that the use of query comprehension in these questions would cause incorrect answers to be obtained through human error as a result of having to perform a relatively lengthy mechanical procedure.

In order to obtain a reliable set of results within a reasonable time frame it was decided that each sub-group would contain six questions, which seemed reasonable given the validity of the results obtained from previous query language evaluations using a fewer number of questions. To prevent subjects less experienced with formal testing being discouraged by having to initially perform a complex task which they may be unable to complete, even if they are capable of answering the majority of the remaining questions, the perceived complexity of the first question in each sub-group was relatively low and the complexity of the remaining questions in the sub-group increased with the order in which the questions were asked.

The complexity of each question was assigned according to the mental effort required for the user to compute the query, not according to the time required to specify the query. Although metrics have been defined for assigning complexity to queries based upon relational algebra operators, such as that reported in [CATA95], these are not applicable to the queries that may be formed using EDVC due to the difference in the query operators and data model supported. Therefore, an estimate was made using comments provided by users.

6.2.7 The Questions

The first sub-group for the explorer interface required the subject to perform the following tasks: add an object to a link chart, display the objects linked to an object, display the objects connected to an object, display the objects linked to a number of other objects, display the paths connecting two objects, and display the objects connected to a group of objects.

Each question focussed upon a query facility directly supported in the interface, the use of which was the most efficient way of performing the task, although, all these questions could be answered using more than one problem solving strategy. Each such question was associated with a link chart, which was displayed before the subject attempted to perform the required task. The query executed at any one point during LA typically depends on the data already displayed in the chart. Thus, providing an initial chart, which may be empty, reproduced these conditions and allowed each subject to answer a question even if they had not answered previous questions correctly.

The first sub-group for filter patterns required the subject to perform the following tasks: construct a filter pattern that would display all objects of a given type, construct a filter pattern that would display all objects of a given type whose attribute information met a given set of conditions, construct a filter pattern that would display all objects belonging to a given set of types that are linked in a particular manner, construct a pattern that would display all objects belonging to a given set of types whose attribute information met a given set of conditions and are indirectly linked in a given manner, and construct a path filter for matching paths of a specified structure.

All these questions involved using the filter pattern editor. It was recognised that an additional paper and pencil evaluation would allow just the query language and not the query language and the interface to be evaluated. However, the evaluation needed to be performed within a reasonable time frame and budget, and a choice had to be made between a paper and pencil test and an on-line test. An on-line test was used as the support provided by the system for constructing filter patterns can greatly decrease both syntactical and semantic errors, which may influence the results of a paper and pencil test. It was also perceived that the explorer interface would be used in practice to validate a filter pattern by visually checking the objects and links retrieved, which was the case in the pilot evaluation.

The questions in the second sub-group for each interface focussed upon the same tasks that were used in the first sub-group and used the same database.

The tasks the subjects were asked to perform were directly motivated by those investigators might typically wish to perform in practice. The tasks relating to the explorer interface that required the user to display the paths connecting two objects, or to display the objects connected to a specified object, are directly related to the fundamental data analysis operations LA is based upon (discussed in section 1.1). The remaining two tasks for this interface, which required the subject to display the objects linked to a number of other objects and to display the objects connected to a group of objects, were motivated by our experience of analysing data such as that used for the tutorial in the user evaluation.

The tasks relating to filter patterns were also motivated by how LA is used in practice. Some searches through the data gathered thus far in an investigation may be *seeded*, such as when a potential suspect had been identified and an investigator wishes to see the connections the suspect has to other objects of interest in the investigation. However, witness descriptions are often used to construct an initial list of potential suspects before such analysis is used. Therefore, filtering objects according to attribute information is highly relevant.

Finding connections between objects where the objects and possibly the connections meet a given set of criteria is also common in LA, particularly, as discussed by Peterson [PETE98], for analysing financial data in organised crime investigations. There are particular types of connection that often occur in this context, such as two distributors in a money-laundering scheme being linked through a corporate structure. Therefore, the tasks that relate to constructing filter patterns for finding connections between objects where the type of link, and the structure of the connection, are specified could also prove of value in practice.

6.2.8 Evaluation Database

Each subject was provided with a sheet that contained a textual description of the database to be used in the evaluation, and a link chart representation of a subset of the contents of the database to illustrate how the data stored would be displayed on screen. The database used for the evaluation, which was fictional, concerned a set of companies that were suspected of being fraudulent, and the people who owned and were employed in the companies. A criminal database of this structure was used as it was realistic and its contents would be familiar to the subjects, allowing a meaningful evaluation to be performed.

The subjects were not given a copy of the database schema. It is doubtful given the average level of database experience in the intended user community that in practice a user would have the background knowledge to understand the concept of a database schema, or the desire to acquire this knowledge. Empirical evidence [CATA00] indicates that in general users have difficulty understanding database schemas, even when presented in the *user-friendly* form of an Entity Relationship Diagram. However, a diagrammatic representation of the schema was available if required via the database structure browser.

6.2.9 Format of the Evaluation

The query system was introduced to each subject in two separate tutorial sessions, described in subsection 6.2.11, one for each of the interfaces evaluated. The order in which the interfaces were introduced to each subject was constant; the explorer interface was introduced first then the filter pattern editor. This was done as it was envisaged that the subjects, and users in practice, may wish to display the data retrieved by a filter pattern to validate that the correct pattern had been constructed, which was the case in the pilot evaluation.

The tutorial for each interface was followed by the questions relating to that interface, so as not to effect the results obtained for the explorer interface in relation to those obtained for filter patterns. Following the questions relating to the explorer interface, and before filter patterns were introduced, there was a break lasting ten minutes.

6.2.10 Observing the Evaluation

During the evaluation the actions of the subjects were directly observed, and logged by the system, so as to allow their problem solving strategies to be studied and compared. The use of video was considered for this purpose but was deemed too expensive both in terms of cost and time as each session would subsequently have to be transcribed. The time taken to complete each question was also noted for each subject. This was used in conjunction with the problem solving strategy to provide an indication of how difficult the subject found using a facility in the explorer interface or a filter pattern construct.

The subjects were also required to indicate how confident they were that they had answered a question correctly, using a sliding scale of 1, very confident, to 5, no confidence. This was used to assess if a subject was answering a question by guessing; it can also be a useful tool for detecting overconfidence, which may lead to incorrect answers being given as a result of a lack of attention. The subjects were made aware that there was potentially more than one method to answer a query writing question, and were instructed that their response for such questions should reflect their confidence that the correct data was displayed, or that the filter pattern constructed would display the correct information, not that they had performed the task using the optimal method.

6.2.11 Method of Instruction

The interfaces were introduced to each subject via two interactive tutorials conducted on a one-on-one basis with the instructor. The tutorials involved the subject analysing a demonstration database by following the instructions in a tutorial booklet, which was made available during the evaluation so as to simulate real-world conditions and to relieve the memory load placed on the subjects.

This aim of the tutorial was for each subject to have a fixed set of skills, those gained through following the actions in the tutorial booklet, before completing the evaluation. This could be determined with a greater degree of accuracy than in a lecture theatre or lab session with a group of subjects, especially as the subjects differed in their level of education, which could influence their ability to learn effectively in large groups.

The demonstration database differed from the evaluation database and concerned a set of intelligence reports submitted by a set of police officers, the people and locations mentioned within those reports, and the associates of those people and the locations that they frequent.

This reduced the probability that the subject was answering questions by rote. The period of time taken to introduce each interface was approximately 45 minutes, although this was not fixed to ensure that each subject had completed the tutorial. There was little difference in the length of time taken by each subject.

6.2.12 Grading the Answers

The method for grading answers was a modified version of that used in [REIS75]. Each answer was graded as either correct, essentially correct but with a minor data error, incorrect, or not attempted. The definition of what *essentially correct but with a minor data error* meant depended on whether the question required query writing or query comprehension.

For query writing it was defined as an answer with a correct structure but where a minor error had been made when entering/selecting a query parameter, such as entering *"Smitg"* instead of *"Smith"*. For query comprehension it was considered to be an answer that clearly showed the subject had understood the semantics of the question but where a minor mistake has been made while highlighting the objects and links, such as failing to highlight an appropriate link or object. The pilot evaluation indicated that some leniency was needed as the lengthy repetitive nature of tasks meant that these mistakes could easily occur.

6.2.13 Post-Evaluation Interview

After each subject had completed the final sub-group of questions they were interviewed to ascertain their opinions and thoughts concerning the system. Each subject was asked a pre-determined set of questions, included in the post-evaluation *questionnaire*, in addition to others pertaining to how the system was used during the evaluation, and how they felt that the level of training they had received prior to the evaluation may have effected their performance. Interviews were used instead of questionnaires as reluctance by a subject to write complex explanations can prevent useful information from being obtained.

6.2.14 Pilot Evaluation

A pilot evaluation was performed using the proposed evaluation method on separate occasions with two research students, both computer scientists with no previous experience of LA. The aims of the pilot were to access the method of instruction, debug the teaching material and questionnaires, and to validate the evaluation procedure.

In addition to minor alterations regarding the wording of the material supplied it was suggested that each subject should be provided with a sheet listing and describing the menu options and toolbar icons representing browsing operations in the explorer interface. Additionally, as described in section 6.2.6, the use of query comprehension for evaluating the subject's understanding of path filters was not considered appropriate.

6.3 The Evaluation Results

The quantitative results are summarised in table 6.3.1, 6.3.2, 6.3.3 and 6.3.4. C, EC, IC and NA refer to the grade assigned to the answers given by the subjects. They are mnemonics for correct, essentially correct but with a minor data error, incorrect, and not attempted. The cells in these columns give the number of answers (out of 20) assigned the corresponding grade for each question. Time and confidence refer to the arithmetic mean of the number of seconds it took the subjects to complete each question, and the arithmetic mean of the level of confidence that the subjects had that the answer they gave was correct.

Question	**C**	**EC**	**IC**	**NA**	**Time**	**Confidence**
1	20	0	0	0	41.20	1.15
2	19	0	1	0	133.50	1.35
3	16	3	1	0	150.90	1.45
4	17	1	2	0	172.70	2.15
5	19	0	1	0	133.55	1.70
6	16	0	4	0	200.85	2.35

Table 6.3.1. Expressing queries using the explorer interface.

Question	**C**	**EC**	**IC**	**NA**	**Time**	**Confidence**
1	20	0	0	0	48.10	1.50
2	19	0	1	0	103.90	2.20
3	19	1	0	0	35.05	1.15
4	20	0	0	0	45.85	1.45
5	20	0	0	0	24.70	1.15
6	16	0	4	0	121.30	2.55

Table 6.3.2. Query comprehension, explorer interface.

Question	C	EC	IC	NA	Time	Confidence
1	20	0	0	0	53.45	1.05
2	19	0	1	0	274.15	1.90
3	18	0	2	0	172.40	1.95
4	2	9	9	0	297.70	2.85
5	15	0	3	2	158.60	2.95
6	8	0	11	1	245.35	3.80

Table 6.3.3. Expressing queries as a filter pattern.

Question	C	EC	IC	NA	Time	Confidence
1	20	0	0	0	23.05	1.00
2	14	0	6	0	52.40	1.50
3	17	2	1	0	57.45	1.55
4	18	0	2	0	58.15	1.70
5	13	0	5	2	131.00	3.40
6	15	1	4	0	97.60	2.55

Table 6.3.4. Query comprehension, filter patterns.

6.4 Analysis of the Evaluation Results

6.4.1 The Explorer Interface

The percentage of answers graded correct, and essentially correct but with a minor data error, for the query writing questions was encouragingly high. As the grades for the query comprehension questions were similar and the time required to complete the tasks were relatively lengthy it can be concluded that the semantics of the queries formed were well understood and that the tasks were considered non-trivial. This is also reflected by the responses given concerning the subject's level of confidence that the answer given was correct, which generally decreased as the perceived complexity of the question increased.

The high percentage of correct answers for the first question was not unexpected. The task was designed to help subjects gain confidence with independent use of the system and with the evaluation procedure; this was also true of the corresponding question for filter patterns. Although, the task was not totally undemanding for someone with little familiarity with the location and purpose of the facilities provided.

This unfamiliarity with the location and purpose of facilities is reflected in the length of time subjects took to complete the query writing tasks. Several subjects remarked that they knew what data should be displayed but could not remember where the required facility was, even with the use of the tutorial booklet. The primary reason given for this was nerves, which were incurred by many of the subjects who had not been involved in a formal test for some time.

The percentage of incorrect answers for the questions focusing upon the use of the facility for retrieving objects connected to a group of objects (the final question in each of the relevant sub-groups) was significantly higher than those for other questions. However, the percentage of correct answers given for these questions is still high. Thus, although the majority of subjects can form such queries the mental effort required to do so is greater than for other queries directly supported in the interface, and may require more attention during the training period. This is also indicated by the time required to answer these questions in relation to others for the interface.

Although the percentage of correct answers given for the fourth question in the query writing sub-group is high, the level of confidence that the answer given was correct is relatively low, and the time required to complete the question is high considering the actions that were needed to perform the task. The reason for this was obtained by analysing the subjects' actions and through the post-evaluation interviews.

The method for computing the answer to this question requires inspecting all objects of an appropriate type, then ascertaining for each object inspected if the information stored directly about the object meets certain conditions. This style of information retrieval is similar to that found in traditional database query languages, not the browsing style of interaction that is the primary purpose of the explorer interface, which the subjects were most familiar with.

During the post-evaluation interviews all subjects remarked that they found the browsing style of interaction very intuitive. 25% of subjects started, and in some cases completed, answering the question by manually performing the method used by the system to compute the query. This was more fitting with the browsing style of interaction they were most familiar and comfortable with, as discussed in section 6.4.3. The length of time required to perform the necessary actions was lengthy and impacted upon the subject's confidence that they had completed all the actions correctly.

6.4.2 Filter Patterns

The percentage of correct answers for the first three query writing questions, which focus upon matching objects of a given type that met a given set of filter conditions, and matching objects linked in a given manner, are all high. Thus, it may be concluded that these relatively simple queries are well supported by both the query language and the interface. This is also indicated by the responses given in the post-evaluation interviews, which all stated that the grids used to combine filter conditions are conceptually easy to understand and easy to use in practice.

The percentage of incorrect answers for the final three questions is significantly higher, and the level of confidence that the answer given was correct, noticeably lower. The fourth question requires the construction of a pattern containing two object constraints indirectly linked via an intermediate object constraint and two link constraints. This is similar to the third question, which requires the construction of a pattern containing two object constraints connected via a link constraint, but was found significantly harder. The percentage of correct answers for the corresponding query comprehension questions, three and four, are however similar.

The subjects remarked during the post-evaluation interviews that they generally found the explorer interface more intuitive to use than the filter pattern editor, primarily for the reason expressed in the following quote: “in the explorer interface you are dealing at the factual level, whereas with filter patterns you are dealing at a conceptual level, which is less familiar and takes longer to get used to”. The mental effort required to deal at the conceptual level increases when longer paths through the database are expressed, which is also indicated by the results obtained from the use of path filters, discussed below. This impacts upon query writing but not query comprehension as the subject is still dealing at the factual level, explaining the above.

There is also a relatively high number of answers graded *essentially correct but with a minor data error* for the fourth question. Answers were graded as such if the pattern constructed had the correct structure and filter conditions specified but did not exclude the object and link constraints needed for filtering, but not for displaying elements in the resulting chart. This is partially a result of the training given to the subjects, which primarily focussed upon constructing patterns where all the objects and links matched by the constraints are displayed in the resulting chart. This made the subjects over-familiar with the process, which was confirmed during the post-evaluation interviews.

The grades for the final two query writing questions, which concerned the use of path filters, were also noticeably lower than those obtained for the first three. This was particularly true for the sixth and final question, where there was a significant difference between the percentage of correct answers for the question and for the corresponding query reading question, number six. This may indicate the provision of an inappropriate interface for constructing path filters or a lack of training. The former would not seem likely due to the similarity in the facilities provided in the interface to those used in the filter pattern editor, and because the answers obtained for question five were similar for both query writing and query reading. However, as discussed in subsection 6.6, this requires further evaluation.

The times recorded for completing the query writing questions were not proportional to the perceived complexity of the question, but does bear some relation to the actions required to complete the questions. Further analysis of the actions required using an empirical measurement tool such as Fitts' [FITT54] Law would be needed to be validate this.

6.4.3 General Points of Interest

As in the pilot evaluation, the subjects made extensive use of the explorer interface for validating the filter pattern constructed by checking the data retrieved. 60% of all answers to such questions used this technique and only one subject did not use the technique at all. This indicates that the subjects were generally not at ease with constructing filter patterns, which is validated by the lower levels of confidence reported by the subjects that their answer was correct than in the explorer interface.

Nonetheless, the vast majority of subjects stated during the post-evaluation interviews that the concept of a filter pattern was readily understandable and their use of this query tool would improve with more training. Thus, while filter patterns may be considered usable for expressing relatively simple queries, more complex queries specifying how the objects matched should be linked via indirect connecting paths are not immediately intuitive to the inexperienced database user and require a suitable training period.

The length of the training period for the explorer interface was less of an issue. All subjects stated that the style of interaction provided by the interface was very intuitive and were comfortable with using facilities for increasing the comprehensibility of chart, such as the layout facilities. The explanation given by several subjects for why this style of database interaction was intuitive is given in the following quote: "it was similar to how you would solve the problem using pen and paper". This analogy would seem appropriate. Unlike the adherence to a user interface metaphor, which is often of little benefit in visual query languages [BLAC99], this does not restrict the manner in which a task is performed.

6.5 Critical Analysis of the Evaluation & Future Work

Although the tutorial period was not constrained by time, the subjects all seemed to have an upper limit on the amount of information they could absorb in a single session. This limit seemed appropriate for acquiring the skills needed to use the explorer interface but inappropriate for acquiring those needed to form filter patterns of any great complexity. This issue seemed to particularly effect the presentation of the path constraints, which was the final concept introduced to the subjects.

Conducting the tutorial sessions for each interface on separate occasions for each subject may have resulted in more reliable results being obtained. A further evaluation focussing upon the use of path filters would prove useful. Such an evaluation could compare the graphical notation used for path filters with alternatives, both textual and graphical, such as those used in the evaluation described in section 2.3.2.

The use of an on-line test proved useful for identifying elements in both the query language and interface requiring further development or further evaluation. However, an additional paper and pencil evaluation would allow more accurate results concerning the query language to be obtained.

The number of subjects used in the evaluation was limited, and none had previous experience of using a traditional database query language. A further evaluation with a greater number of subjects, a subset of whom should have experience of using such languages, would prove constructive and allow useful comments regarding comparisons with aspects of other query languages to be obtained.

EDVC has yet to be used in a commercial or governmental environment, although, the system has been used to analyse real-world data by researchers within Birkbeck College. Also, during the user evaluation, the dataset used in the tutorial was a subset of that of a real intelligence database, where details about individuals and reports had been changed to provide anonymity.

Conducting an evaluation with intelligence analysts such as forensic accountants or members of the law enforcement community using real data, and comparing the time taken to analyse data using EDVC with that taken to do so manually or with a system such as Watson, would also prove of value. However, the comments provided in section 6.2.3 regarding how the results of a user evaluation with law enforcement community officers may be influenced by the subjects' previous experience of LA software, or the opinions of their colleagues, should be considered when analysing the results obtained.

6.6 Discussion

The user evaluation of EDVC was used to identify areas of weakness in the user interface and to obtain original research results. These results concerned whether a browsing style of interaction not requiring the user to have explicit knowledge of the database schema is more intuitive for inexperienced users to use than a query language where greater knowledge of the schema is required.

The results from the evaluation indicate that: the style of interaction provided by EDVC is well suited to supporting the intended user community, subjects found the browsing style of interaction more intuitive than the query language style of interaction, simple but useful queries may be formed using the filter pattern query language with a small amount of training, the user interface for constructing path filters needs to be improved or that a greater level of training needs to be provided.

A comparison of the results obtained for the two styles of database interaction supported in EDVC would seem to validate the empirical results reported by Catarci [CATA00], which concerned inexperienced database users finding database schemas difficult to understand. They would also seem to indicate that removing the need for a user to explicitly possess knowledge of the database schema allows inexperienced users to be more productive when using a database query interface.

Chapter 7 – Implementation

7.1 Introduction

LA products such as Watson (described in section 2.2.1) provide access to the data stored in an existing relational database. As there is a difference between the data model of a relational database and the model of data used in LA to find connections between objects this requires the user to analyse the existing relational database in order to determine how objects and links are represented, both when initially configuring LA products and also whenever the database schema is modified. Hence, these products must be configured to the schema of the database to be accessed, and reconfigured, usually manually, whenever the schema is modified.

The need to reconfigure LA products can greatly increase their system maintenance costs as incremental schema development is often required in criminal investigations, and converting between data models may also adversely affect performance. Whilst an important motivation for the implementation of EDVC was to allow the design produced from the criteria specified in section 1.4 to be evaluated, the implementation was also motivated by the need to reduce the setup and maintenance costs of existing LA products.

7.2 Overview of the System Architecture

EDVC is implemented using version 1.3.1 of the Java programming language. This not only allowed the system to be implemented using a single programming language, but allows EDVC to be used on any platform that supports the standard edition of the Java 2 platform. The underlying DBMS Sentences (described in section 2.2.4) is also implemented using Java, and also runs on the standard edition of the Java 2 platform.

As shown in figure 7.2.1, the software components within the system are logically divided into those used in the user-interface and those for retrieving data from the DBMS; the later support browsing operations in the explorer interface and the evaluation of FPQs. The values passed to the components that support data retrieval operations, and the values returned from these components, are independent of the DBMS. These values represent the identifiers of object or links in a link chart, are values representing parameters for browsing operations, or represent a graph corresponding to a filter pattern (see section 7.4). This, allows EDVC to be used as an interface to any data source for which implementations of the retrieval operation components are provided.

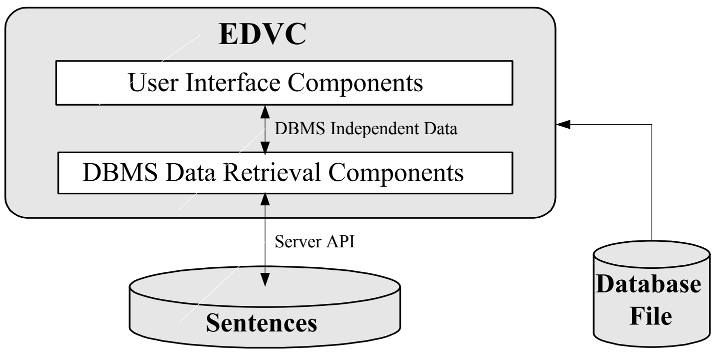


Figure 7.2.1. The EDVC system architecture. The database file is stored in the user's file space.

EDVC communicates directly with the Sentences database via the *Server API*, which is a Java programming interface for Sentences that provides support for data retrieval operations such as *list instances of the entity type person*, and *list the associations defined from this entity*, that are the building blocks for more sophisticated queries. As the Server API allows metadata to be accessed, the use of this architecture means EDVC does not have to be configured to the schema of the database accessed, reducing both the set-up and maintenance costs of the system when compared to those of existing LA products.

Each EDVC database has a corresponding database file, created using the Database Manager Tool, a separate application developed during the implementation of EDVC[6]. A database file stores information regarding the schema of the database, and the name of the database, which must correspond to the name of the Sentences profile representing the database queried. All this information could be stored within the profile, and would be in a production system, but is stored in a separate file to retain a high-level of DBMS independence within EDVC.

7.3 Sentences

Sentences was used as the underlying DBMS as the support provided for the constructs of the data model used in EDVC, described in section 3.2, allowed the system architecture in figure 7.2.1 to be used. However, the data model used by Sentences is not identical to that used by EDVC, but there is simple clearly defined mapping between them that prevents having to convert between two distinct models of data. Each object type is modelled within a Sentences profile as an entity type. This is also true of value types, although these types also have an appropriate associated data type. Each link type is modelled as an association type, and its attributes as association types defined from the association type representing the link type to the appropriate value types.

[6] A user manual for the Database Manger Tool is included as appendix E.

For example, the following Sentences schema, expressed in a convenient textual notation, corresponds to an EDVC database schema that contains two object types, person and report, two value types, text and number, and a link type named *mentions* defined from the object type report to the object type person, and which has an attribute named *context* of type text.

```
value type number, associated data type number
value type text,   associated data type text

entity type person
entity type report

association type mentions     report -> person
association type context      mentions -> text
```

Using Sentences as the underlying DBMS also provided for a simpler system design as the Java programming interface for Sentences could be directly used in EDVC, without requiring an interface between two programming languages.

A further advantage of using Sentences was that the system may be used to integrate the data stored in two or more existing relational databases, possibly with that stored in a chapter in the users file space. An EDVC database is represented within Sentences as a *profile*, which is uniquely named and consists of a set of *chapters*. Each chapter within a profile may store: entities and associations, metadata, information regarding the changes made to the profile, or as shown in figure 7.3.1 may represent the data stored in a relational database.

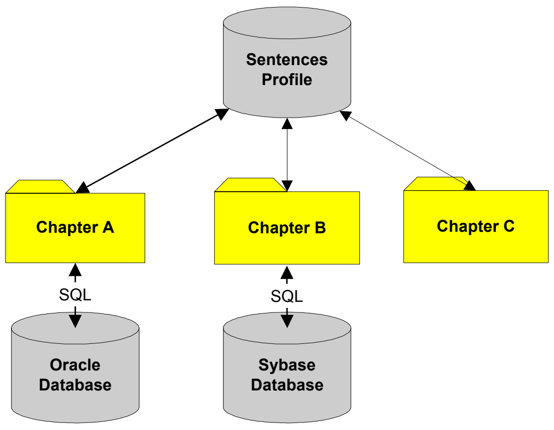


Figure 7.3.1. A Sentences profile that consists of three chapters, two representing existing relational databases, and the third data collected specific to the investigation and held in Sentences.

This is useful in investigative work as the data used in investigations, as described in section 1.2, may be stored in several distinct databases, which can make useful connections difficult to detect and liable to be missed. Using this feature of Sentences these connections may more easily be detected in EDVC. This feature of Sentences also allows the data stored in legacy databases to be migrated to what may be considered a more appropriate technology for LA.

7.4 User Interface Components

The class library provided with the Java runtime system contains several hundred classes for a very diverse range of operations. These classes are used extensively in the implementation of EDVC. The interfaces for querying the database and its schema, reference objects and values that may need to be accessed by several Java objects, which often support distinct areas of system functionality.

The elements displayed in the interfaces for querying the database and its schema and are all represented as some form of directed graph. Each type of graph, and each type of element within each type of graph, is modelled as a distinct class. However, all share a common format. Classes representing nodes model information regarding appearance, such as whether or not the node is highlighted, and the position of the node. The position of a node is stored as the coordinates of the node's minimum enclosing rectangle. This allows classes that manipulate the elements to use a common method of interaction for nodes that may be displayed using a variety of different shapes.

Classes representing edges also model information regarding appearance, such as whether or not a directed arrow should be displayed, and reference the nodes that the edge goes from and to. Additionally, a *link number* is stored for each edge. This number, which is always a non-negative integer, determines if the edge is rendered as a straight line or a curved arc, which allows more than one arc to been drawn between the same pair of nodes. As shown in figure 7.4.1, if the link number of an edge is zero the edge is displayed as a straight line, otherwise, as a curved arc where the direction and degree of curvature depends on the link number.

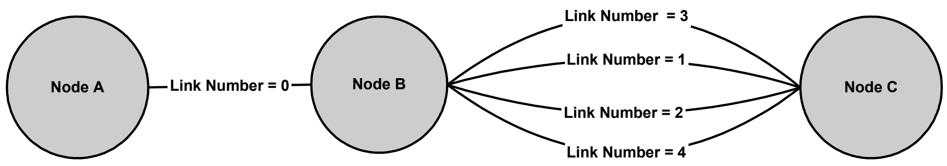


Figure 7.4.1. Illustrating the effect that the link number of an edge has on its curvature.

Generally, the higher the link number, the greater the degree of curvature. Two edges defined between the same nodes whose link number minus one then divided by two (using integer division) is the same, have the same degree of curvature but not the same direction. Edges are drawn as *bspline curves* [BOOR78] whose points are generated using the position of the elements connected and the link number as guides.

To allow the type of elements that may be included in a certain type of graph to be extended without unnecessary modifications to the system, the classes representing graph elements are incorporated into type hierarchies such as that in figure 7.4.2.

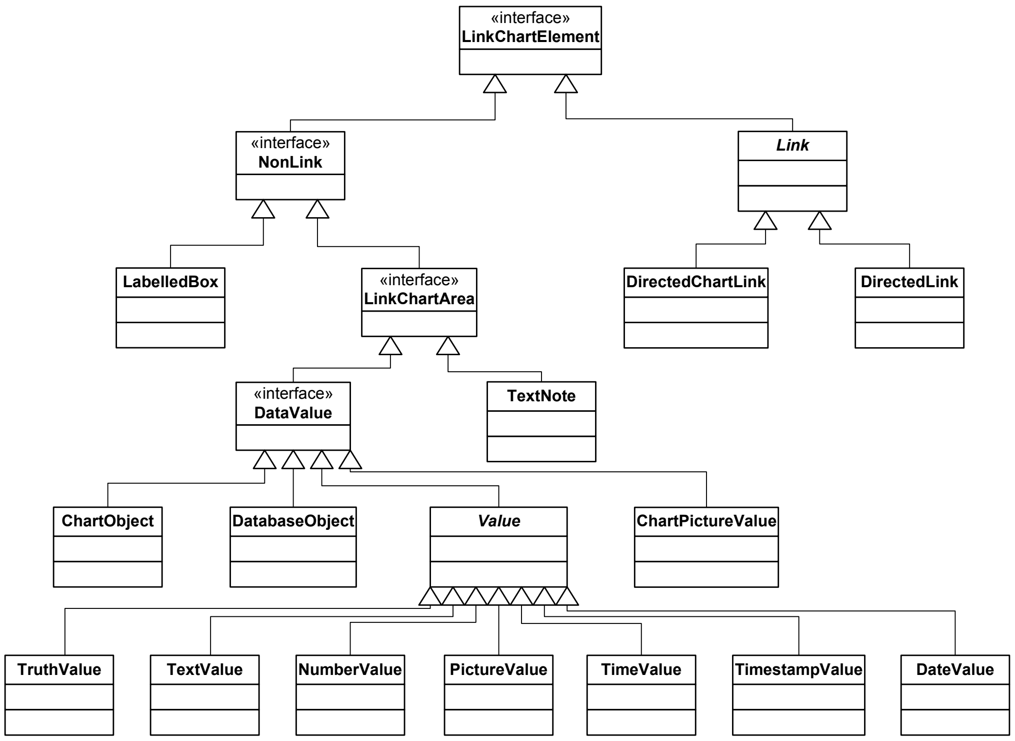


Figure 7.4.2. The class/interface type hierarchy for link chart elements.

Classes modelling graphs also have a similar design. Nodes and edges are referenced using separate lists, and the connections between them may be queried using methods supporting operations such as *list the nodes with this label* and *list the edges involving this node*. The methods within these classes do not maintain edge link numbers; this is the responsibility of the objects that manipulate the graphs. This prevents unnecessary link number modifications, which may have an undesirable impact on display quality. Schema diagrams, link charts and filter pattern queries are stored on disk by serializing[7] the corresponding graph object.

Classes that support chart layout are based upon those implemented for the system described in [ERLI]. The circular layout evenly distributes objects and values at random over the circumference of a circle, the radius of which is proportional to the value of a corresponding instance variable defined for the object representing the associated link chart. Similar to the other layout procedures, elements other than objects, values and links, such as text notes or labelled boxes, are not considered during the layout process. These other elements are positioned above the objects, values and links after the layout process is completed.

[7] Serialization is the Java process for converting a main memory object into a platform independent file that may be accessed to reconstruct the object.

The hierarchical layout is based on the methods described by Eades & Sugiyama [EADE91] and Rowe et al. [ROWE87]. The sub-chart considered during the layout process is modelled as a directed graph. Any cycles in the graph are then removed by strategically reversing the direction of certain edges. Nodes are then assigned levels in the hierarchy by considering the edges going to each node, and its neighbours. The nodes within each level in the hierarchy are then assigned positions to obtain a pleasing layout by evaluating their barycenters [TOMS].

The layout that attempts to minimise the number of links that cross uses the force-directed method described by Fruchterman & Reingold [FRUC91]. The sub-chart to be repositioned is modelled as a set of magnets that repel one another, which represent objects and values, and a set of springs that attract the magnets they connect, which represent links; see figure 7.4.3. A simulated-annealing process iteratively lays out the magnets by minimising the movement in the system caused by the attractive and repulsive forces.

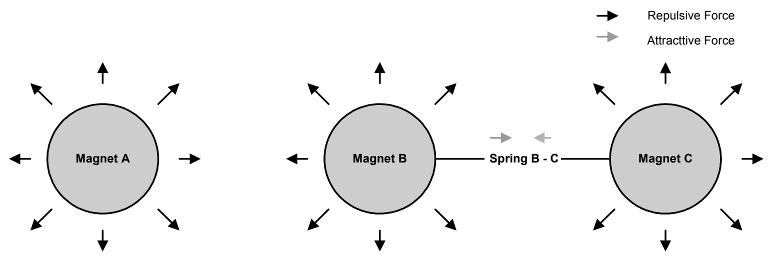


Figure 7.4.3. Illustrating the repulsive and attractive forces generated by the magnets and springs using a force-directed layout method.

Both the separation of the levels in the hierarchical layout and that of the nodes in the layout which attempts to minimise the number of links that cross, is proportional to the value of the corresponding instance variable defined for the object representing the associated link chart. As described in section 4.4.3, these values are settable by the user and allow the system to layout charts appropriately irrespective of the information contained in the labels of objects.

7.5 Retrieving Data from the DBMS for the Explorer Interface

7.5.1 Introduction

Each request to retrieve data from the DBMS initiated within the explorer interface runs in its own thread of execution. Each type of request that may be initiated within the interface, such as display all the objects and paths connected to a particular object, is modelled within the system as a separate class. All such classes are subclasses of a single class, which defines methods for supporting operations such as halting execution and accessing results, which are provided for all such requests. This allows the system to use a common method of interaction.

Instances of these classes are not responsible for displaying results in the explorer interface. Each object representing a request, references an object that is notified when the request finishes processing. It is this object that is responsible for displaying the results obtained, as this allows the results to be displayed in more than one way without having to modify the class representing the request. All such classes communicate with Sentences via the Server API. This API provides meta-query facilities that allow the schema of the database to be analysed, and provides support for data retrieval operations such as *list the instances of the entity type person* and *list the associations that go from this entity*. These retrieval facilities are the building blocks that support the browsing and query facilities of the explorer interface.

7.5.2 Displaying the Information Stored Directly About an Object

The information stored directly about an object in an EDVC database is represented within a Sentences profile by the entities and values linked to the entity corresponding to the object via associations. Retrieving the associations that have a particular entity as either their source or target is an operation that is directly supported by the Server API, as is ascertaining what the source and target of an association are.

As the API directly supports these operations, retrieving the information stored directly about an object only involves converting between the constructs representing the retrieval requests to those constructs used by the Server API, then converting the results obtained via the API into DBMS independent values. This is also true for retrieving the common properties of two objects.

7.5.3 Filtering a Group of Objects

As indicated in section 7.5.1, listing the entity types defined in a Sentences profile, and the instances of a particular entity type, are retrieval operations supported by the Server API. This allows the programmer to iterate through the entities in a profile, or just those instances of particular entity types, and evaluate each in turn. This iterative method is used for finding the objects deemed similar to a specified object according to a given set of filter conditions, and for displaying objects according to their link count.

The similarity conditions specified by the user are transformed into a *condition tree*, such as that in figure 7.5.3.1. Each condition tree is constructed using a leftmost top-down parsing strategy [HUNT99] that provides a single condition tree for a given set of filter conditions.

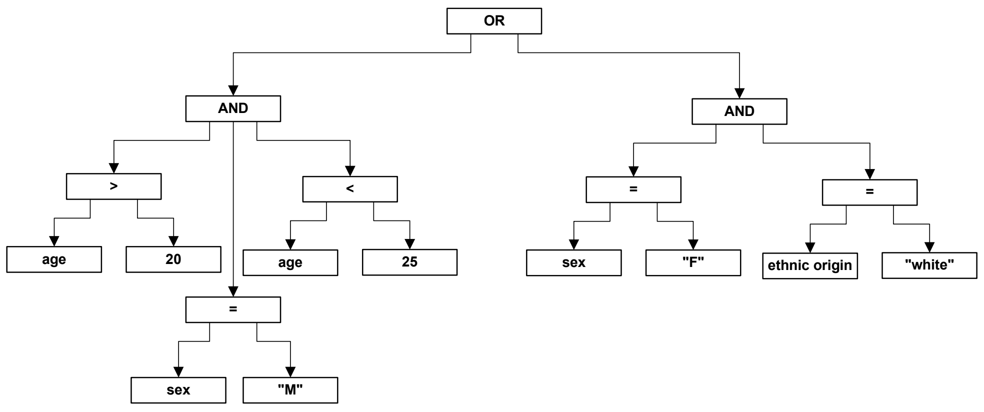


Figure 7.5.3.1. A condition tree for matching males aged between 20 and 25, and white females.

The nodes representing operators, such as Boolean connectives and comparisons, all return either true or false, and only consider the values returned by their children during evaluation. The set of conditions modelled are deemed satisfied if the root node evaluates to true. The condition tree derived from the similarity conditions specified by the user is used to determine which instances of the entity type of the entity representing the object selected by the user satisfy the conditions. The names of those entities are in the result.

When selecting objects according to their link count the instances of entity types representing object types specified as valid by the user are evaluated in turn. For each entity considered, the associations involving the entity that represent links of an appropriate type are counted, and then compared against the condition specified by the user to determine if the name of the entity should be returned. If the user restricted the types of links considered, an appropriate type is any association type that corresponds to a link type selected by the user, otherwise, all associations are considered valid, providing an entity is not associated with a value.

In both these operations all the entities within a profile may be evaluated. The computational cost of retrieving the associations involving a particular entity is O(Na), where Na is the number of associations in the profile. Thus, the computational cost of both these operations is O(Ne.Na), where Ne is the number of entities in the profile. In practice, these operations would generally be more efficient as retrieving the associations that involve a particular entity is an operation that would typically be optimised in a binary-relational DBMS.

7.5.4 Displaying the Paths Connecting Two Specified Objects

The algorithm for finding the paths that connect two specified objects, represented by the *start* and *end* entities, begins by finding all entities linked to the start entity by an association of an appropriate type (see section 7.5.3). A check is then made for each entity listed to ascertain if: the entity is the *end entity*, if there are too many links in the path, or if the entity has been encountered previously in the same path. If any of these conditions is true no action is taken. Otherwise, the entities linked to the entity are listed and the same checks are performed. This process is recursive, and is represented by the algorithm in figure 7.5.4.1.

```
maximum = maximum number of links in a valid path

previous = new list()
Add start entity to the end of previous

queue = new queue()
Add start entity to the end of queue
Add previous to the end of queue

paths = new set()

while( queue is not empty )
{
        current  = remove first element from queue
        previous = remove first element from queue
        length = ((length of previous) – 1) / 2

        if( (current equals end entity) and (length > 0) )
        {
                Add previous to paths
        }

        else if( length < maximum )
        {
                associations = all valid associations involving current

                for( each association a in associations )
                {
                        next = entity linked to current by a

                        if( (previous does not contain a) and
                            ((previous does not contain next) or (next equals end)) )
                        {
                                path = new list()

                                Add to path all elements contained within previous
                                Add a to the end of path
                                Add next to the end of path

                                Add next to the end of queue
                                Add path to the end of queue
                        }
                }
        }
}
```

Figure 7.5.4.1. The algorithm for finding the paths connecting two specified objects that contain less than or equal to a specified number of links; the paths found are returned in **paths**. A similar procedure is used to find the shortest paths(s) connecting two specified objects, except the maximum number of links, initially set to infinity, is set during the algorithm to the length of the first valid path found.

The computational cost of this algorithm is O(Ne.Na). All the entities in a profile could be traversed in a path, which would result in the while loop executing Ne times, and as discussed in section 7.5.3, listing the associations that involve a particular entity is an O(Na) operation.

This assumes that an efficient data structure is used for storing the elements encountered previously in a path, which allows the presence or absence of an element in the list to be determined in constant time. In practice, the algorithm would typically run more efficiently. The length of a valid path would usually be constrained as would the types of links considered, and large networks, particularly social networks [BATA97], are often sparse.

The paths found using the algorithm are added to a chart using the following algorithm. If the start and end entities were distinct and a single path was found, the straight line between the centres of the objects selected by the user is first determined then the number of objects that need to be added to the chart for the path to be displayed are calculated. These figures are then used to determine the gap between the objects that need to be added to the chart so that they are evenly spaced in a straight line between those selected by the user; see figure 7.5.4.2.

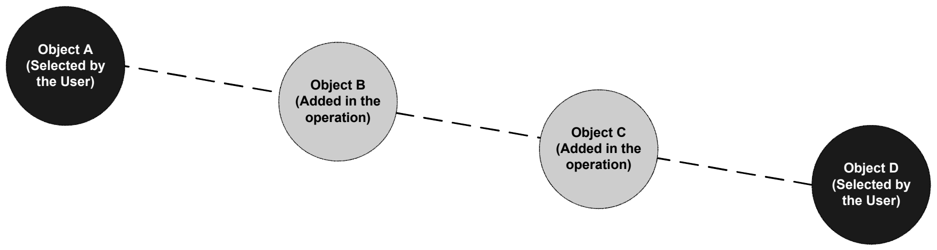

Object A (Selected by the User)
Object B (Added in the operation)
Object C (Added in the operation)
Object D (Selected by the User)

Figure 7.5.4.2. Illustrating how objects added to a link chart are positioned when a single path is found and the objects selected by the user are distinct.

If the start and end entities were distinct and several paths were found then a modified version of this procedure is used. The objects added to the chart for each path are not placed in a straight line between the centres of the objects selected by the user, but in a straight line parallel to this line. As illustrated in figure 7.5.4.3, the location of the line for a particular path is dependent on the number of paths that have already been placed.

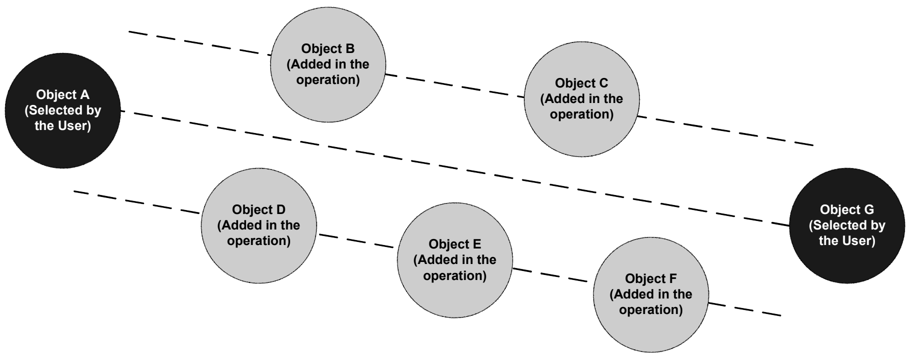

Object A (Selected by the User)
Object B (Added in the operation)
Object C (Added in the operation)
Object D (Added in the operation)
Object E (Added in the operation)
Object F (Added in the operation)
Object G (Selected by the User)

Figure 7.5.4.3. Illustrating how the objects added to a link chart are positioned when multiple paths are found and the objects selected by the user are distinct.

After the objects have been positioned the links between the objects in the paths that are not displayed in the chart are also added. Thus, the objects that were already in the chart before the path was added will retain their original position, and only those objects that were added will be positioned in a line between the centre of the start and end object, or in a line parallel to this line.

If the start and end entities are not distinct the following strategy is used. First, all the objects and links representing the entities and associations included in the paths retrieved that are not already displayed in the chart are added; the objects are positioned randomly. Then, the algorithm described in section 7.4 for laying out charts with a minimal number of link crossings is used to reposition the objects included in the paths retrieved. The algorithm only considers objects and links included in the paths retrieved. Finally, the objects that were displayed in the chart before the paths were added are returned to their original position.

7.5.5 Displaying the Objects Connected to a Specified Object

The algorithm for finding the objects connected to a specified object, represented by the *start entity*, is similar to that for finding paths that connect two specified objects. First, a list containing all entities that are linked to the start entity by an association of an appropriate type (see section 7.5.3) is generated. A check is then made for each entity listed to ascertain if there are too many links in the connecting path or if the entity has been encountered previously in the same path. If any of these is true no action is taken, otherwise, the entities linked to the entity are listed and the same checks made. This process is recursive.

Unlike the algorithm discussed in section 7.5.4, the entities and associations found thus far are not stored in separate lists, but using a single graph data structure. This may reduce main memory usage, as if several paths are found they will tend to have a high degree of similarity. Thus, the algorithm used is that in figure 7.5.5.1.

The computational cost of the algorithm is O(Ne.Na). All of the entities in a profile could be traversed in a path, which would result in the while loop executing Ne times. As discussed in section 7.5.3, listing the associations that involve a particular entity is an O(Na) operation. This assumes that an efficient data structure is used for storing the elements encountered previously, which allows the presence or absence of an element to be determined in constant time. For the same reasons as those discussed in section 7.5.4, in practice this algorithm would typically run more efficiently.

```
graph = new graph()

start = add a node to graph representing the start entity
maximum = maximum number of links in a valid connecting path

previous = new list()
Add start to the end of previous

queue = new queue()
Add start to the end of queue
Add previous to the end of queue

while( queue not empty )
{
        current  = remove first element from queue
        previous = remove first element from queue
        length = (length of previous) – 1

        if( length < maximum )
        {
                associations = all valid associations involving current

                for( each association a in associations )
                {
                        if( graph does not contain an edge representing a )
                        {
                                Add to graph an edge representing a

                                if( previous does not contain next )
                                {
                                        e = entity linked via a

                                        if( graph has a node representing e )
                                        {
                                                next = the node in graph representing e
                                        }

                                        else
                                        {
                                                next = add a node to graph representing e
                                        }

                                        l = new list()

                                        Add to l all elements contained within previous
                                        Add next to the end of l

                                        Add next to the end of queue
                                        Add l to the end of queue
                                }
                        }
                }
        }
}
```

Figure 7.5.5.1. The algorithm for finding the objects connected to a specified object by a path with less than or equal to a specified number of links; the paths are stored in **graph**. A similar procedure is used to find all objects connected to a specified object by a valid path regardless of its length.

The objects and links representing the entities and associations retrieved are added to the link chart as follows. First, all the objects and links not already displayed in the chart are added; the objects are positioned randomly. The algorithm described in section 7.4 for laying out charts with a minimal number of link crossings then repositions the objects that were included in the paths retrieved. The algorithm only considers objects and links included in the paths retrieved. Finally, the objects that were displayed in the chart before the paths were added are returned to their original position.

7.5.6 Displaying the Objects Connected to Several Specified Objects

The objects to be displayed are retrieved using the following algorithm. For each entity in the profile that represents an object specified by the user, a list of the entities connected to the entity by a path that contains less than or equal to the number of links specified by the user, and which only contains links of an appropriate type, is generated. This list is generated using an algorithm similar to that in section 7.5.5, although, no graph data structure is used as there is no need to store the connecting paths. The entities connected at any point during processing are stored as a list.

The lists returned are then analysed. A count representing the number of lists that an entity is included in is calculated for each entity contained in at least one of the lists returned. If the count for an entity is greater than or equal to the parameter specifying the minimum number of specified objects that an object displayed must be connected to, and the entity is an instance of an entity type corresponding to an object type specified as valid by the user, then the entity is returned in the result. Thus, the algorithm is that given in figure 7.5.6.1.

```
counts = new dictionary()
result = new set()

for( each entity e1 corresponding to an object selected by the user )
{
       list = the list of entities connected to e1 by an appropriate path

       for( each entity e2 in list )
       {
              if( counts contains a key entry for e2 )
              {
                     Increment the count for e2
              }

              else
              {
                     Add to counts the key e2 with the associated count 1
              }
       }
}

for( each key value k in counts )
{
       if( (k does not represent one of the objects specified by the user) and
           (k is an entity of an appropriate type to return)              and
           (The count for k is >= the minimum number of specified
            objects that an object displayed must be connected to)             )
       {
              Add k to result
       }
}
```

Figure 7.5.6.1. The algorithm for finding the objects connected to a group of specified objects; those found are returned in **result**.

The cost of the algorithm is $O(Ne^2.Na)$. All the objects in a database could be displayed in a link chart, and the user could specify that all the objects displayed by the operation must be connected to all these objects. This would result in the first for loop performing Ne iterations.

As discussed in section 7.5.5, generating a list of entities connected to a particular entity is an O(Ne.Na) operation. Such a list may be generated for each entity in the Sentences profile, which is the dominant process within the outer for loop.

When an object returned by the operation, and the connecting paths, are to be added to the link chart the following algorithm is used. The object is positioned in the chart by averaging the x and y coordinates of the objects that were selected by the user. Then for each object selected by the user, all the valid paths that connect the object to the object just added to the chart are added using the procedure described in section 7.5.4.

7.5.7 Retrieving Data Via the Programming Interface

As discussed in subsection 7.5.1, every class that implements a request to retrieve data from the DBMS for the explorer interface is a subclass of a common super-class. All the classes implementing retrieval operations provided via the programming interface, which is described in section 4.3.5, are also subclasses of this class. This allows the system to use a common method of interaction with all such classes, and for the classes to inherit functionality that they need to support regardless of the operation they implement, such as adding objects and links to a chart.

The responsibility for ensuring that a retrieval operation provided via the programming interface stops when requested to do so by the system lays with the programmer who implemented the corresponding class, as queries that do not terminate will not in general prevent the user from exploring the database further because they run in a separate thread of execution.

7.6 Evaluating Filter Pattern Queries

7.6.1 Introduction

To efficiently evaluate a filter pattern query (FPQ) containing one or more path constraints would require the development of novel query optimisation techniques. As the focus of the research documented in this thesis is not query optimisation, this significant workload is left as further work, although as discussed in section 8.2.2, several potentially useful ways of improving FPQ evaluation times have been identified. This issue is circumvented in the implementation by the use of two evaluation strategies supported by the same evaluation architecture.

7.6.2 The Filter Pattern Query Evaluation Architecture

A FPQ need not be a connected graph. Prior to evaluation a FPQ is divided into one or more *query sub-graphs* that may be processed in parallel to improve performance. A query sub-graph is defined as a set of constraints where every object constraint is connected via one or more *links* to every other object constraint within the same sub-graph. In this context we define two object constraints as *linked* if they are connected via a link constraint, a path constraint, an object comparison constraint, or that one object constraint has a filter condition that references the information stored directly about the object matched by the other. See figure 7.6.2.1 for examples.

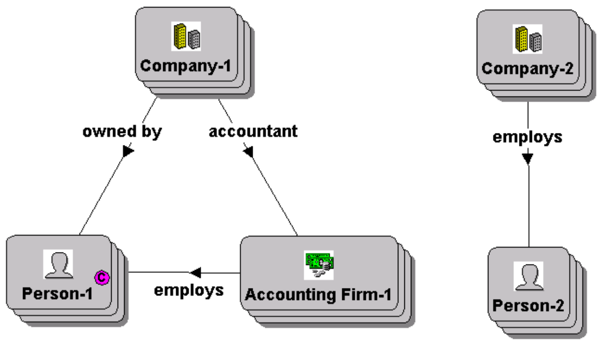


Figure 7.6.2.1. A filter pattern query that may be divided into two query sub-graphs; those constraints located on the left side of the pattern form one sub-graph, those located on the right form the other.

Query sub-graphs are processed using separate threads of execution. The identifiers of the objects and links returned after processing each sub-graph are combined then used to create a new link chart.

7.6.3 Overview of the Evaluation Strategies

There are two strategies for evaluating query sub-graphs, one for *simple query sub-graphs*, and another for all others, which is a direct implementation of the algorithm give in figure 5.4.3.1.1. A simple query sub-graph does not contain any path constraints, and satisfies the following condition: if all object comparison constraints specifying that the objects matched by two constraints should differ were removed, and all *links* representing an object constraint has a filter condition that references the information stored directly about an object matched by another object constraint were not considered, then the sub-graph would still be connected.

For example, the query sub-graph given in figure 7.6.3.1 is regarded as a simple query sub-graph, since if the object comparison constraint connecting the object constraints *Person-1* and *Person-2* was removed, the sub-graph would still be connected. However, if the link constraint that connects the object constraints *Company-1* and *Company-2* was removed, the sub-graph would no longer be deemed a simple query sub-graph as all the object constraints would no longer be connected to one another by one or more *links*.

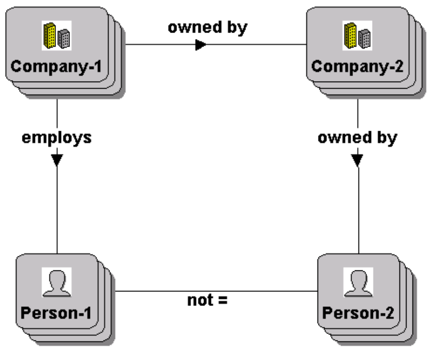


Figure 7.6.3.1. An example of a simple query sub-graph.

7.6.4 Evaluating Simple Query Sub-Graphs

7.6.4.1 Constructing the Evaluation Tree

Simple query sub-graphs are evaluated by matching the constraints in the sub-graph against connected sub-networks of the network formed by the entities and associations in a Sentences profile. This is a two-stage process. First, the query sub-graph is converted into an evaluation tree, where each tree node is associated with a constraint in the query sub-graph.

The root node is always associated with an object constraint, which is chosen using the following strategy: if there are one or more object constraints that match a specific object in the database, then choose one of these randomly; otherwise, if there are one or more object constraints that only match instances of a specified object type and have filter conditions, then choose one of these randomly; otherwise, if there are one or more object constraints that only match instances of a specified object type, then choose one of these randomly; otherwise, randomly choose an object constraint.

All link constraints and object comparison constraints specifying that the objects matched by the associated object constraints must be identical, that involve the object constraint chosen, are then processed. For each such constraint there are two nodes added to the evaluation tree, one that is associated with the constraint, and one that is associated with the object constraint linked via the constraint. The node associated with the object constraint is made a child of the node associated with the constraint currently being processed, which is then made a child of the root node. This process is carried out recursively on all objects constraints linked to the root that have not been processed previously in the algorithm. Thus, the algorithm used to construct the evaluation tree is given in figure 7.6.4.1.1. *Appropriate linkage constraint* is used in the algorithm to refer to a link constraint, or an object comparison constraint specifying that the objects matched by the associated object constraints must be identical.

```
constraint = select the object constraint that the root node will correspond to
root = generate tree node for constraint

processed = new dictionary()
comparisons = new set()
queue = new queue()

Add to processed the key constraint and the associated value root
Add constraint to the end of queue

while( queue not empty )
{
        current = remove first element from queue
        cnode = the value in processed corresponding to the key value current

        for( each appropriate linkage constraint lc involving current )
        {
                next = object constraint linked to current by lc

                if( processed does not contain a key value next )
                {
                        Add next to the end of queue
                }

                if( processed does not contain a key value lc )
                {
                        node1 = generate tree node for next
                        node2 = generate tree node for lc

                        Make node1 a child of node2
                        Make node2 a child of cnode

                        Add to processed the key value lc and the associated value node2

                        if( processed contains the key value next )
                        {
                                node3 = the value in processed corresponding to the key
                                        value next

                                Add to comparisons a comparison construct representing
                                an equals comparison between node1 and node3
                        }

                        else
                        {
                                Add to processed the key value next and the associated
                                value node1
                        }
                }
        }
}

for( each object comparison constraint occ in the query sub-graph )
{
        Let node1 and node2 equal the tree nodes corresponding to the object
        constraints connected by occ

        if( occ indicates that the objects matched by the constraints should differ )
        {
                Add to comparisons a comparison construct representing a not equals
                comparison between node1 and node2
        }

        else
        {
                Add to comparisons a comparison construct representing an equals
                comparison between node1 and node2
        }
}
```

Figure 7.6.4.1.1. The algorithm for constructing an evaluation tree, and the associated comparison constructs, for a simple query sub-graph.

The object constraints that are reachable via more than one path without loops from the object constraint that the root node is associated with will have more than one node in the evaluation tree associated with them; one for each path that connects the associated object constraint with the object constraint that the root node corresponds to. For example, the query sub-graph formed by the constraints located to the left of the FPQ in figure 7.6.2.1 would be represented as the evaluation tree in figure 7.6.4.1.2.

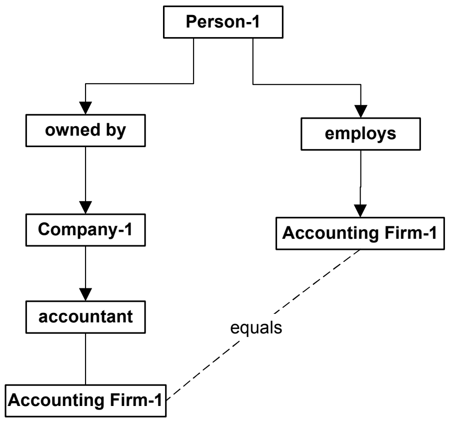


Figure 7.6.4.1.2. The comparison constructs are represented as dotted labelled lines connecting the associated tree nodes, and not as separate nodes.

The root of this evaluation tree is associated with the object constraint *Person-1* as there are no object constraints in the query sub-graph that only match a specified object, and this object constraint is the only one that only matches instances of a specified object type and has a set of filter conditions.

7.6.4.2 Overview of the Evaluation Tree

Each node in an evaluation tree has an associated identifier, which may change during the course of processing. This identifier is ether the identifier of an entity or association in the Sentences profile, or null. The nodes in an evaluation tree have different roles, which depend on the type of constraint in the query sub-graph that they are associated with.

The root node iterates over the entities in the Sentences profile of the appropriate type that satisfy any associated filter conditions, which like all conditions associated with tree nodes depend on the constraint in the query sub-graph that the node is associated with. As the node iterates the identifier associated with the node is set to that of the entity currently considered, if there are no valid entities the identifier is set to null.

A tree node associated with a link constraint iterates over the associations of the appropriate type have the entity associated with its parent as a source or target value, and which satisfy any associated filter conditions. As the node iterates its identifier is set to that of the association currently considered.

Each such node also has an associated *child identifier*, which is the identifier of the entity that is linked via the association currently considered to the entity associated with the parent node. If no association is currently considered, as is the case when the identifier of the parent node is null or when there are no valid associations for the entity associated with the parent node, the identifier and child identifier are null.

Tree nodes that correspond to object constraints, with the exception of the root node, check that the child identifier of their parent node is that of an entity of an appropriate type that satisfies any associated filter conditions. If so, the node has the same identifier as the child identifier of its parent, otherwise, null.

Nodes that correspond to object comparison constraints that specify that the objects matched by the associated object constraints should be identical, like those that correspond to link constraints, also have a child identifier. The child identifier of such a node, as is the identifier associated with the node, is the same as the identifier of its parent.

7.6.4.3 Processing the Evaluation Tree

First, the root node is set to its first identifier, providing it has one; then the children of the root node are set to their first identifier, providing they have one. This is repeated recursively for all tree nodes, where appropriate to do so. If after this process is completed all tree nodes have a non-null identifier then the comparison constructs are processed. If the comparisons represented by these constructs are satisfied, the identifiers of the nodes corresponding to object and link constraints are added to the list of those to return.

The tree then iterates to a new set of identifiers, providing there is one. First, a node is chosen to iterate to its next identifier. This node must either be the root node or one that corresponds to a link constraint, and is selected using the recursive procedure given in figure 7.6.4.3.1.

As the root node is first passed to the procedure, this corresponds to an ordered search of the evaluation tree. Once an appropriate node has been found, providing there is one, it is iterated to its next identifier. All nodes that were tested before the node was found are reset to their first identifier, where appropriate to do so. The process of checking if all tree nodes have non-null identifiers and if the comparison constructs are satisfied, storing the list of associated identifiers if so, then iterating the tree to a new set of identifiers, continues until there are no further sets of identifiers to consider.

```
FindTreeNode(TreeNode current)
{
        for( each child c of current processed from leftmost child to rightmost child )
        {
                returned = FindTreeNode(c)

                if( returned does not equal null )
                {
                        return returned
                }
        }

        if( current has another identifier to iterate to )
        {
                return current
        }

        else
        {
                return null
        }
}
```

Figure 7.6.4.3.1. The procedure used to select the next node to iterate.

7.6.4.4 The Efficiency of the Evaluation Strategy

The factors that primarily determine the efficiency of the evaluation strategy are the policy used to select the object constraint that the root node is associated with, and the order that link constraints are processed when the evaluation tree is constructed. The sophistication of our approach is restricted by the level of metadata available via the Sentences Server API, which does not allow the number of entities or associations stored of a given type to be determined without iterating through them, which is not practical for non-trivial databases.

If the Sentences API permitted the number of entities or associations of a given type to be determined directly the manner in which the evaluation tree is constructed (described in section 7.6.4.1) could be improved upon. The approximate IO costs for all, or a subset of, the possible evaluation trees for a query sub-graph could be calculated and the tree that would seem to cost least to be evaluated could be selected.

7.7 Discussion

The successful user evaluation of EDVC, and the use of the system both on a UNIX and on a Microsoft Windows based platform, validated our design choice of using Java for reasons of portability, and for the support provided for browsing/query operations and data visualisation.

The use of the system architecture in figure 7.2.1 also proved successful, and meant that the setup and maintenance cost of current LA products were not present in EDVC.

We have recently started to develop a version of EDVC that uses a file based data store. Our efforts so far indicate that separating the user interface and the data retrieval components in EDVC, and only passing DBMS independent data structures between the two, was wise. This has made modifying the system to use a different data source significantly easier. However, if we were to re-implement EDVC, we would use a 3-layer architecture whereby the data retrieval components would be separated from the EDVC user interface, and used as sever. This would allow EDVC to be used as thin-client, and effectively utilised in a world-wide-web browser environment.

The technique currently used to process non-simple query sub-graphs (discussed in section 7.6.3) resulted in lengthy evaluation times for large data sets, and as discussed in section 8.2.2 is the subject of further work. A further modification we would perform, given the provision of a more sophisticated API in Sentences that allowed the number of instances of a particular type to be determined, would be the development of a more sophisticated technique for constructing an evaluation tree for a simple query sub-graph, discussed in sections 7.6.4.1 and 7.6.4.4. By estimating the number of elements that may be iterated over using a selection of possible evaluation trees for a given query sub-graph then selecting the one with the lowest estimate the time required to evaluate many FPQs could be greatly reduced.

Chapter 8 – Conclusions & Further Work

8.1 Principle Achievements

The main contribution of the work documented in this thesis is the design, implementation and evaluation of a visual database interface supporting LA that satisfies all the design criteria specified in section 1.4. The system implemented, EDVC, improves upon the level of support for LA that may be obtained from previous database interfaces by providing:

- A data model, discussed in chapter 3 and developed from those used in mainstream LA products, which has constructs for modelling distinct types of objects and links in the problem domain and object and link attribute information. A facility for constructing and displaying data models graphically is provided that allows the user to specify default values for the visual properties of different types of objects and links when displayed in a link chart.

- An interface, which is described in chapter 4, that allows link charts to be constructed by integrating and editing the results of a number of browsing and query operations in a single link chart. The facilities provided in the interface allow visual elements not representing objects and links in the database to be added to a chart, which is described in section 4.4.1, and for charts to be stored for subsequent display and further development, which is described in section 4.6.

- A set of browsing operations that may be extended without further system development which directly support the fundamental data analysis operations used in LA, which are described in section 1.1. These operations are described in section 4.2; further browsing operations we consider of general value are described in section 4.3.

- A query language, which is described in chapter 5, that allows queries to be expressed that filter objects and links in the database, or in a link chart, according to their attribute information and connections to other objects. Support is provided for constraining the types of connections displayed to just those that represent particular types of relationship in the problem domain, and for specifying that objects must be connected, irrespective of how they are connected.

- An implemented system that our user evaluation, reported in chapter 6, indicates has a style of interaction suitable for supporting the intended user community. The results of the evaluation indicate that users find the browsing style of interaction more intuitive than the query language style of interaction. This would seem to be because, when browsing users are dealing at a factual level whereas when using the query language they must deal with a higher conceptual level, which is less familiar and requires greater knowledge of the database schema.

- A system architecture, which is outlined in section 7.2, that doesn't require the user to configure the system to the schema of the database queried or to reconfigure the system when the schema is modified.

8.2 Future Work

8.2.1 Introduction

For EDVC to be effectively used in a commercial or governmental environment a number of additional facilities would need to be incorporated into the system, such as facilities for data entry and manipulation, and for creating and modifying the database schema. While these are mainly development issues, there are a number of interesting possible areas of future research available.

8.2.2 Query Optimisation

The strategy currently employed for evaluating filter patterns, described in section 7.6, is not suitable for querying the paths through large and well-connected datasets. This is partially a consequence of the restricted level of metadata available via the programming interface for the underlying DBMS Sentences. The use of a more refined connection searching strategy would greatly improve evaluation times.

Several forms of query optimization have been developed for querying semi-structured data [FERN98, MCHU99] such as XML, which analyze some abstract model of the structure of the data queried prior to inspecting the data in order to compute how valid connections may be derived. This in certain circumstances is appropriate in EDVC. Many of these techniques however, rely upon data being hierarchically structured in a rooted tree, and assume that the elements in the connecting paths are not to be displayed. Further research into broadening the scope of some of these approaches for more general forms of graph could prove successful for improving query performance within EDVC.

8.2.3 Support for Hypothesis Testing

From the perspective of a user each Sentences database is represented as a *profile* consisting of a set of *chapters*. Each chapter may store metadata, or one or more entities or associations. Information may also be stored concerning changes made to the profile [WILL02]. As the chapters that a profile consists of can easily be changed this has great potential for use for hypothesis testing with criminal networks, where the boundaries to the legitimate world are often hard to define [SPAR91].

If each investigator is provided with a separate profile containing one or more chapters that store information available to all investigators and an additional chapter that stories modifications made by that investigator, the impact of these modifications can be analysed without effecting the view that other investigators have of the common data (see figure 8.2.3.1). Further research into how best to integrate these facilities with those for querying the data stored at specific points in time would provide increased support for hypothesis testing.

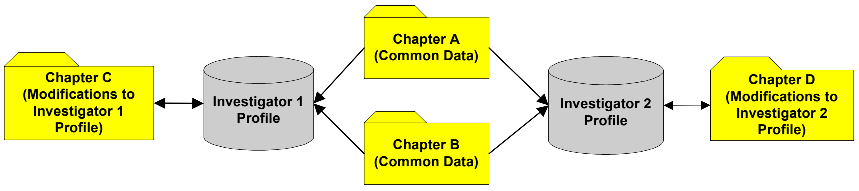

Chapter A (Common Data)
Chapter C (Modifications to Investigator 1 Profile)
Investigator 1 Profile
Investigator 2 Profile
Chapter D (Modifications to Investigator 2 Profile)
Chapter B (Common Data)

Figure 8.2.3.1. Each profile represents the common data stored in chapters A and B, in addition to that stored in an exclusive additional chapter that stores the modifications made to the profile.

8.2.4 Further Evaluation of EDVC

As stated in section 6.5, conducting further user evaluations of EDVC would prove of value. A paper and pencil evaluation of the filter query language would allow more accurate results to be obtained as they would not be influenced by the style of user interface in EDVC for constructing filter pattern queries. An additional evaluation using the data from previously solved crimes, and a number of investigators, would allow EDVC be compared against both manual techniques and other systems for supporting LA, such as Watson (described in section 2.2.1). However, the comments in section 6.2.3 regarding how the results of a user evaluation with law enforcement community officers may be influenced by the subjects' previous experience of LA software or the opinions of their colleagues should be considered.

8.2.5 Providing Personalised User Views

The relationships in the problem domain of relevance to a particular user may correspond to two or more of the links in a database. Consider the data in figure 1.1.1. An investigator may prefer to only see personal relationships between people, the locations they frequent, and if they are associated in some way. If the investigator defines two people as associated if they work at a common location then the chart would now look like that in figure 8.2.5.1.

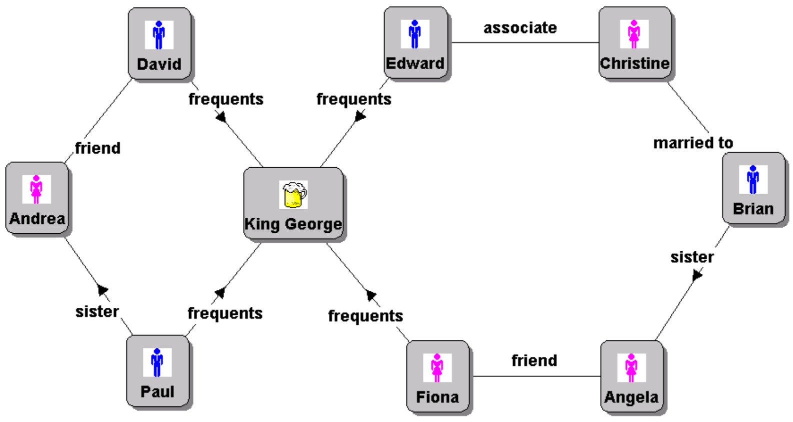


Figure 8.2.5.1. The data displayed in figure 1.1.1, where the problem domain relationships representing that people work at a common location are replaced by links signifying they are associated.

Facilities for defining new link types representing the concatenation of two or more of those defined in the database schema would reduce the number of objects and links displayed in a link chart, thus, increasing comprehensibility while retaining the required information.

A visual specification of the links to be derived from those stored may be used to define a new link type. This could subsequently be utilised by the EDVC to access the data stored. How best to modify the EDVC class library to allow for data to be materialised from that stored in the DBMS is a matter for future work.

8.2.6 Enhanced Data Visualisation Facilities

Due to the modular system architecture the results returned by query components may be used in alternative query result visualisation interfaces. Support for emphasizing the order in which a series of events occurred, as in figure 1.2.1, has proved of value in many problem domains where LA is used. Additionally, facilities for visualising large networks, or an overview of their structure, may prove productive for identifying sub-networks of interest that can be explored in greater detail using the explorer interface. There exist many techniques for displaying such networks [BECK95, LAMP96, PARK98]; see figure 8.2.6.1 for an example.

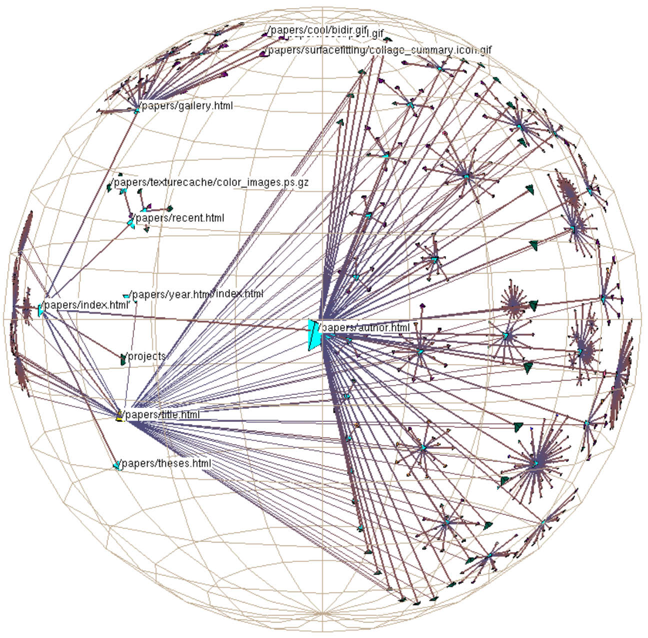
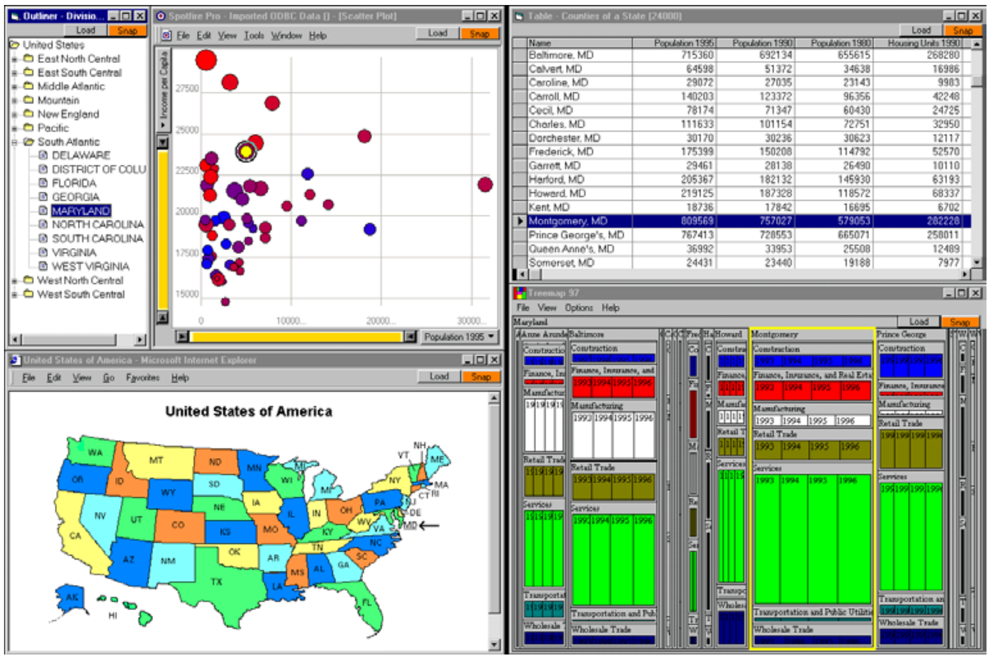


Figure 8.2.6.1. A 3-D hyperbolic representation of a network derived from the structure of a website.

Further research into how these techniques could be incorporated into EDVC so that the user can obtain a coherent view of both the sub-network being explored and the context of the network within the database would be of great value for exploring large networks. Candidate strategies include coordinating the views from several interfaces, such as in *snap-together visualizations* [NORT00], an example of which is given in figure 8.2.6.2, and the use of *focus and context* visualisation paradigms [LENG99].



Name	Population 1995	Population 1990	Population 1980	Housing Units 1990
Baltimore, MD	715360	692134	655615	268280
Calvert, MD	64598	51372	34638	16986
Caroline, MD	29072	27035	23143	9983
Carroll, MD	140203	123372	96356	42248
Cecil, MD	78174	71347	60430	24725
Charles, MD	111633	101154	72751	32950
Dorchester, MD	30170	30236	30623	12117
Frederick, MD	175399	150208	114792	52570
Garrett, MD	29461	28138	26490	10110
Harford, MD	205367	182132	145930	63193
Howard, MD	219125	187328	118572	68337
Kent, MD	18736	17842	16695	6702
Montgomery, MD	809569	757027	579053	282228
Prince George's, MD	767413	728553	665071	258011
Queen Anne's, MD	36992	33953	25508	12489
Somerset, MD	24431	23440	19188	7977

Figure 8.2.6.2. A *Snap-Together Visualization* of census data from the United States of America.

8.2.7 Use of a More General Data Model

The data model supported in EDVC has proved well suited to modelling the datasets typically analysed in LA, such as those relating to criminal activities or financial transactions. However, datasets with a more complicated structure may require a more expressive data model, such as that supported in the DBMS Sentences (described in section 2.2.4).

Supporting this data model in EDVC would require several enhancements to the system. In addition to development issues, such as altering the data visualisation facilities to take into consideration that links may go from, or to, other links, an alternative query processing strategy would be required as connecting paths may not involve intermediate objects.

The automatic chart layout algorithms currently used would also seem unsuitable. A heuristic method for minimising the number of links that cross is not suitable in charts where a link may be defined from or to another link, or for emphasizing the structure of connecting paths returned by query operations that contain such links. Thus, further research would be required to retain the usefulness of these facilities.

8.2.8 Query Language Enhancements

For EDVC to be used as a general-purpose database query interface the expressive power of the filter pattern query language would need to be enhanced to support set operators, which have proved of great value in data processing applications. One method of supporting these operators within the system is to increase the range of constraints that may be used in a filter pattern to include a subset of those provided in the visual database query language Gql [PAPA94], examples of which are given in figure 8.2.8.1.

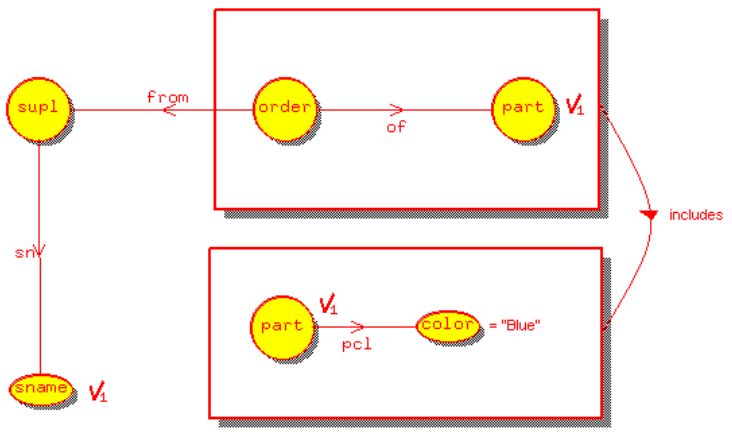


Figure 8.2.8.1. A Gql query that retrieves the names of suppliers that supply all blue parts. The boxes represent collections, and the ticked elements indicate those elements to be included in the result.

The set of queries that may be expressed in Gql, which uses a binary relational data model, is a superset of that which may be expressed using the relational algebra by providing additional support for operators for computation and the grouping and ordering of results. Supporting analogous constraints in the filter pattern query language would enhance the expressive power of the language to include a useful class of queries. However, further research would be required to ascertain how best to incorporate these constructs within the language, and how to efficiently implement them in the underlying query components when they are used in conjunction with those for matching paths through the database.

References

[ABRE95] R. M. Abreu. A Visual Query Interface to the Associational Functional Database Language Hydra, MSc. Thesis, Birkbeck College, University of London, 1995.

[AHO74] A. V. Aho, J. E. Hopcorft, J. D. Ullman. The Design and Analysis of Computer Algorithms, Addison-Wesley, ISBN 0201000296, 1974.

[AHO79] A. V. Aho, J. D. Ullman. Universality of Data Retrieval Languages, Proceedings of the 6th ACM Symposium on Principles of Programming Languages, 1979.

[ANAC] Anacapa Sciences Inc. http://www.anacapasciences.com,
correct as of the 1st of November 2003.

[ANAL] Analytic Technologies Limited. UCINET, http://www.analytictech.com,
correct as of the 1st of November 2003.

[ANGE90] M. Angelaccio, T. Catarci, G. Santucci. QBD*: A Graphical Query Language with Recursion, IEEE Transactions on Software Engineering Vol. 16 No. 10, 1990.

[AYRE95] R. Ayres. Enhancing the Semantic Power of Functional Database Languages, Ph.D. Thesis, Birkeck College, University of London, 1995.

[AYRE96] R. Ayres, P. J. H. King. Querying Graph Databases Using a Functional Language Extended with Second Order Facilities, Proceedings of the 16th British National Conference on Databases, 1996.

[BATA97] V. Batagelj, A. Mrvar. Pajek – Program for Large Network Analysis, University of Ljubljaba, 1999.

[BECK95] R. A. Becker, S. G. Eick, A. R. Wilks. Visualizing Network Data, IEEE Transactions on Visualization and Computer Graphics Vol. 1 No. 1, 1995.

[BLAC99] A. Blackwell, T. R. G. Green. Does Metaphor Increase Visual Language Usability?, Proceedings of the IEEE Symposium on Visual Languages VL'99, 1999.

[BLAC01] A. F. Blackwell. See What You Need: Helping End-users to Build Abstractions, Journal of Visual Languages & Computing Vol. 12 No. 5, 2001.

[BOOR78] C. de Boor. A Practical Guide to Splines, Springer Verlag, ISBN 0387953663, 1978.

[CARE96] M. Carey, L. Haas, V. Maganty, J. Williams. PESTO: An Integrated Query/Browser for Object Databases, Proceedings of the 22nd Conference on Very Large Databases, 1996.

[CATA95] T. Catarci, G, Santucci. Diagrammatic Vs Textual Query Languages: A Comparative Experiment, Proceedings of the IFIP W.G. 2.6 Working Conference on Visual Databases, 1995.

[CATA97] T. Catarci, M. F. Constabile, S. Levialdi, C. Batini. Visual Query Systems for Databases: A Survey, Journal of Visual Languages and Computing Vol. 8 No. 2, 1997.

[CATA00] T. Catarci. What Happened When Database Researchers Met Usability, Information Systems Vol. 25 No. 3, 2000.

[CATT00] R. G. Cattell, D. K. Barry, M. Berler, J. Eastman, D. Jordan, C. Russell, O. Schadow, T. Stanienda, F. Velez. The Object Database Standard: ODMG 3.0, Morgan Kaufman Publishers, ISBN 1558606475, 2000.

[CERI99] S. Ceri, S. Comai, E. Damiani, P. Fraternali, S. Paraboschi, L. Tanca. XML-GL: a Graphical Language for Querying and Restructuring XML Documents, Proceedings of the 8th International World Wide Web Conference, 1999.

[CHAN88] M. K. Chandra. Theory of Database Queries, Proceedings of the 7th ACM Symposium on Principles of Database Queries, 1988

[CHAV97] M. Chavda, P. T. Wood. Towards an ODMG-Compliant Visual Object Query Language, Proceedings of the 23rd International Conference on Very Large Databases, 1997.

[COLE96] M. K. Coleman. Aesthetics-Based Graph Layout for Human Consumption, Software Practice & Experience Vol. 26 No. 1, 1996.

[CONS90] M. P. Consens, A. O. Mendelzon. GraphLog: a Visual Formalism for Real Life Recursion, Proceedings of the 9th ACM SIGACT-SIGMOD Symposium on Principles of Database Systems, 1990.

[CONS94] M. P. Consens. Creating and Filtering Structural Data Visualization Patterns, Ph.D. Thesis, University of Toronto, 1994.

[CRIM90] C. Crimi, A. Guercio, G. Nota, G. Pacini, G, Tortora, M. Tucci. Relation Grammars for Modelling Multi-dimensional Structures, Proceedings of the 1990 IEEE Workshop on Visual Languages, 1990.

[CYRA] Cyram Limited. NetMiner, http://www.netminer.com, correct as of the 1st of November 2003.

[EADE91] P. Eades, K. Sugiyama. How to Draw a Directed Graph, Journal of Information Processing Vol. 13 No. 4, 1991.

[ERLI] U. Erlingsson, M. Krishnamoorthy. Intercative Graph Drawing on the World Wide Web, available from http://www.cs.rpi.edu/projects/pb/graphdraw, correct as of the 1st of November 2003.

[ERWI03] M. Erwig. Xing: A Visual XML Query Language, Journal of Visual Languages and Computing Vol. 14 No. 1, 2003.

[FEKE99] J. D. Fekete, C. Plaisant. Excentric Labeling: Dynamic Neighborhood Labeling for Data Visualization, Proceedings of CHI'99, 1999.

[FERN98] M. Fernandez, D. Suciu. Optimizing Regular Path Expressions Using Graph Schemas, Proceedings of the 14th International Conference on Data Engineering, 1998.

[FITT54] P. M. Fitts. The Information Capacity of the Human Motor System in Controlling the Amplitude of Movement, Journal of Experimental Psychology Vol. 47, 1954.

[FOGG84] D. Fogg. Lessons from a "Living in a Database" Graphical Query Interface, Proceedings of the SIGMOD '84 Conference, 1984.

[FORE] Forensic Accounting. http://www.forensicaccounting.com, correct as of the 1st of November 2003.

[FRAN95] K. A. Frank. Identifying Cohesive Subgroups, Social Networks Vol. 17 No.1, 1995.

[FREE00] L. C. Freeman. Visualizing Social Networks, Journal of Social Structure Vol. 1 No. 1, 2000.

[FRUC91] T. Fruchterman, E. Reingold. Graph Drawing by Force-Directed Placement, Software Practice & Experience Vol. 21 No. 11, 1991.

[GOLI89] E. J. Golin, S. P. Reiss. The Specification of Visual Language Syntax, Proceedings of the 1989 IEEE Workshop on Visual Languages, 1989.

[GREE90] S. Greene, S. Devlin, P. Cannata, L. Gomez. No Ifs, ANDs or ORs: a Study of Database Querying, International Journal of Man-Machine Studies Vol. 32. No. 3, 1990.

[HANN99] Robert A. Hanneman. Introduction to Social Network Methods, http://wizard.ucr.edu/~rhannema/networks/nettext.pdf, correct as of the 1st of November 2003.

[HEIL85] S. Heiler, A. Rosenthal. G-WHIZZ, a Visual Interface for the Functional Model with Recursion, Proceedings of the 11th International Conference on Very Large Databases, 1985.

[HERM00] I. Herman, G. Melançon, M. S. Marshall. Graph Visualization and Navigation in Information Visualization: a Survey, IEEE Transactions on Visualization and Computer Graphics Vol. 6 No. 1, 2000.

[HU03] P. J. Hu, C. Lin, H. Chen. Examining Technology Acceptance by Law Enforcement Officers: An Exploratory Study, Proceedings of the 1st NSF/NJI Symposium on Intelligence and Security Informatics, 2003.

[HUNT99] R. Hunter. The Essence of Compilers, Prentice Hall, ISBN 0137278357, 1999.

[ICL94] ICL Limited. Intelligence Analyst Workbench Training Manual, 1994.

[I2] I2 Limited. http://www.i2.co.uk, correct as of the 1st of November 2003.

[KENT79] W. Kent. Limitations of Record-Based Information Models, ACM Transactions on Database Systems Vol. 4 No. 1, 1979.

[LAMP96] J. Lamping, R. Rao. The Hyperbolic Browser: A Focus + Context Technique for Visualizing Large Hierarchies, Journal of Visual Languages and Computing Vol. 7 No. 1, 1996.

[LAZY] LazySoft Limited. htttp://www.lazysoft.com, correct as of the 1st of November 2003.

[LENG99] Y. K. Leung, M. D. Apperley. A Review and Taxonomy of Distortion-Oriented Presentation Techniques, Readings in Information Visualization: Using Vision to Think, Morgan Kaufmann, ISBN 1558605339, 1999.

[MCGR80] D. R. McGregor, R. G. Malone. The FACT Database System, Proceedings of Symposium for Database Systems, in Systems for Large Databases, edited by P. C. Lockemann & E. J. Newhold, Amsterdam, 1980.

[MCHU99] J. McHugh, J. Widom. Query Optimization for XML, Proceedings of the 25th International Conference on Very Large Data Bases, 1999.

[MUNR00] K. D. Munroe, Y. Papakonstantinou. BBQ: A Visual Interface for Integrated Browsing and Querying of XML, 5th IFIP 2.6 Working Conference on Visual Database Systems, 2000.

[MURR00] N. S. Murray, N. W. Paton, C. A. Goble. J. Bryce. Kaleidoquery – A Flow-based Visual Language and its Evaluation, Journal of Visual Query Languages and Computing Vol. 11 No. 2, 2000.

[NORT00] C. North, B. Shneiderman. Snap-Together Visualization: A User Interface for Coordinating Visualizations via Relational Schemata, Proceedings of Advanced Visual Interfaces 2000, 2000.

[OREF92] S. Orefice, G. Palese, M. Tucci, G. Tortora, G. Costagliola, S. K. Chang. A 2D Interactive Parser for Iconic Languages, Proceedings of the 1992 IEEE Workshop on Visual Languages, 1992.

[PAPA94] A Papatonakis, P. J. H. King. Gql, A Declarative Graphical Query Language Based on the Functional Data Model, Proceedings of the Conference on Advanced Visual Interfaces 94, 1994.

[PAPA95] A. Papatonakis. Gql, a Declarative Graphical Query Language Based on the Functional Data Model, Ph.D. Thesis, Birkbeck College, University of London, 1995.

[PARK98] G. Parker, G. Franck, C. Ware. Visualization of Large Nested Graphs in 3D, Journal of Visual Languages and Computing Vol. 9 No. 3, 1998.

[PETE98] M. B. Peterson. Applications in Criminal Analysis: A Sourcebook, Greenwood Press, ISBN 027596468X, 1998.

[POUL01] A. Poulovassilis, S. G. Hild. Hyperlog: A Graph-Based System for Database Browsing, Querying, and Update, IEEE Transaction on Data and Knowledge Engineering Vol. 13 No. 2, 2001.

[RAYW83] V. J. Rayward-Smith. A First Course in Formal Language Theory, Blackwell Scientific Publication, ISBN 0632011769, 1983.

[REIS75] P. Reisner, R. F. Boyce, D. D. Chamberlain. Human Factors Evaluation of Two Database Query Languages – Square and Sequel, Proceedings of the AFIPS National Computer Conference, 1975.

[REIS81] P. Reisner. Human Factors Studies of Database Query Languages: A Survey and Assessment, Computing Surveys Vol. 13 No. 1, 1981.

[RODG97] P. J. Rodgers, P. J. H. King. A Graph Rewriting Visual Language for Database Programming, The Journal of Visual Languages and Computing Vol. 8 No. 6, 1997.

[ROWE87] L. A. Rowe, M. Davis, E. Messinger, C. Meyer, C. Spirakis, A. Tuan. A Browser for Directed Graphs, Software Practice & Experience Vol. 17 No. 1, 1987.

[SANT97] G. Santucci, L. Tarantino. A Hypertabular Visualizer of Query Results, IEEE Symposium on Visual Languages, 1997.

[SOUT92] S. B. Southerdon. Information Technology: Support for Law Enforcement Investigations and Intelligence, ICL Technical Journal November 1992, 1992.

[SPAR91] M. K. Sparrow. The Application of Network Analysis to Criminal Intelligence: an Assessment of the Prospects, Social Networks Vol. 13, 1991.

[THOM76] J. C. Thomas. Quantifiers and Question Asking, IBM Research Report RC-5866, 1976.

[TOMS] Tom Sawyer Software Limited. http://www.tomsawyer.com/glossary.html, correct as of the 1st of November 2003.

[WADL98] P. Wadler. Why No One Uses Functional Languages, SIGPLAN Notices Vol. 33 No. 8, 1998.

[WILL02] S. Williams. The Associative Model of Data, LazySoft Limited, ISBN 1903453011, 2002.

[WWWC] The World Wide Web Consortium. XML Path Language (XPath) Version 1.0, http://www.w3.org/TR/xpath, Correct as of the 1st of November 2003.

[XANA] Xanalys Limited. http://www.xanalys.com, correct as of the 1st of November 2003.

[ZLOO77] M. M. Zloof. Query-by-Example: A Data Base Language, IBM Systems Journal Vol. 16 No. 4, 1977.

Appendix A – User Evaluation Documents

A.1 The Pre-Evaluation Questionnaire

Personal Details

Name: ____

Email: ____

Occupation / Title of Course: ______________________________

Educational Background

Please tick the boxes that apply to you.

☐ I have obtained a GCSE or an equivalent qualification

☐ I have obtained an A level or an equivalent qualification

☐ I have obtained a qualification from a higher educational institution

☐ I have obtained a postgraduate qualification

☐ I have passed professional exams

Computing Experience

How often do you use computer software that has a graphical user-interface, such as that produced by Microsoft®?

☐ I have never used such software

☐ I have used such software on very few occasions

☐ I use such software occasionally

☐ I use such sofware frequently

Have you any experience of using a World Wide Web browser?

☐ I have never used such software

☐ I have used such software on very few occasions

☐ I use such software occasionally

☐ I use such sofware frequently

Have you any experience of using a database management system? Examples of this genre of software product include Microsoft® Access and Oracle®.

- ☐ I have never used such software
- ☐ I have used such software on very few occasions
- ☐ I use such software occasionally
- ☐ I use such sofware frequently

How do you use database software?

- ☐ I have never used such software
- ☐ I scroll or browse the data stored and/or execute predefined queries
- ☐ I form and/or execute queries using a language such as SQL, QBE or OQL
- ☐ I create and/or maintain databases

Do you have any knowledge of some of the underlying concepts used in database systems?

- ☐ Yes
- ☐ No

Have you any experience of using a procedural computer programming language? Examples of procedural computer programming languages include C, C++ and Java.

- ☐ I have never used such a computer programming language
- ☐ I have used such computer progamming languages

A.2 The Tutorial Booklet

A Tutorial for EDVC

Contents

1 Introduction

This booklet is a tutorial for the Exploratory Database View Constructor (EDVC), a database query interface for supporting Link Analysis. It is assumed that the reader has a working knowledge of the operating system installed on their computer and understands how to use menus and a mouse. The term *click* is used throughout this booklet to refer to a single click of the leftmost mouse button. When a double-click or the rightmost mouse button is required it will be explicitly stated.

2 Introducing EDVC

Data is often best comprehended when presented as a network if the connections between objects of interest, such as people or locations, are of as great an importance, if not more so, as the information known about the objects, such as the age or occupation of a person. For example, if we have the information that David is a friend of Andrea, who is the sister of Paul, who frequents the King George public house, as does David, Fiona and Edward, who works at Daltons bank where he has a colleague called Christine, who is married to Brian, who is the brother of Angela, who is Fiona's friend, then this information is best presented as a network such as that displayed in figure 2.1.

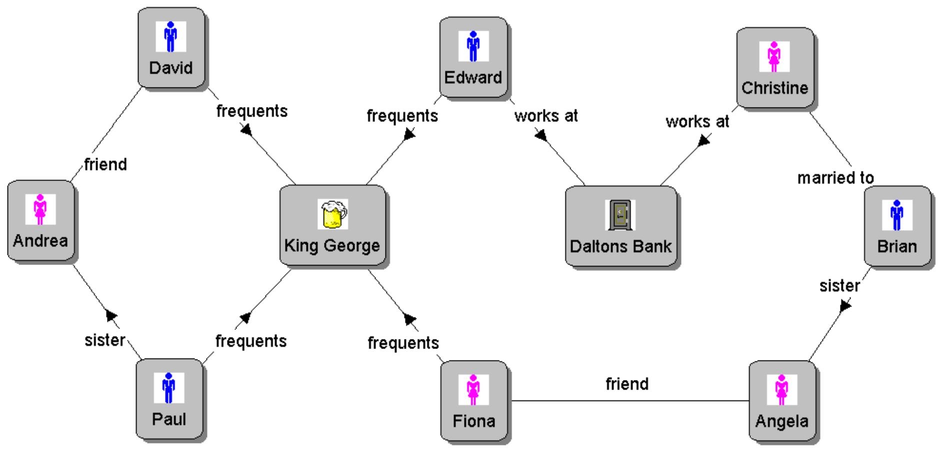


Figure 2.1. A network representing the connections between a set of people

Presenting such data in this manner increases comprehensibility, providing that the data set displayed is not too large, and enables the reader to establish if objects are connected. Thus, if we ask if there is a connection between Christine and Paul who we believe are not known to each other, the connections are easily seen.

One field where visualising a social network such as that displayed in figure 2.1 is of particular importance is in the analysis of data relating to criminal activities. The fundamental approach used is to incrementally construct a meaningful subset of the data gathered using a diagram similar to that in figure 2.1, which is referred to as a *link chart*, in order to better understand the connections between objects of interest.

Data such as that in figure 2.1 is often stored within a *database*, which can be thought of as a computerised filing cabinet that can answer clearly defined queries concerning the data stored. Thus, the objects and links displayed in a link chart, which are usually only a small subset of those known, often represent those stored within a database.

In practice link charts are usually constructed incrementally over a period of time. The objects and links included in a chart are often altered on numerous occasions during the lifetime of the chart, as part of the natural exploratory nature of the knowledge discovery process and to reflect the acquisition of new information. Thus, one can consider a link chart to represent the results of a number of queries posed to a database, such as *display all people who work at Daltons Bank,* and *display all paths that connect Paul and Edward*, each of which may add one or more objects or links to the chart. It is this incremental style of exploration of the data stored within a database that EDVC is designed to support.

3 Introducing the Tutorial Database

The database used throughout the remainder of this booklet for illustrative purposes is fictitious and stores information regarding the intelligence reports submitted by a set of police officers, the people and locations mentioned within those reports, and the associates of these people and the locations they frequent.

The way in which is data stored is stored within the database is analogous to the way in which data is displayed in figure 2.1. Each intelligence report, police officer, person and location is stored as an *object* and is uniquely identified by an associated piece of text referred to as an *object identifier*. Each link is explicitly stored within the database and may have an associated set of *attribute values* that represent information about the link.

There are no facilities provided in EDVC for editing the data stored. Therefore you can explore the facilities provided in EDVC, even those not described in this booklet, without being concerned about the affect your actions may have on the database.

4 Starting EDVC & Connecting to a Database

If you are using a version of Microsoft® Windows® double-click on the **EDVC** icon on the desktop, if you are using a UNIX based platform type **java edvc.Edvc** at the command line and press enter; an EDVC main window similar to the one in figure 4.1 will be displayed.

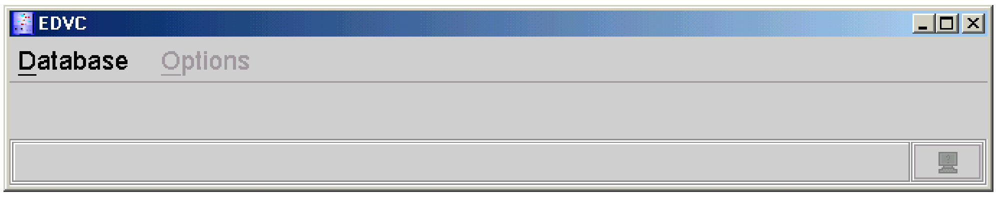


Figure 4.1. An EDVC main window

To connect to the example database select **open** from the **database** menu, a file chooser dialog will appear, then select **tutorial.dbf** and click the **open** button.

5 Exploring a Database

5.1 Introduction

EDVC provides two distinct styles of interface for querying the data stored. One of the interfaces is designed for exploring the database incrementally in the manner described in section 2, the other is designed for constructing a pattern that may be used to filter the objects and links stored in the database, or those displayed in a specified link chart, so that only those objects and links that match the pattern are displayed. This section provides an introduction to the first style of interface; the second is described in section 6.

5.2 Opening an Explorer

Select **explore** from the **database** menu in the EDVC main window; an explorer similar to the one in figure 5.2.1 will be displayed.

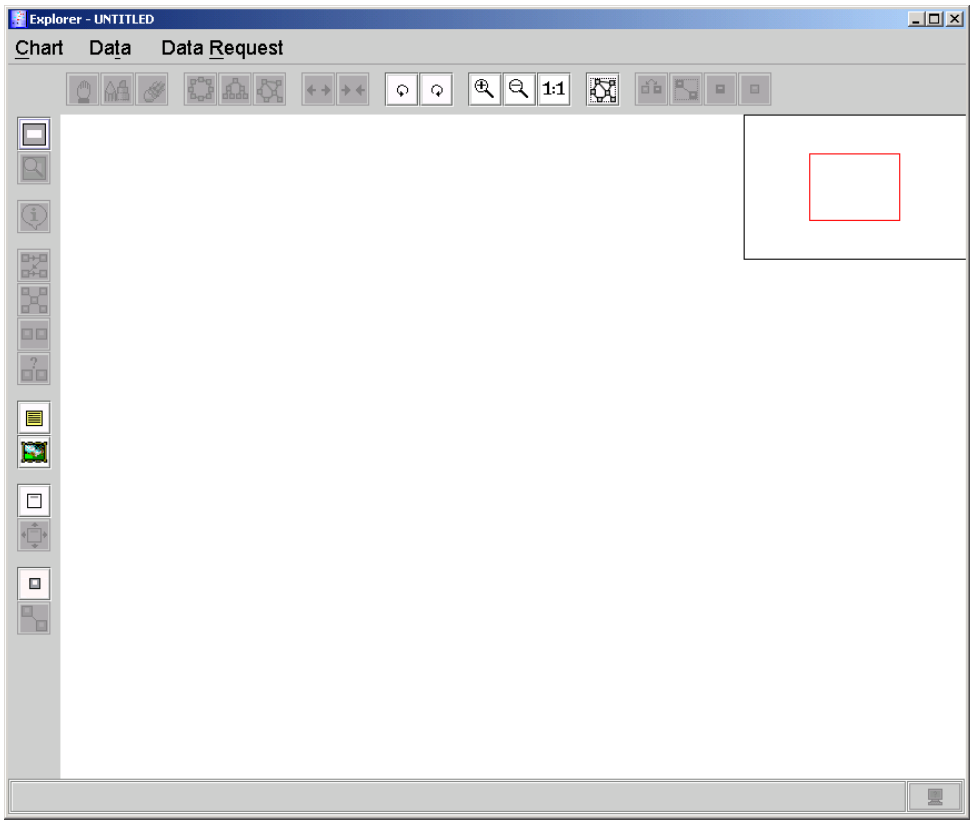


Figure 5.2.1. An explorer

The area on which a link chart is displayed is referred to as *the canvas*. When a link chart is too large to display within the canvas a section of the chart will be displayed. The section displayed may be altered by dragging the *view rectangle* (the smaller of the rectangles located in the top right corner of the canvas) to the appropriate position in the in the *pan window* (the larger of the rectangles located in the top right corner of the canvas), where a miniature version of the entire chart is displayed.

A short description of the facility represented by a particular menu item or toolbar option may be obtained by leaving the pointer over a menu item or a toolbar option for a short period of time. Where appropriate, instructions are given in the status bar (the panel located at the bottom of an explorer) when a toolbar option has been selected.

5.3 Displaying Objects by Counting Links

Not all of the people mentioned in the intelligence reports are necessarily criminals; some may be perfectly innocent such as witnesses or victims. However, if a particular person is mentioned in several reports, say at least four, that person may be worthy of further investigation.

1. Select **select by link count** from the **data request** menu, a dialog similar to the one in figure 5.3.1 will be displayed

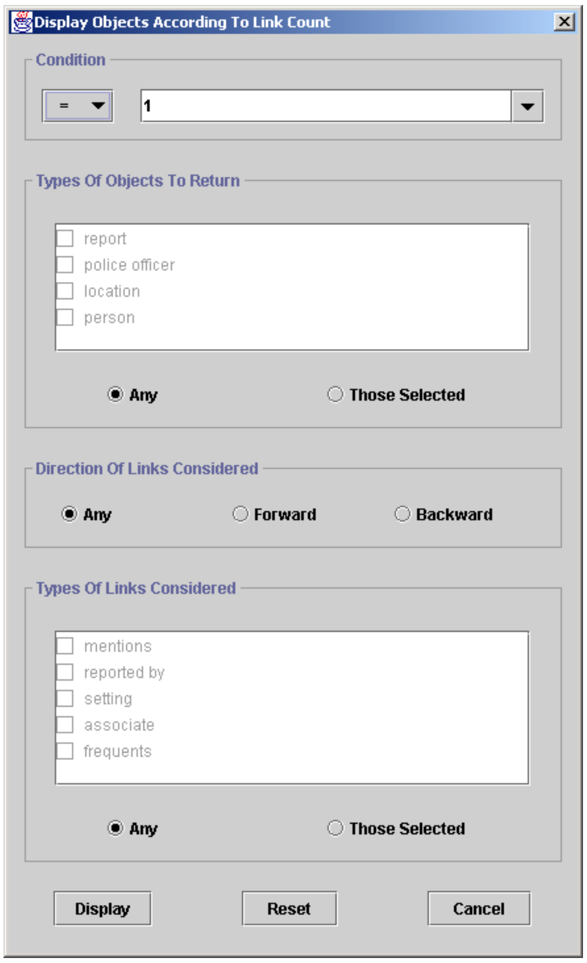


Figure 5.3.1. A dialog for displaying objects according to the number of links they are involved in

2. Select the **those selected** button in the **types of objects to return** panel

3. Select the **person** box in the list in the **types of objects to return** panel

4. Select the **those selected** button in the **types of links considered** panel

5. Select the **mentions** box in the list in the **types of links considered** panel

6. Select >= from the left list in the **condition** panel

7. Select **4** from the right list in the **condition** panel

8. Click the **display** button, a dialog similar to the one in figure 5.3.2 will be displayed

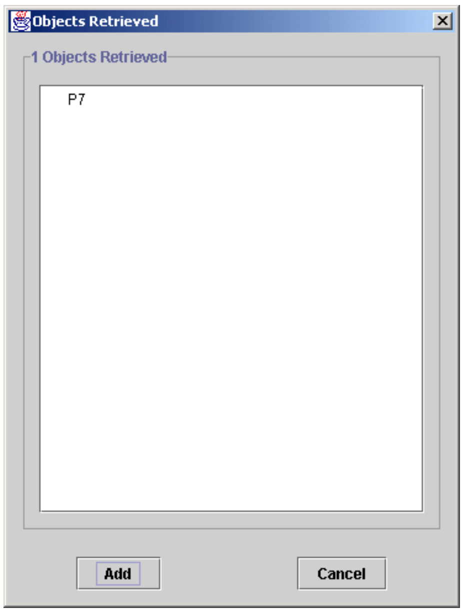


Figure 5.3.2. A dialog displayed after retrieving objects according to the number of links they are involved in

The police use identifiers such as P7 for a number of reasons. Meaningful *identifiers* such as national insurance numbers or car registration numbers may be duplicated by criminals, additionally there may be people that need to be stored in the database that you cannot identify or do not wish to identify, such as an undercover police officers.

We will add P7 to the chart so that we can investigate further.

1. Click on **P7**

2. Click the **add** button

The icon that appears beside P7 in the dialog indicates that the object is now displayed in the chart. This technique is used in all dialogs that display the results returned by a query facility. The blue circle displayed on the object representing P7 in the chart indicates that there are links stored in the database involving P7 and another object that are not currently displayed in the chart. Cancel the dialog by clicking the **cancel** button.

5.4 Displaying the Icon Key

P7 has automatically been given the icon associated with people in the database. To view the icons associated with all types of objects in the database select **view icon key** from the **data** menu, you may then close the dialog by clicking the **close** button.

5.5 Moving Objects & Values

As P7 is the focus of our investigation we will move P7 to the centre of the chart.

1. Select the option from the top toolbar

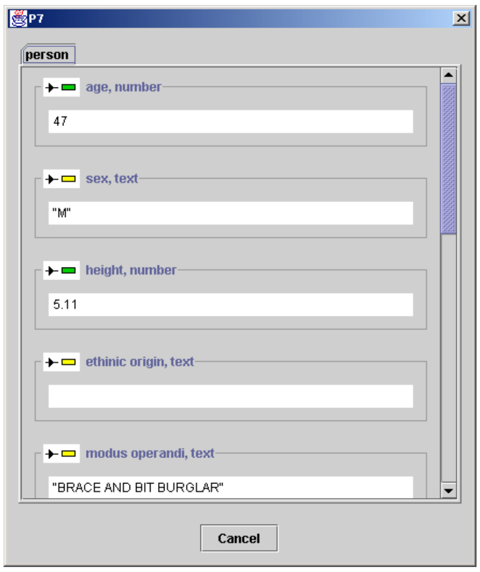
2. Drag **P7** to the desired location

5.6 Displaying Information Stored Directly About an Object

We will now see what information is stored directly about P7.

1. Select the option from the side toolbar

2. Click on **P7**, a dialog similar to the one in figure 5.6.1 will be displayed



age, number: 47

sex, text: "M"

height, number: 5.11

ethinic origin, text:

modus operandi, text: "BRACE AND BIT BURGLAR"

Figure 5.6.1. A dialog displaying the objects and values linked to P7

The dialog displays the objects and values (items stored within the database which are directly displayable such as text, numbers and pictures) linked to P7. Any of these objects or values, and the connecting links, may be added to the chart by either clicking on them, or where appropriate clicking the **add** button in the associated panel. If an object or value, and the connecting link(s), are already displayed in the chart the object or value will be highlighted, or where appropriate the button will be labelled *remove*. Display the id photo of P7 and the incident reports that P7 is mentioned.

1. Scroll down to the **id photo** panel then click the **add** button, the photograph and the associated link will be added to the chart

2. Click on all the reports listed in the **mentions** panel, the reports and the associated links will be added to the chart

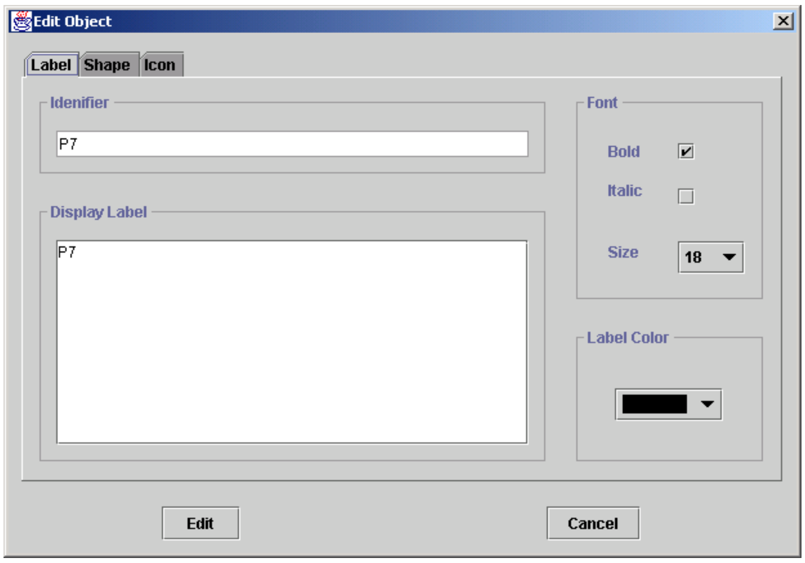
3. Click the **cancel** button

5.7 Editing the Visual Properties of an Object

P7 was automatically given the icon associated with all objects representing people and labelled with the object identifier P7. As we have seen a photograph of P7 and now know who P7 is, and what type of criminal he is, it would be helpful to reflect this in the chart by changing the label of the object (which only effects the object in this chart and not in any others) and the icon associated with the object.

1. Select the option from the top toolbar

2. Click on **P7**, a dialog similar to the one in figure 5.7.1 will be displayed


Edit Object

Label | Shape | Icon

Idenifier: P7

Display Label: P7

Font
Bold ☑
Italic ☐
Size 18

Label Color

Figure 5.7.1. A dialog for editing the visual properties of object P7

3. Enter **David Jones** with a carriage return between the two words in the area located in the **display label** panel

4. Select the **icon** tabbed pane

4. **Double-click** on the **crimes** folder in the **icons available** panel and select the **burglary** icon

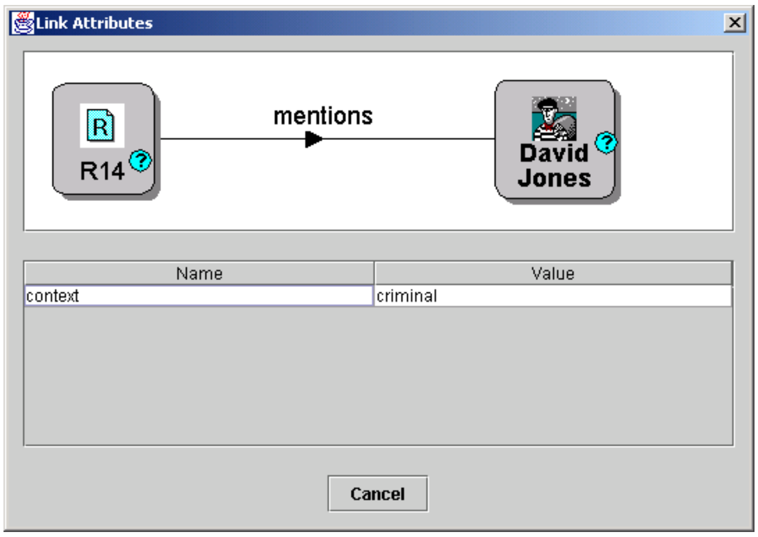
5. Click the **edit** button

5.8 Displaying the Attribute Values of a Link

The links displayed in the chart that link David Jones (formerly P7) with the intelligence reports that he is mentioned have attribute values which specify the context in which David is mentioned in a report, such as being mentioned as a criminal, a witness or a victim. To display the attribute values of a link:

1. Select the ⓘ option from the side toolbar

2. Click on the label of the link (the text associated with the link) or the arrow, a dialog similar to the one in figure 5.8.1 will be displayed



Name	Value
context	criminal

Figure 5.8.1. A dialog displaying the attribute values of the link connecting R14 and David Jones

When you have viewed the information displayed cancel the dialog by clicking the **cancel** button.

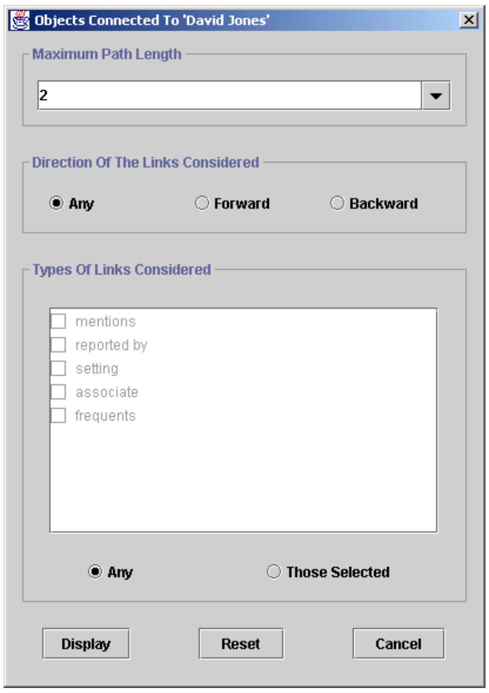
5.9 Displaying the Objects Connected to an Object

We will now explore all those people and reports directly or indirectly connected to David Jones.

1. Select the option from the side toolbar then click on **David Jones**, a dialog similar to the one in figure 5.9.1 will be displayed

2. Select the **those selected** button in the **types of links considered** panel

3. Select the **mentions** box in the list in the **types of links considered** panel


Objects Connected To 'David Jones'

Maximum Path Length: 2

Direction Of The Links Considered: [✓ Any] Forward Backward

Types Of Links Considered:
- mentions
- reported by
- setting
- associate
- frequents

[✓ Any] Those Selected

Figure 5.9.1. A dialog for displaying the objects connected to David Jones.

5. Select **3** from the list in the **maximum path length** panel to limit the context of our search

6. Click the **display** button

5.10 Altering the Appearance of a Link Chart

The layout of the link chart has now been degraded, we can either move the objects to better positions (as described in section 5.5), or we can use the automatic layout facilities provided. Three layout are provided: circular, hierarchical and another for laying out the chart with a minimal number of links crossing. To use these facilities select the appropriate option from the top toolbar.

The appearance of a link chart may also be altered by selecting the following options from the top toolbar: the zoom in and zoom out options may be used for enlarging or decreasing the display size of a link chart, selecting the 1:1 option returns the display size back to the original setting, the expand ↔ and shrink options may be used for increasing or decreasing the distance between the elements in a link chart, and the rotate clockwise and rotate anti-clockwise options may be used to rotate the elements in a link chart.

5.11 Removing Elements

Not all of the objects and links displayed by facilities such as that described in section 5.9 may be of interest. Any element on a link chart may be removed by selecting the option from the top toolbar then clicking on the element.

5.12 Multiple Views of a Database

If a link chart contains a large number of elements the comprehensibility of the information displayed may be degraded. Therefore it can be useful to have several charts open at the same time, each displaying information from the same database but reflecting a different aspect of the underlying data. EDVC provides support for this by allowing the user to have several explorers open at the same time.

5.13 Copying Elements

Elements from one link chart may be copied to another chart. We will copy David Jones to a new link chart.

1. Open a new explorer (described in section 5.2)

2. Select the option from the top toolbar in the original explorer

3. Click on **David Jones**, the background and outline colour of the object will change

4. **Right-click** on **David Jones**, a dotted outline of that object will follow the pointer until another action is performed

5. **Right click** on the chart displayed in the explorer that was just opened, the object will be copied to the chart

An analogous procedure to this may be used to copy elements from an object neighbourhood viewer into an explorer. An object neighbourhood viewer allows the objects and links directly and indirectly connected to a selected object to be incrementally explored. To activate this facility:

1. Select the option from the side toolbar

2. Click on an object whose neighbourhood is to be explored

5.14 Saving a Link Chart

The chart displaying the people and intelligence reports directly or indirectly connected to David Jones by a path of length three or less will not be used in the remainder of the tutorial, save the chart and exit the associated explorer.

1. Select **save as** from the **chart** menu

2. Enter an appropriate name for the chart

3. Click the **save** button

4. Select **exit** from the **chart** menu

5.15 Adding Objects Directly to a Link Chart

We suspect that David Jones and P5 are members of a criminal gang so it would be helpful to ascertain if, and how, they are connected. First we must add P5 to the chart.

1. Select the option from the side toolbar, a dialog similar to the one in figure 5.15.1 will be displayed

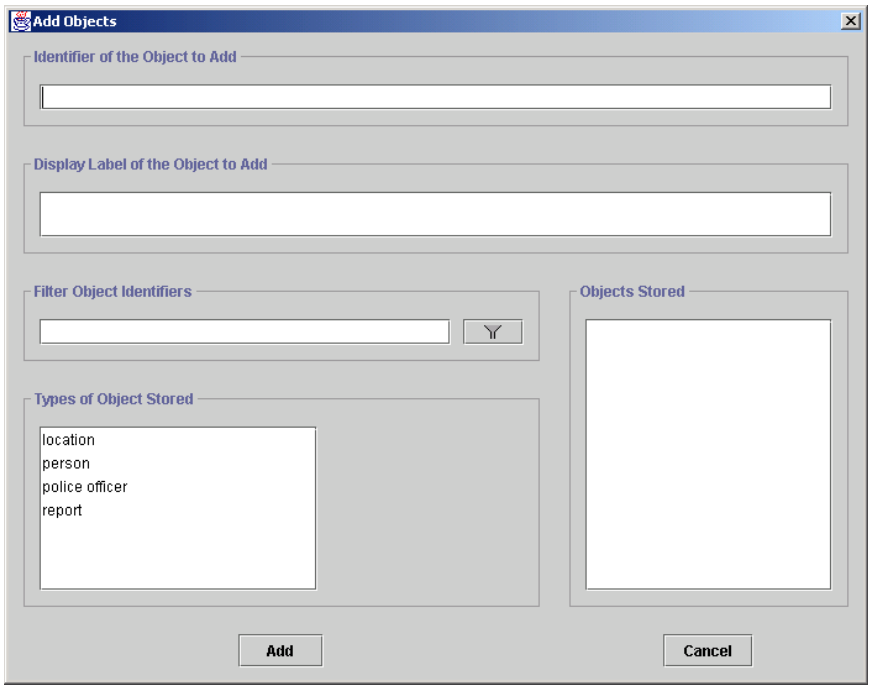

Add Objects
Identifier of the Object to Add
Display Label of the Object to Add
Filter Object Identifiers
Objects Stored
Types of Object Stored
location
person
police officer
report
Add
Cancel

Figure 5.15.1. A dialog for adding objects stored in the database

2. Select **person** from the list in the **types of object stored** panel

3. Select **P5** from the list in the **objects stored** panel

4. Click the **add** button

5. Click the **cancel** button

5.16 Displaying the Paths that Connect Two Objects

We will now investigate so see if, and how, David Jones and P5 are connected.

1. Select the option from the side toolbar

2. Click on **P5** then on **David Jones**, a dialog similar to the one in figure 5.16.1 will be displayed

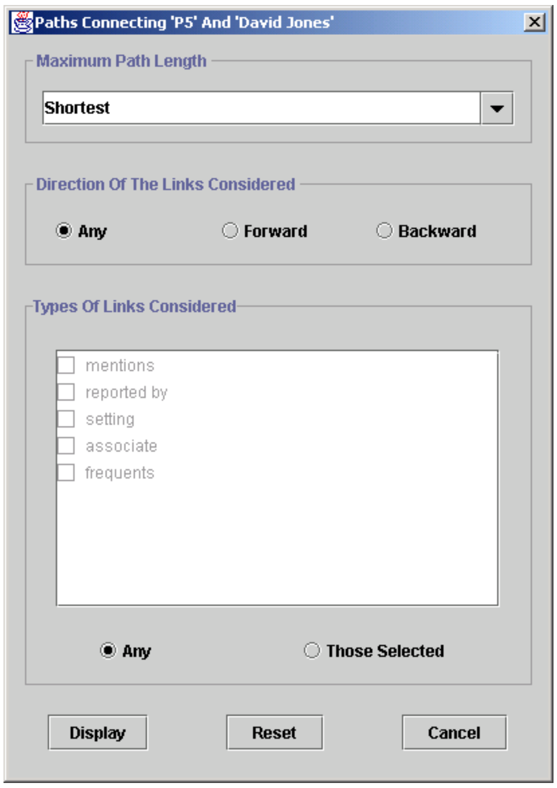


Figure 5.16.1. A dialog for displaying the paths that connect David Jones and P5

3. Select the **those selected** button in the **types of links considered** panel

4. Select the **mentions** box in the list in the **types of links considered** panel

5. Select **4** from the list in the **maximum path length** panel; the shortest path may not necessarily be the one of most interest

6. Click the **display** button

5.17 Displaying the Objects Connected to a Group of Objects

The paths connecting David Jones and P5 all involve person P6; hence P6 may also be part of the same gang that David Jones and P5 are suspected of being involved in. One way to identify other possible gang members and relevant intelligence reports is to display people and reports that are connected to these three people.

1. Select the option from the top toolbar

2. Click on **David Jones**, **P6** and **P5**

3. Select **center** from the **data request** menu, a dialog similar to the one in figure 5.17.1 will be displayed

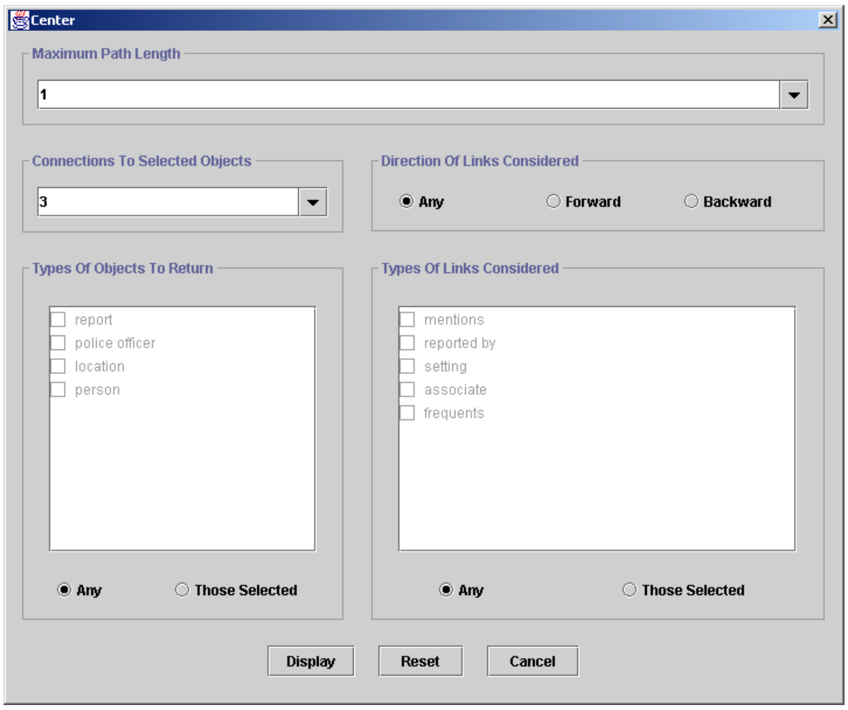


Figure 5.17.1. A dialog for displaying the objects connected to a group of selected objects

4. Select the **those selected** button in the **types of objects to return** panel

5. Select the **person** and **report** boxes in the **types of objects to return** panel

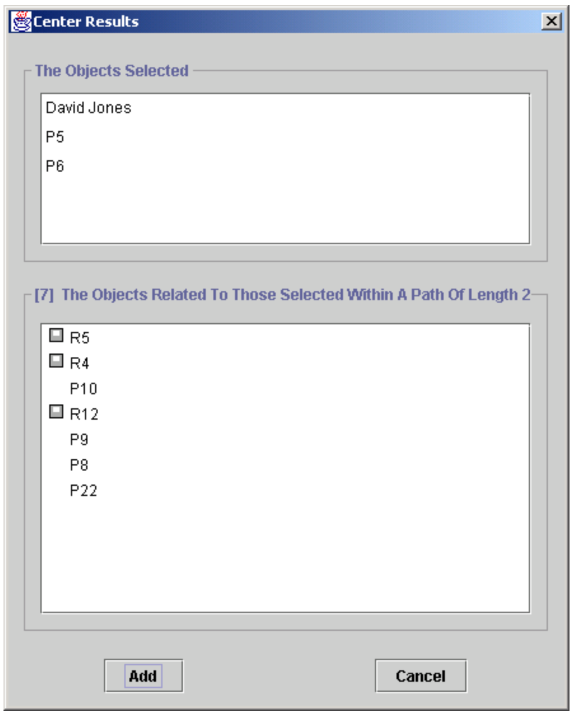
6. Select the **those selected** button in the **types of links considered** panel

7. Select the **mentions** box in the list in the **types of links considered** panel

8. Select **2** from the list in the **connections to selected objects** panel, this indicates that the objects displayed must be connected to at least two of the objects selected

9. Select **2** from the list in the **maximum path length** panel

10. Click the **display** button, a dialog similar to the one in figure 5.17.2 will be displayed


Center Results
The Objects Selected
[7] The Objects Related To Those Selected Within A Path Of Length 2
Add
Cancel

David Jones
P5
P6

☐ R5
☐ R4
P10
☐ R12
P9
P8
P22

Figure 5.17.2. A dialog displaying objects connected to David Jones, P5 and P6

Add the objects displayed in the bottom panel in the dialog to the chart. For each object displayed:

1. Click on the object
2. Click the **add** button, a dialog will be displayed asking if you would like to display the paths connecting the object to the objects originally selected
3. Click the **yes** button

Cancel the dialog by clicking the **cancel** button.

5.18 Displaying the Objects Similar to an Object

Criminals often use aliases. Hence, it is possible for several objects in a database to represent the same person, each representing a different identity that the criminal uses. As David Jones is highly connected and possibly significant we will investigate to see if he has any aliases.

1. Select the ▫▫ option from the side toolbar
2. Click on **David Jones**, a dialog similar to the one in figure 5.18.1 will be displayed

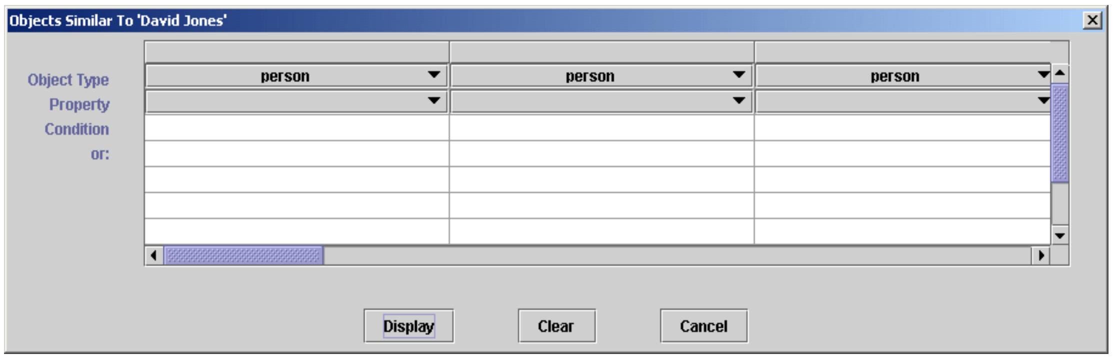
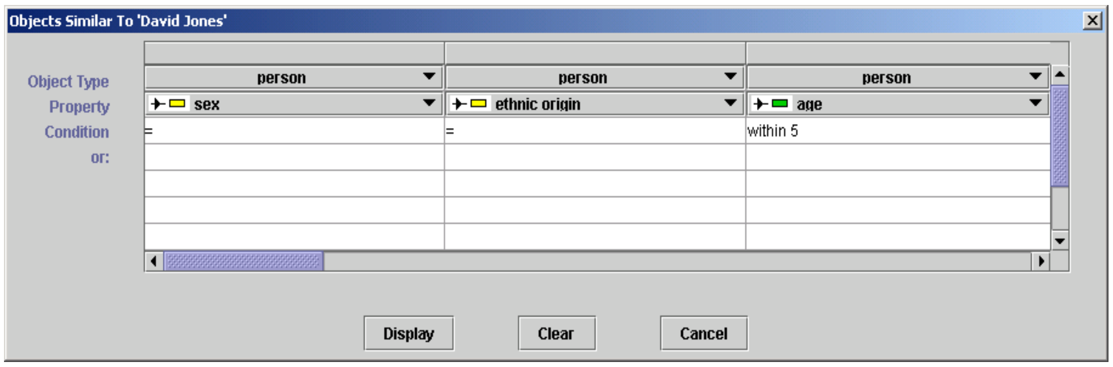


Figure 5.18.1. A dialog for displaying objects similar to David Jones

The grid located within this dialog will contain the similarity conditions that are used to deem if an object is similar to David Jones. Each grid cell may contain at most a single similarity condition. A similarity condition may either relate to the information stored about the object selected, such as *the age of all similar objects must be within 5 years of the person selected*, or a more general condition, such as *the height of all similar objects must be greater than seventy two inches*. For an object to be deemed similar the object must satisfy all the conditions specified in at least one of the rows in the grid. For example, the grid specified in figure 5.18.2 would retrieve all people whose sex and ethnic origin is the same as David Jones, and whose age is within 5 years of David's age.


Objects Similar To 'David Jones'

Object Type	person	person	person
Property	sex	ethnic origin	age
Condition	=	=	within 5
or:			

Figure 5.18.2. A dialog that would display all people whose sex and ethnic origin is the same as David Jones, and whose age is within five years of David's age

It is this example that we will now use.

3. Select **sex** from the list in the first column

4. Click on the leftmost grid cell in the first row, a dialog similar to the one in figure 5.18.3 will be displayed.

5. Select = from the **operator** list

6. Click the **set** button.

7. Select **ethnic origin** from the list in the second column

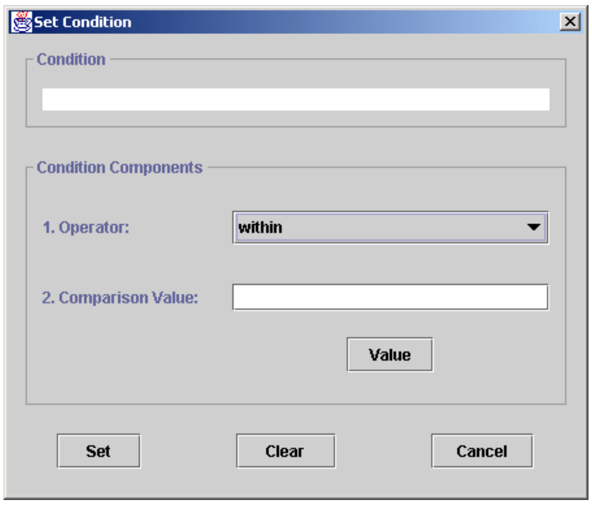


Figure 5.18.3. A dialog for editing a similarity condition

8. Click on the grid cell second from the left in the first row, a dialog similar to the one in figure 5.18.3 will be displayed

9. Select = from the **operator** list

10. Click the **set** button.

11. Select **age** from the list in the third column

12. Click on the grid cell third from the left in the first row, a dialog similar to the one in figure 5.18.3 will be displayed

13. Select **within** from the **operator** list, a dialog asking you to select a number will appear

14. Select **5** from the list and click the **select** button

15. Click the **set** button

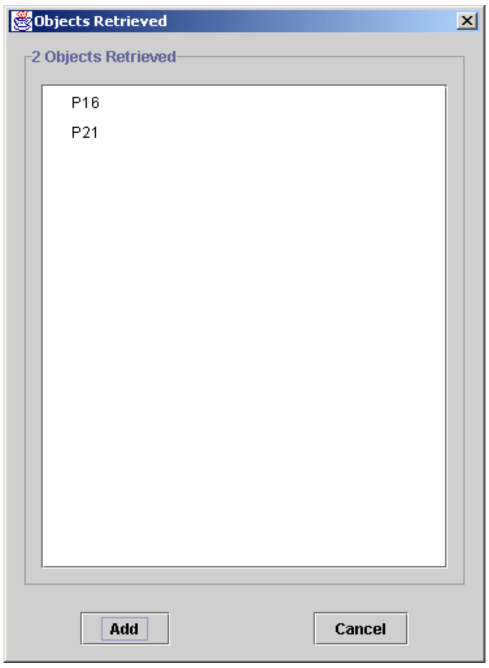
16. Click the **display** button, a dialog similar to that displayed in figure 5.18.4 will be displayed

Add both these objects to the chart. For each object displayed:

1. Click on the object

2. Click the **add** button

Cancel the dialog by clicking the **cancel** button.



P16
P21

Figure 5.18.4. A dialog displaying all the objects that were deemed similar to David Jones

5.19 Ascertaining How Similar Two Objects Are

We will now investigate to see how similar the objects that were retrieved are to David Jones.

1. Select the option from the side toolbar

2. Click on **David Jones** then on **P21**, a dialog similar to the one in figure 5.19.1 will be displayed

The dialog displays all those objects and values linked to both the objects selected by the same type of link. Any of these objects or values, and the connecting links, may be added to the chart by either clicking on them or where appropriate clicking the **add** button in the associated panel. If an object or value, and the connecting link(s), are already displayed in the chart the object or value will be highlighted or where appropriate the button will be labelled *remove*.

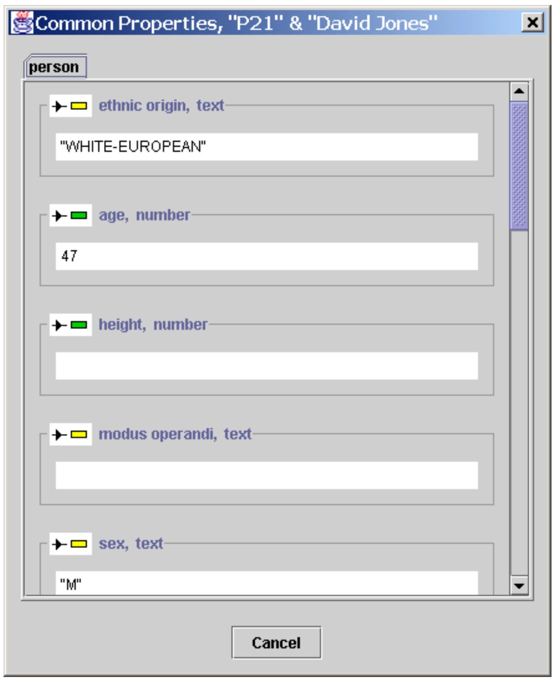


Figure 5.19.1. A dialog displaying those objects and values linked to David Jones and P21 by the same type of link

5.20 Adding Elements Not Stored in the Database

Elements other than those representing objects and links stored in the database may be added to a link chart. The add text note option is used to add a piece of text, the add labelled box option is used to add a labelled box (typically used to indicate the grouping of objects), the add picture from file option is used to add a picture from a file in either Graphics Interchange Format (GIF) or Joint Photographic Experts Group (JPEG) format, and the add object and add link options are used to add objects and links that are not currently stored in the database.

6 Filter Patterns

6.1 Introduction

The facilities provided in the EDVC explorer are designed to support the queries that are often asked of databases such as the one used in this booklet. However, it is not possible to predict all the queries that may be asked of such a database. Therefore, a second style of interface is provided that allows the user to construct a pattern that may be used to filter the objects and links stored, or those displayed in a specified link chart, so that only those objects and links that meet the conditions specified in the pattern are displayed.

For example, if a robbery had taken place and three men were seen leaving the scene of the crime, one of whom was aged in his late forties and another in his late fifties, displaying the people who match these descriptions, the reports they are mentioned in, and the people mentioned in the same reports could prove of value as we could identify possible suspects, ascertain if they have any previous convictions for this type of crime, and identify possible accomplices. It is this example that is used in the following subsections.

6.2 Opening a Filter Pattern Editor

As we wish to consider all the people and reports stored, we shall filter the objects and links stored in the database, not just those displayed in a specified link chart. Select **filter** from the **database** menu in the EDVC main window, a filter pattern editor similar to the one in figure 6.2.1 will be displayed.

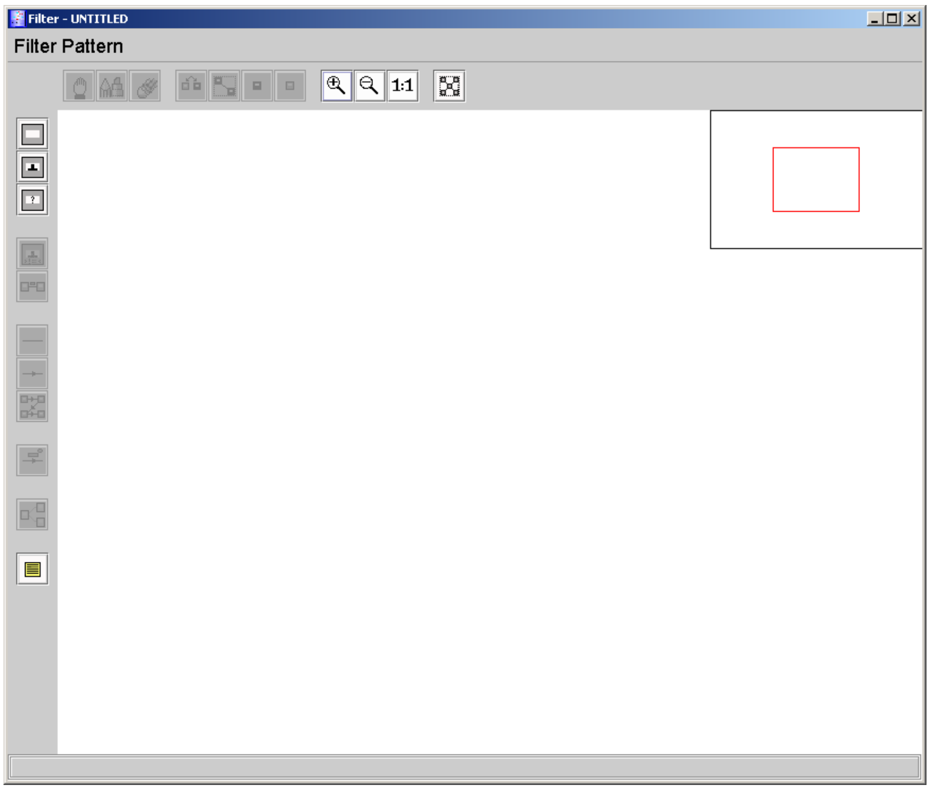


Figure 6.2.1. A filter pattern editor displaying an empty filter pattern

A filter patterns consists of a set of constraints, of which there are several types.

6.3 Adding Object Constraints

Object constraints are used to filter objects. An object constraint may match a particular object, such as the object P7, all objects of a specified type, such as all objects representing people, or all objects regardless of type. We will use three object type constraints, one to represent the people matching the descriptions given by the witnesses, one to represent the reports these people are mentioned in, and another to represent the people mentioned in the same reports.

1. Select the option from the side toolbar

2. Click on the filter pattern (the area in the middle of the filter pattern editor), a dialog similar to the one in figure 6.3.1 will be displayed

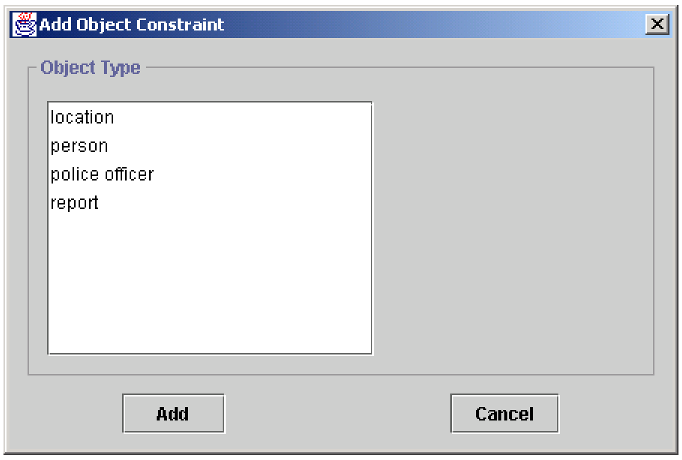


Figure 6.3.1. A dialog for adding an object constraint to a filter pattern

3. Select **person** from the list in the **object type** panel

4. Click the **add** button, an object constraint similar to the one in figure 6.3.2 will be added to the filter pattern

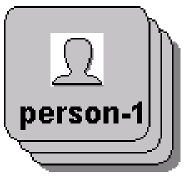


Figure 6.3.2. An object constraint that matches all objects representing people

5. Click on the filter pattern, a dialog similar to the one in figure 6.3.1 will be displayed

6. Select **report** from the list in the **object type** panel

7. Click the **add** button, an object constraint similar to the one in figure 6.3.2 but displaying the icon associated with reports and labelled *report-1* will be added to the filter pattern

8. Click on the filter pattern, a dialog similar to the one in figure 6.3.1 will be displayed

9. Select **person** from the list in the **object type** panel

10. Click the **add** button, an object constraint similar to the one in figure 6.3.2 but labelled *person-2* will be added to the filter pattern

Object constraints that match any object regardless of type, and those that match a particular object, are added to a filter pattern in a similar manner by selecting the ▫ and the ▭ options from the side toolbar.

6.4 Adding Link Constraints

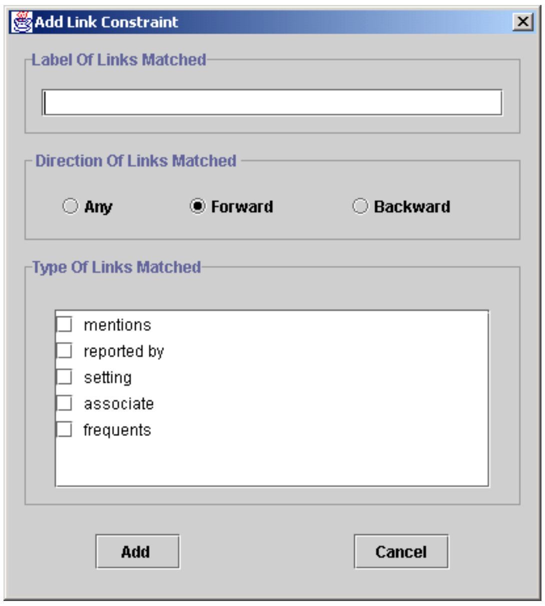
The filter pattern that we have constructed thus far would return all objects stored in the database representing people and all objects representing intelligence reports. To specify that the objects matched by these constraints should be connected we need to add link constraints to the filter pattern.

Link constraints appear in a filter pattern as an arc connecting two object constraints, and specify that the objects matched by the object constraints should be linked in a particular manner. We will add two link constraints to the filter pattern to indicate that the objects representing people that are matched by the object constraints *person-1* and *person-2* must be mentioned in the same report.

1. Select the → option from the side toolbar

2. Click on the object constraint **report-1**, the background colour of the constraint will change

3. Click on the object constraint **person-1**, a dialog similar to that in figure 6.4.1 will be displayed


Add Link Constraint

Label Of Links Matched

Direction Of Links Matched: Any [✓ Forward] Backward

Type Of Links Matched:
- ☐ mentions
- ☐ reported by
- ☐ setting
- ☐ associate
- ☐ frequents

Figure 6.4.1. A dialog for selecting the type of link that a link constraint should match

The links matched by a link constraint may be constrained by specifying the type of links that should be matched.

4. Select the **mentions** box in the list in the **types of links matched** panel

5. Click the **add** button, a link constraint similar to the one given in figure 6.4.2 will be added to the filter pattern

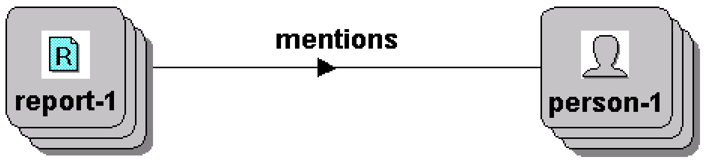


Figure 6.4.2. A link constraint connecting the object constraints report-1 and person-1

6. Repeat steps 1 to 5 except select the constraints **report-1** and **person-2**, the filter pattern should now look like the one displayed in figure 6.4.3

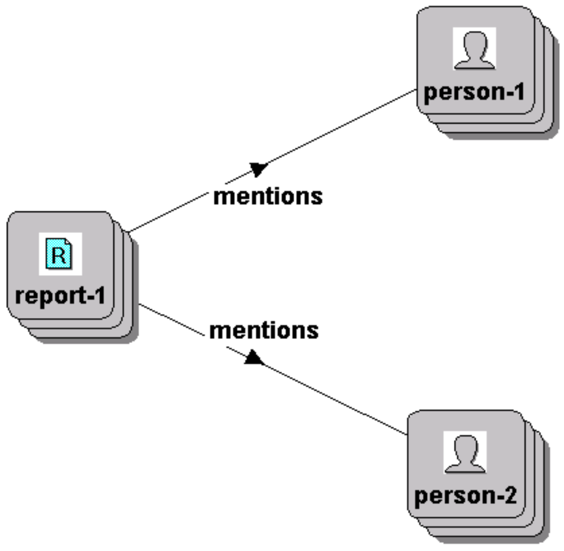


Figure 6.4.3. A filter pattern that would display all those people, and the associated reports and links, which are mentioned in the same intelligence report

The link constraints we have added to the filter pattern take the direction of the links matched into consideration during the matching process. However, link constraints do not have to take the direction of the links matched into consideration. When adding a link constraint to a filter pattern if the **any** option is selected in the **direction of links matched** panel in the dialog that appears when the object constraints have been selected the direction of the links matched by the constraint is not taken into consideration during the matching process.

6.5 Specifying Object Constraint Filter Conditions

To constrain the objects representing people that are matched by the object constraint *person-1* to only those who match the descriptions given by the witnesses we need to specify a set of filter conditions.

1. Select the option from the side toolbar

2. Click on the constraint **person-1**, a dialog similar to the one in figure 6.5.1 will be displayed

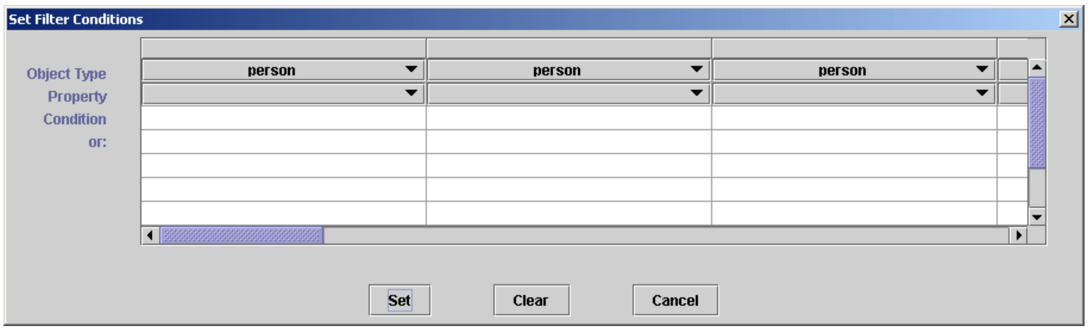


Figure 6.5.1. A dialog for specifying object constraint filter conditions

The grid located within this dialog will contain the filter conditions used to deem if an object is matched by the constraint. Each grid cell may contain at most a single filter condition. For an object to be matched by the constraint the object must satisfy all the conditions specified in at least one of the rows in the grid.

3. Select **sex** from the list in the first column

4. Click on the leftmost grid cell in the first row, a dialog similar to the one in figure 6.5.2 will be displayed

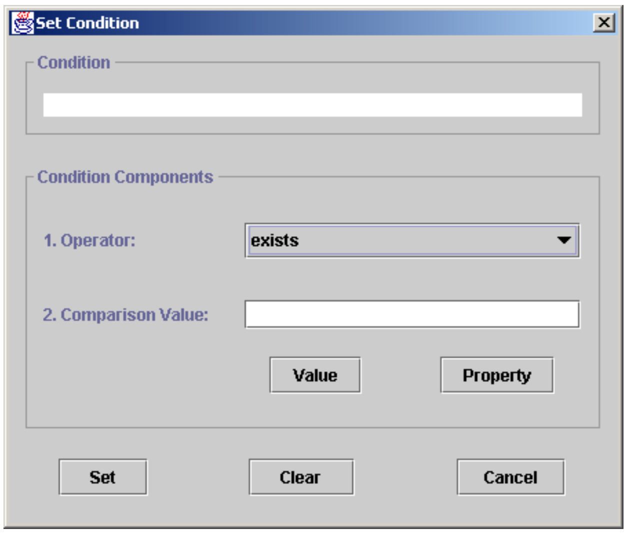


Figure 6.5.2. A dialog for setting a filter condition

5. Select = from the **operator** list

6. Enter **"M"** in the **comparison value** field then hit the **enter** key, *"M"* will appear alongside = in the condition field

7. Click the **set** button

8. Select **age** from the list in the second column

9. Click on the grid cell second from the left in the first row, a dialog similar to the one in figure 6.5.2 will be displayed

10. Select >= from the **operator** list

11. Enter **45** in the **comparison value** field then hit the **enter** key, *45* will appear alongside >= in the condition field

12. Click the **set** button

13. Select **age** from the list in the third column

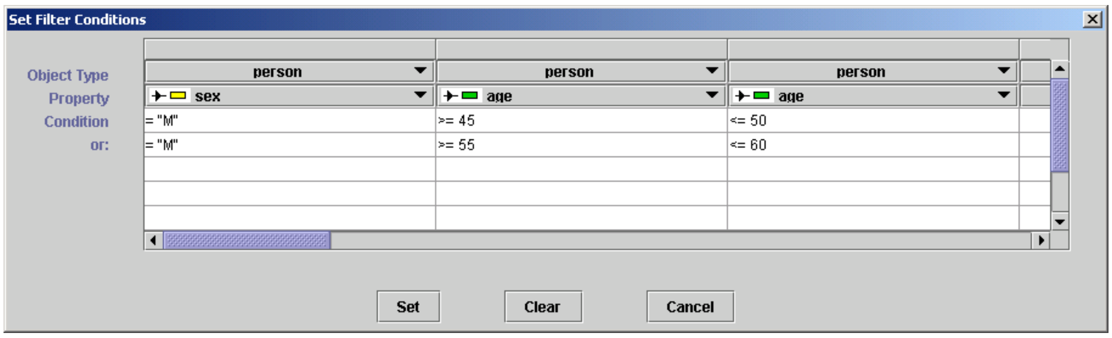
14. Click on the grid cell third from the left in the first row, a dialog similar to the one in figure 6.5.2 will be displayed

15. Select <= from the **operator** list

16. Enter **50** in the **comparison value** field then hit the **enter** key, *50* will appear alongside <= in the condition field

17. Click the **set** button

18. Perform the actions listed in steps 4 to 7, 9 to 12, and 14 to17; except use the cells in the second row and use the values *55* and *60* instead of *45* and *50*, the dialog should now look like the one displayed in figure 6.5.3



Object Type	person	person	person
Property	sex	age	age
Condition	= "M"	>= 45	<= 50
or:	= "M"	>= 55	<= 60

Figure 6.5.3. The dialog used to specify the filter conditions for the object constraint person-1 after the filter conditions have been entered

19. Click the **set** button, a magenta circle enclosing a C should now appear next to the label *person-1* to indicate that the object constraint has filter conditions

6.6 Specifying Link Constraint Filter Conditions

As described in section 3, a link may have a set of attribute values that represent the information stored directly about the link. The links representing that a person is mentioned in a report in the tutorial database have an attribute that signifies the context in which the person is mentioned in a report, such as a witness, a victim or a criminal. As we wish the objects matched by the object constraint person-1 to be mentioned as a criminal we will now specify a filter condition to signify this.

1. Select the ⇢ option from the side toolbar

2. Click on the link constraint connecting the object constraints **report-1** and **person-1**, a dialog similar to the one in figure 6.6.1 will be displayed

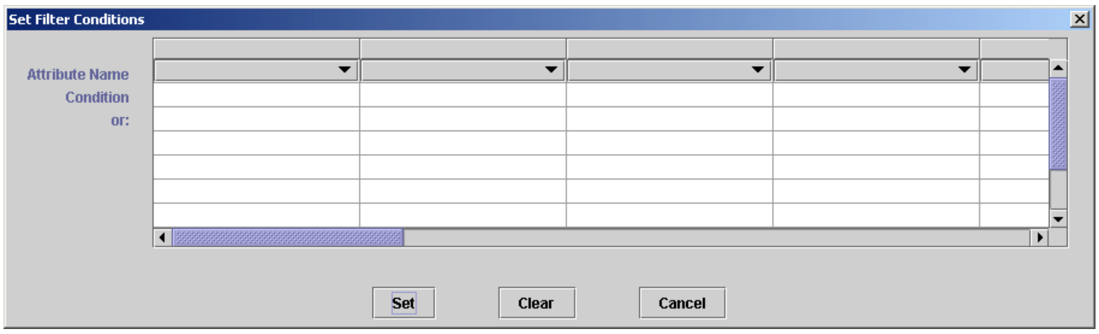


Figure 6.6.1. A dialog for specifying link constraint filter conditions

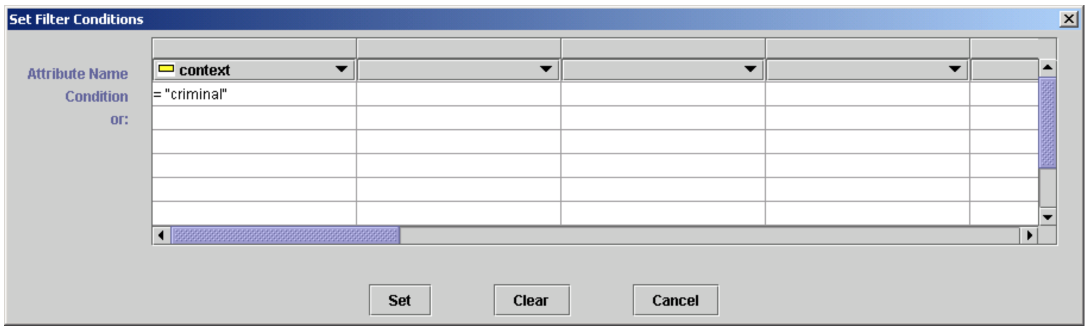
3. Select **context** from the list in the first column

4. Click on the leftmost grid cell in the first row, a dialog similar to the one in figure 6.5.2 will be displayed

5. Select = from the **operator** list

6. Enter **"criminal"** in the **comparison value** field then hit the **enter** key, *"criminal"* will appear alongside = in the condition field

7. Click the **set** button, the dialog should now look that that displayed in figure 6.6.2



Attribute Name	context			
Condition	= "criminal"			

Figure 6.6.2. The dialog used to specify the filter conditions for the link constraint connecting the object constraints report-1 and person-1 after the filter conditions have been entered

8. Click the **set** button

6.7 Adding an Object Comparison Constraint

Object comparison constraints specify that the objects matched by two object constraints should differ or be identical. The filter pattern we have constructed thus far would display all reports mentioning a male whose age is between 45 and 50 or between 55 and 60. To specify that we only wish to see reports that mention at least two people, one of whom is male and aged between 45 and 50 or between 55 and 60, we need to add an object comparison constraint between the object constraints person-1 and person-2 to dictate that the objects matched by these constraints should differ.

1. Select the ▭=▭ option from the side toolbar

2. Select the object constraint **person-1**

3. Select the object constraint **person-2**, a dialog similar to that in figure 6.7.1 will be displayed

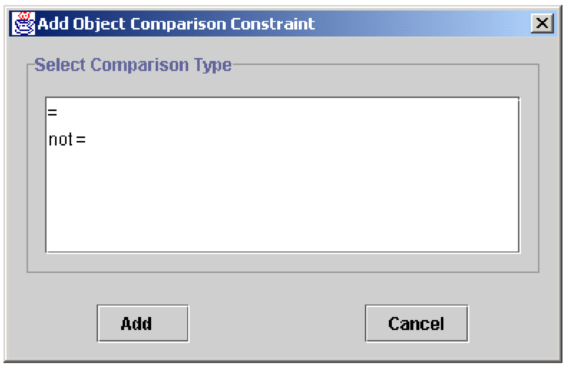


Figure 6.7.1. The dialog used to select the type of comparison represented by an object comparison constraint

4. Select **not** = from the **select comparison type** list

5. Click the **add button**, an object comparison constraint similar to the one displayed in figure 6.7.2 will be added to the filter pattern

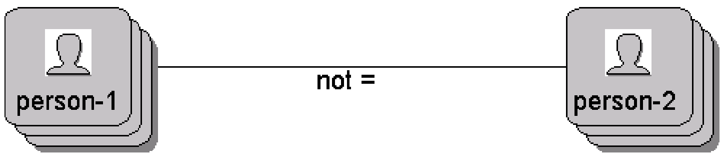


Figure 6.7.2. An object comparison constraint connecting the object constraints person-1 and person-2 that specifies that the objects matched by the constraints should differ

6.8 Displaying Objects & Links that Match a Filter Pattern

Select **display** from the **filter pattern** menu; an explorer displaying a link chart containing the objects and links matching the filter pattern will be displayed.

6.9 Excluding Constraints

By default all objects and links matched by the constraints in a filter pattern are displayed in the resulting link chart. However, a constraint may be used to filter the objects or links stored without contributing to those subsequently displayed. For example, if we only wished to display the people that are mentioned in the same reports as a male in his late forties or late fifties then we could exclude all the objects and links matched by the constraints in the filter pattern we have constructed thus far except the object constraint *person-2*.

1. Select the option from the side toolbar

2. Click on all the object and link constraints except the object constraint **person-2**, the background colour of the constraints will change to light grey

3. Select **display** from the **filter pattern** menu, an explorer displaying a link chart containing the people matched by the filter pattern will be displayed

6.10 Adding Path Constraints

Only displaying people mentioned in the same report as a male in his late forties or fifties could prevent relevant suspects in the robbery from being displayed as the accomplice may be connected to the robbers in some other way, such as being an associate or frequenting the same location as an associate. Therefore we will edit the filter pattern we have constructed thus far to display all people who are closely connected with males in their late forties or fifties.

Specifying such connections using just link constraints is difficult as the length and structure of a connecting path cannot be predicted. For specifying these types of connections path constraints are better suited. Path constraints appear in a filter pattern as a dashed arc connecting two object constraints, and specify that the objects matched by the object constraints should be connected. The connecting path(s) may be the shortest path(s) connecting the objects or those that contain less than a specified number of links. They may also constrained using a path filter, which is described in section 6.11.

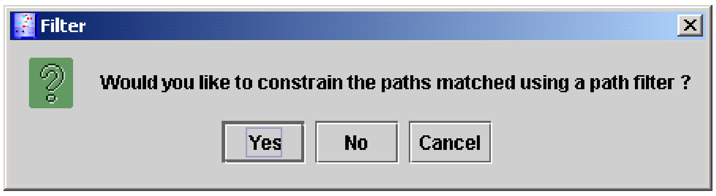
1. Select the option from the top toolbar

2. Click on the link constraint connecting the object constraints **report-1** and **person-2**, the constraint will be removed

3. Select the option from the side toolbar

4 Click on the object constraint **person-1**, the background colour of the constraint will change

5. Click on the object constraint **person-2**, a dialog similar to that in figure 6.10.1 will be displayed

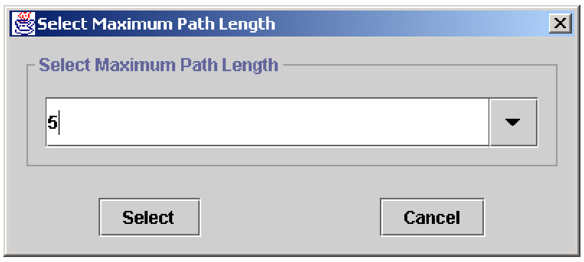


Would you like to constrain the paths matched using a path filter ?

Figure 6.10.1. A dialog for specifying if the paths matched by a path constraint should be constrained using a path filter

As we want to the path constraint to match any type of path connecting the objects matched by the object constraints we will not specify a path filter.

6. Click on the **no** button, a dialog similar to that in figure 6.10.2 will be displayed



Select Maximum Path Length: 5

Figure 6.10.2. A dialog for specifying the maximum number of links that may be contained in a path matched by a path constraint

As we want people to be closely connected, we will restrict the paths matched to only those with 3 or less links.

7. Select **3** from the list in the **select maximum path length** panel

8. Click the **select** button, the path constraint will be added to the filter pattern

9. Select **display** from the **filter pattern** menu, an explorer displaying a link chart containing the people matched by the filter pattern will be displayed

6.11 Constructing Path Filters

As described in section 6.10, a path filter may be used to constrain the paths matched by a path constraint. A path filter is represented as a set of object and link constraints, which are similar to the object and link constraints described in sections 6.3 and 6.4 for filter patterns. When it has been specified that the paths matched by a path constraint should be constrained using a path filter a window similar to the one in figure 6.11.1 is displayed.

The toolbar options provided in this window are similar to those described previously for filter pattern editors. The object constraints that have by default been added to the path filter represent the object constraints connected by the associated path constraint. These constraints are always highlighted and are referred to as the start and finish states.

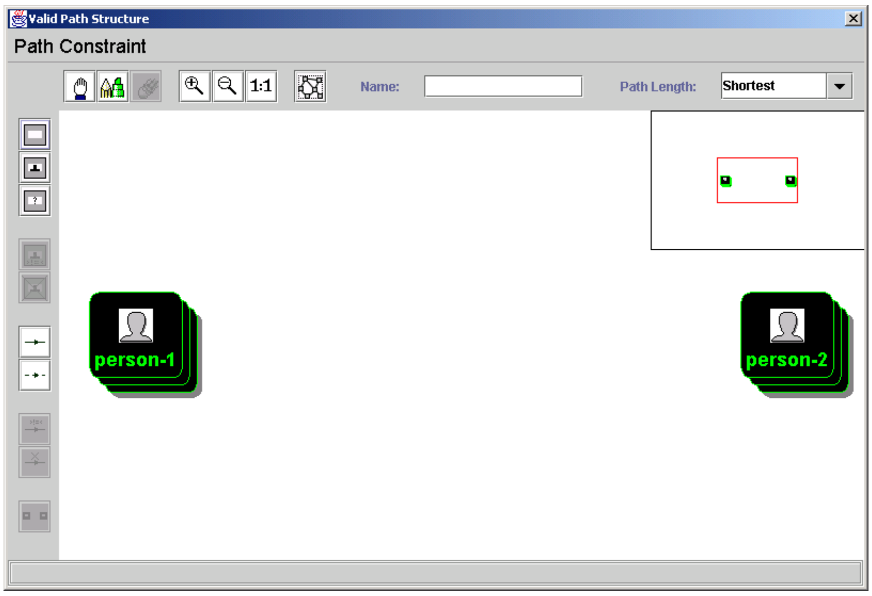


Figure 6.11.1. A window for specifying a path filter

The type of paths that are matched by a path filter is indicated by the structure of the paths that connect the start and finish states. As shown in figure 6.11.2, a single path filter may match several different types of path if there are two or more paths that connect the start and finish states.

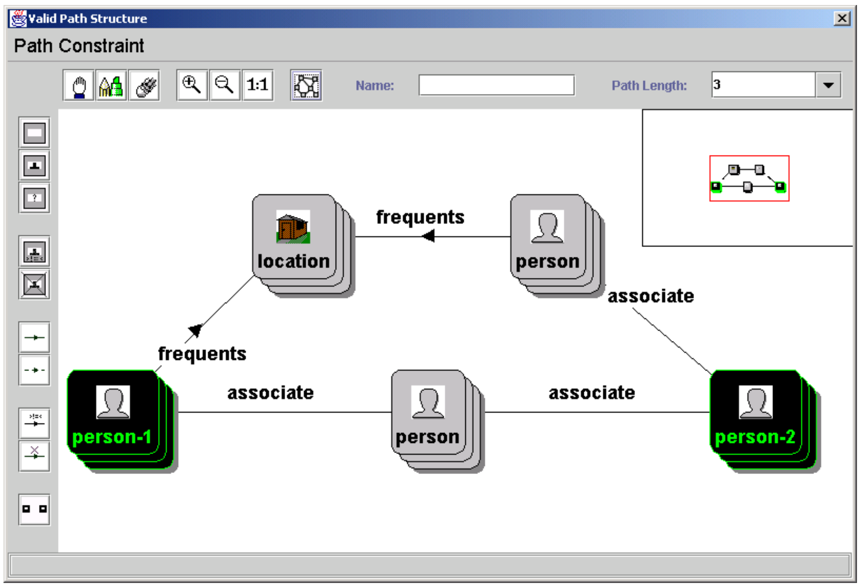


Figure 6.11.2. A path filter that matches paths that connect the objects matched by the object constraints person-1 and person-2 that either represent they have a mutual associate, or that the first person frequents a location also frequented by an associate of the second

As shown in figure 6.11.3, when a path filter contains a loop the types of objects and links that are represented by the constraints within the loop may occur several consecutive times in a path matched by the path filter.

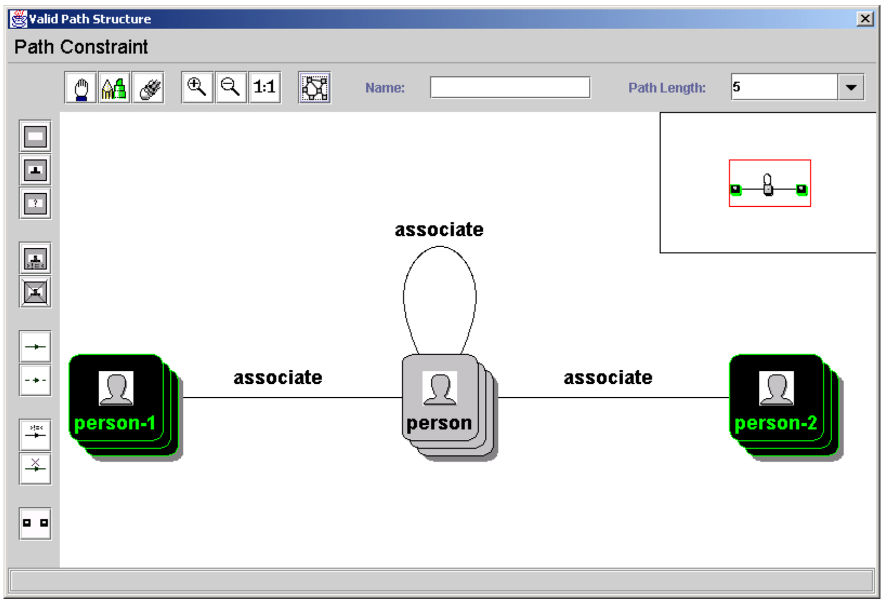


Figure 6.11.3. A path filter that matches the paths connecting the objects matched by the object constraints person-1 and person-2 that consist only of links that represent that two people are associates, and where all the objects in the paths represent people

Path filters that match paths that contain several consecutive links of the same type can be specified more concisely than in figure 6.11.3, by using an indirect link constraint. An indirect link constraint matches one or more consecutive links in a path that meet the conditions specified for the link constraint. As shown in figure 6.11.4, an indirect link constraint appears in a path filter with a dashed connecting arc, and is added to path filter in the same manner as described in section 6.4 for link constraints in a filter pattern except that the -•- option should be selected from the side toolbar.

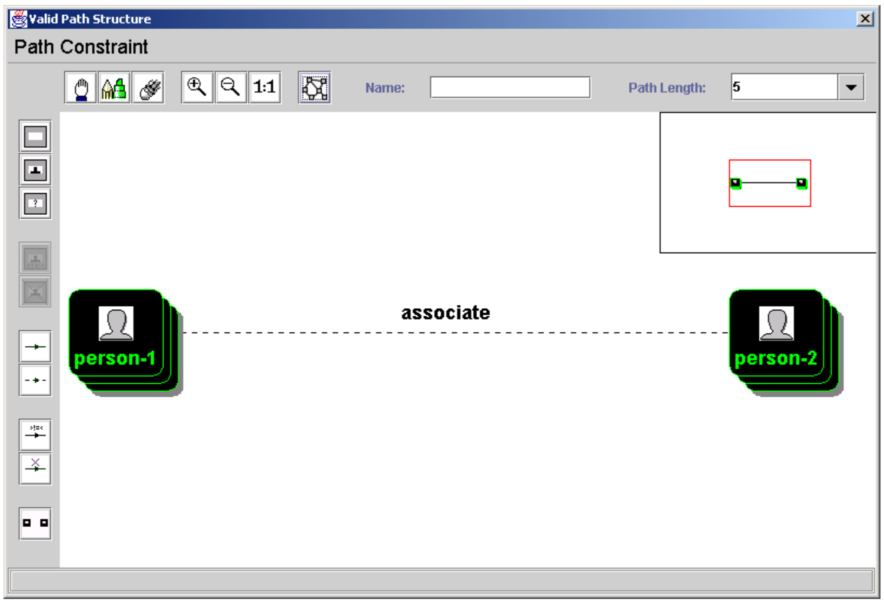
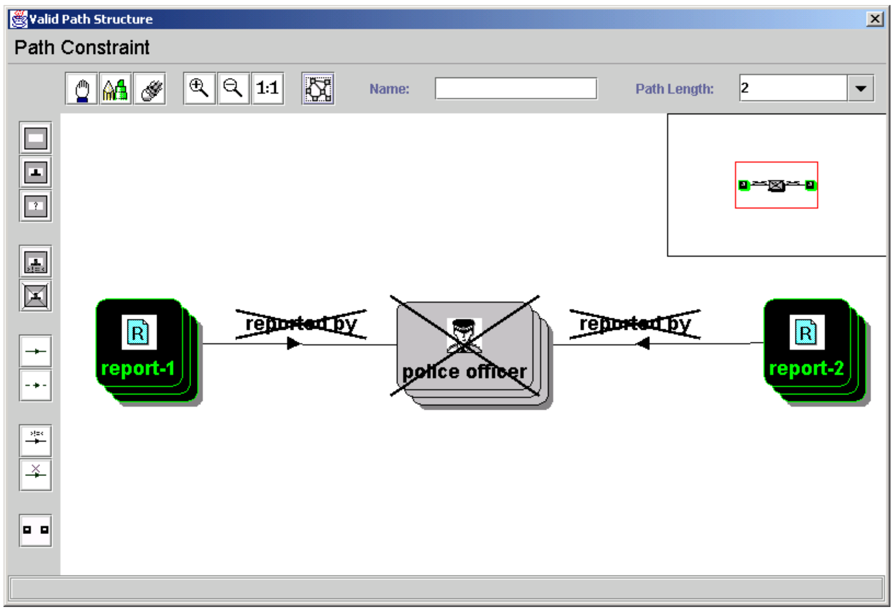


Figure 6.11.4. A path filter containing an indirect link constraint that matches the same paths as the path filter in figure 6.11.3

With the exception of the start and finish states, any object or link constraint in a path filter that only matches a specified type of object or link may be inverted by selecting either the or the option from the side toolbar and then clicking on the constraint. When a constraint is inverted only objects or links that do not satisfy the conditions specified are matched by the constraint. As shown in figure 6.11.5, if an object or link constraint is inverted it appears crossed out.



Name:

Path Length: 2

Figure 6.11.5. A path filter that matches paths connecting the objects matched by the object constraints report–1 and report–2 that do not represent that the reports are reported by the same police officer

The maximum number of links that may be contained in a path that is matched by a path filter is indicated by the value selected in the list located in the top right hand corner of the window. The value selected in this list may be edited, as may the name associated with a path filter. The name associated with a path filter is specified in the text field adjacent to the list and is used to label the associated path constraint in the filter pattern.

Providing a valid path filter has been constructed (there should be at least one path connecting the start and finish states) the path constraint associated with a path filter may be added to the filter pattern by selecting **add** from the **path constraint** menu.

A.3 Menu/Toolbar Options in the Explorer Interface

Side Toolbar

Add one or more objects selected from a list of those stored to the link chart

Incrementally explore the objects and links connected to a selected object

Display the information stored directly about an object or a link in a separate dialog

Display the paths that connect two objects

Display the objects, and the connecting paths, that are connected to an object

Display in a separate dialog those objects deemed similar to an object according to given set of conditions, and if desired add them to the link chart

Display in a separate dialog the objects and values linked to two objects by the same type of link, and if desired add them to the link chart

Data Request Menu

Select by Link Count. Display in a separate dialog all objects linked to a number of other objects, and if desired add them to the link chart

Center. Display in a separate dialog those objects connected to two or more selected objects by paths of a given maximum length, and if desired add them to the link chart

A.4 The Description of the User Evaluation Database

The database used in the evaluation is different from that used in the tutorial, and stores information concerning a set of companies that are suspected of being fraudulent, and the people who are employed in and own the companies. As shown in figure 1, each company may employ or be owned by several people, and each person may be employed in or own several companies.

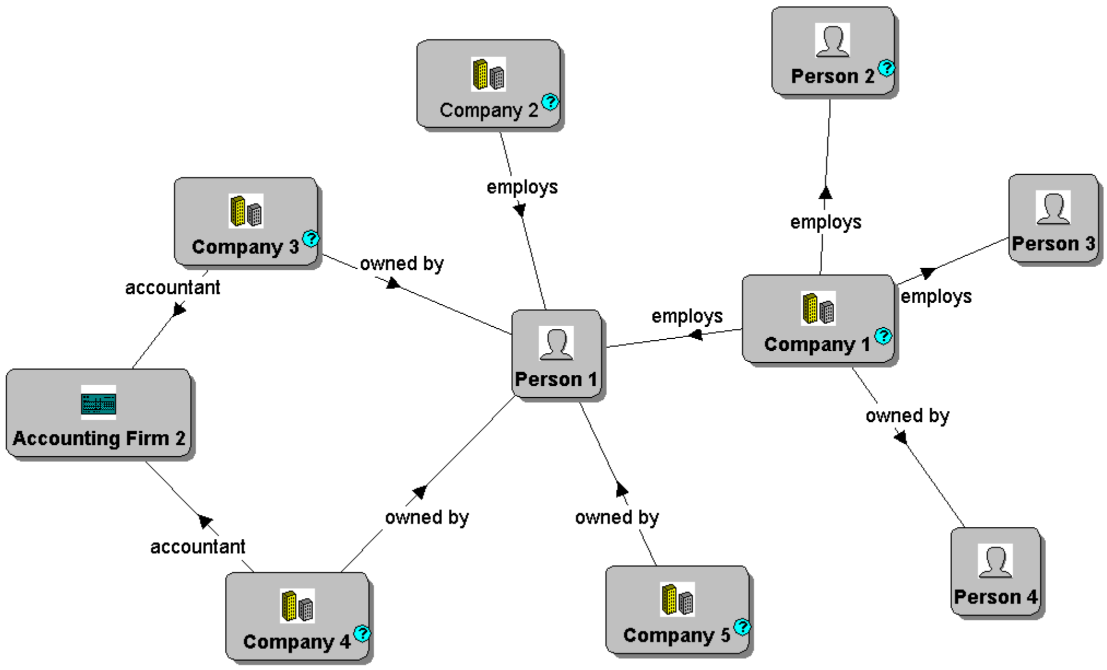


Figure 1. The structure of the data stored in the database used in the evaluation

The following information is stored about each person: name, age, nationality and occupation. The following information is stored about each company: name, registered address, date started, and the firm of accountants that process the accounts of the company.

A.5 The User Evaluation Questions

1 Expressing Queries using the Explorer Interface

1. Add to the chart *Person 1*.

2. Add to the chart the companies that *Person 1* is employed in, the companies that *Person-1* owns, and the connecting links.

3. Add to the chart the people employed in the same companies as *Person 1*, and the connecting links.

4. Add to the chart the people who own at least three companies.

5. Add to the chart the paths of length 4 or less that connect *Person 2* with *Person 7* that only consist of links that represent that a company employs a person or is owned by a person.

6. Add to the chart all people connected to *Person 1*, *Person 8* and *Person 2* by paths of length 3 or less that only consist of links that represent that a company employs a person or is owned by a person.

2 Query Comprehension

1. Highlight people employed in the same companies as *Person 1*.

2. Highlight all objects, and the connecting paths, that are connected to *Company 3* by a path(s) of length 3 or less that only consist of links that represent that a company employs a person.

3. Highlight all companies that employ 3 or more people.

4. Highlight all paths of length 4 or less that connect *Person 2* and *Person 8*.

5. Highlight the shortest path connecting *Person 9* and *Person 12* that only consist of links that represent that a company employs a person.

6. Highlight the objects, and the connecting paths, connected to *Person 1*, *Person 3* and *Person 5* by paths of length 3 or less that only consist of links that represent that a company employs a person.

3 *Expressing Queries as a Filter Pattern*

1. Construct a filter pattern that would display all companies stored in the database.

2. Construct a filter pattern that would display all people who are female and are at least thirty-five, or are male and their occupation is author.

3. Construct a filter pattern that would display all people that own companies, the companies that they own, and the connecting links.

4. Construct a filter pattern that would display all people aged 35 and over who are employed in a company owned or part owned by *Person 1*.

5. Construct a path filter that would match all paths, regardless of length, that connect the people matched by the selected object constraints that only consist of links that represent that a company employs a person.

6. Construct a path filter that would match all paths of length two or less that connect the people matched by the selected object constraints that represent that they are employed in the same company, or that one person is employed in a company owned by the other person.

4 *Query Comprehension*

1. Highlight all people stored in the database.

2. Highlight the people whose age is greater than 40 and less than 45, or are male and British.

3. Highlight the people, and the associated companies and links, employed in at least two companies.

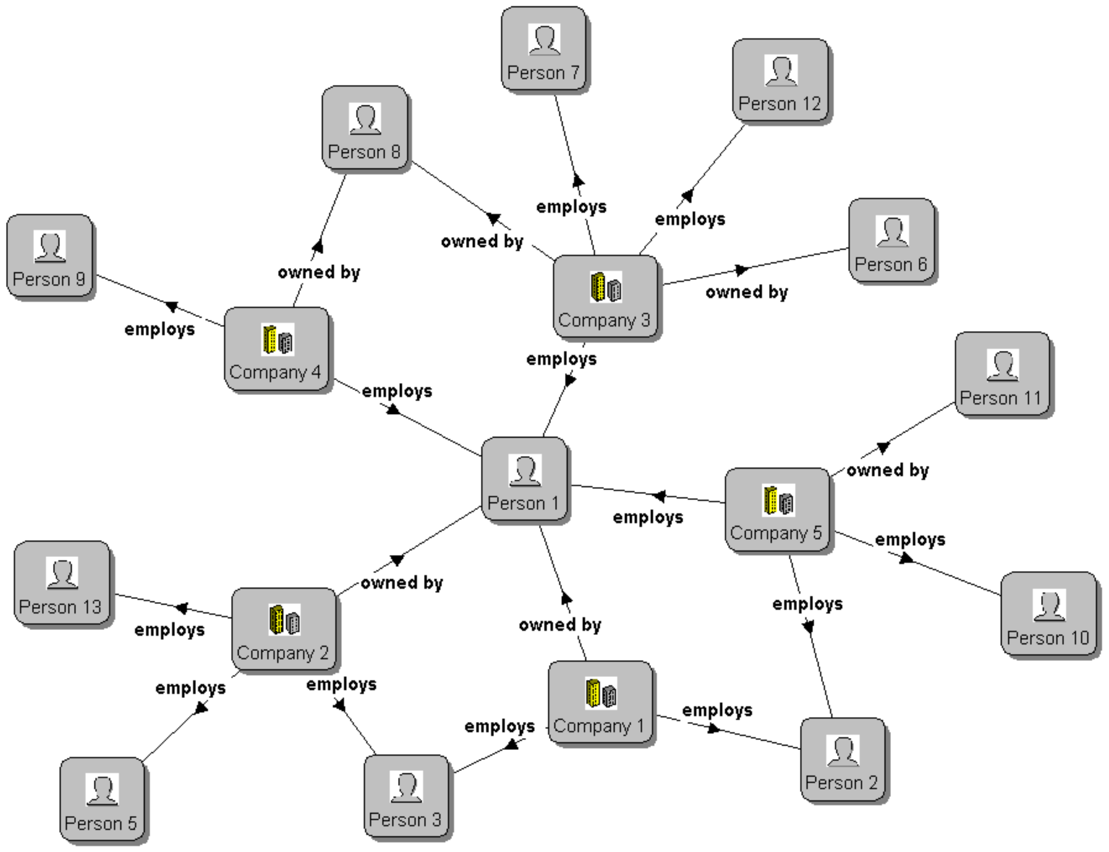
4. Highlight the people aged less than 43 who are employed in the same company as *Person 2*.

5. Describe in English what the paths matched by the path filter given on the accompanying sheet would represent.

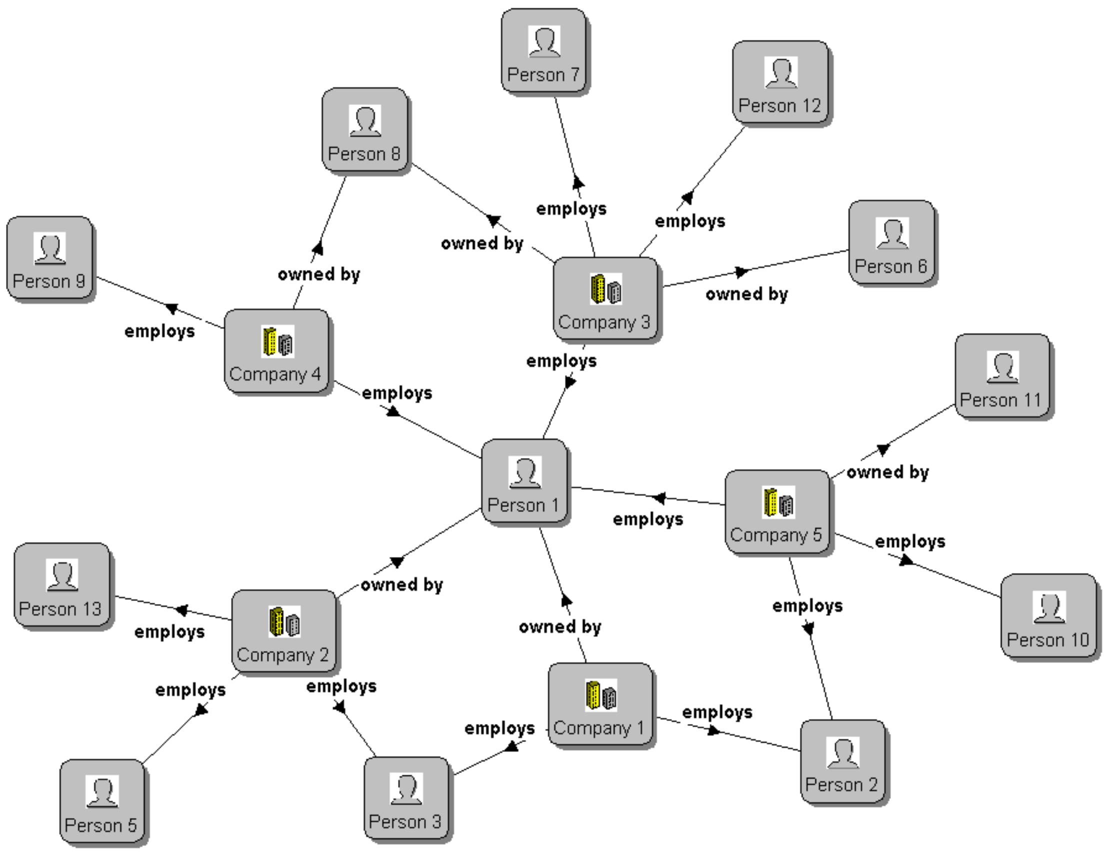
6. Describe in English what the paths matched by the path filter given on the accompanying sheet would represent.

A.6 The Query Comprehension Sheets

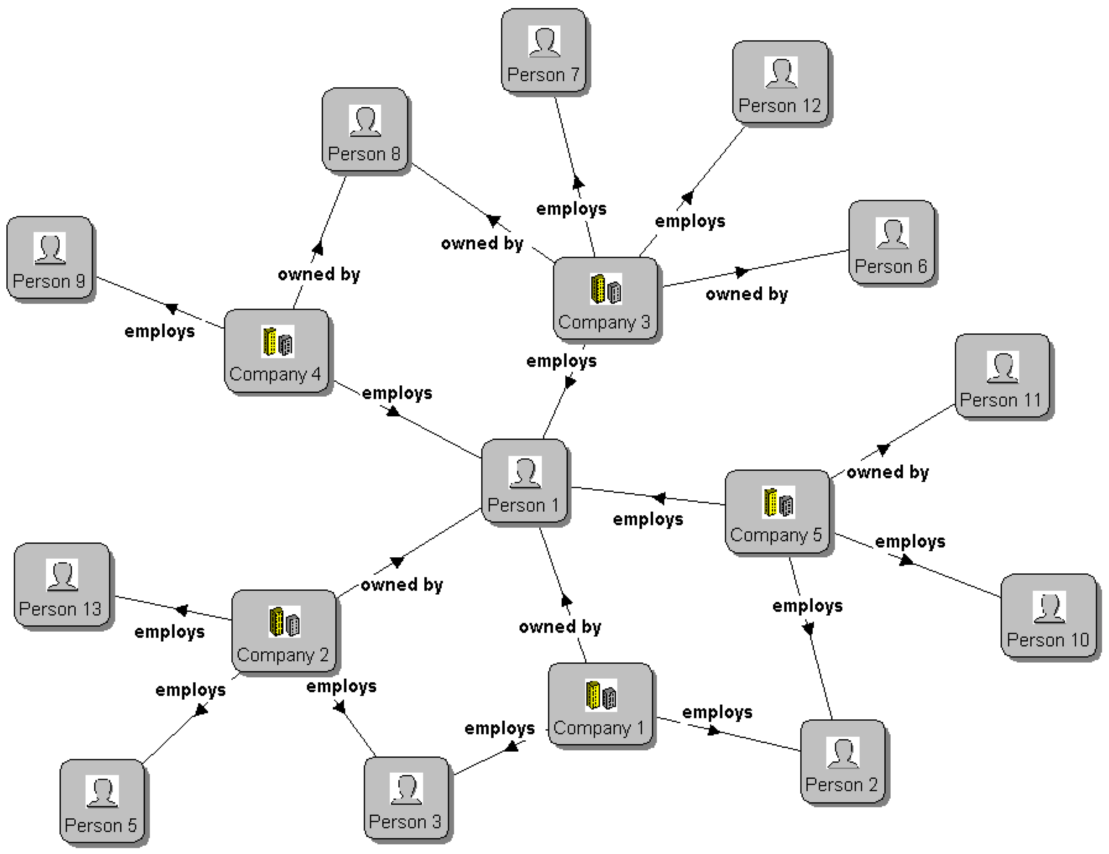
The Explorer Interface – Question 1



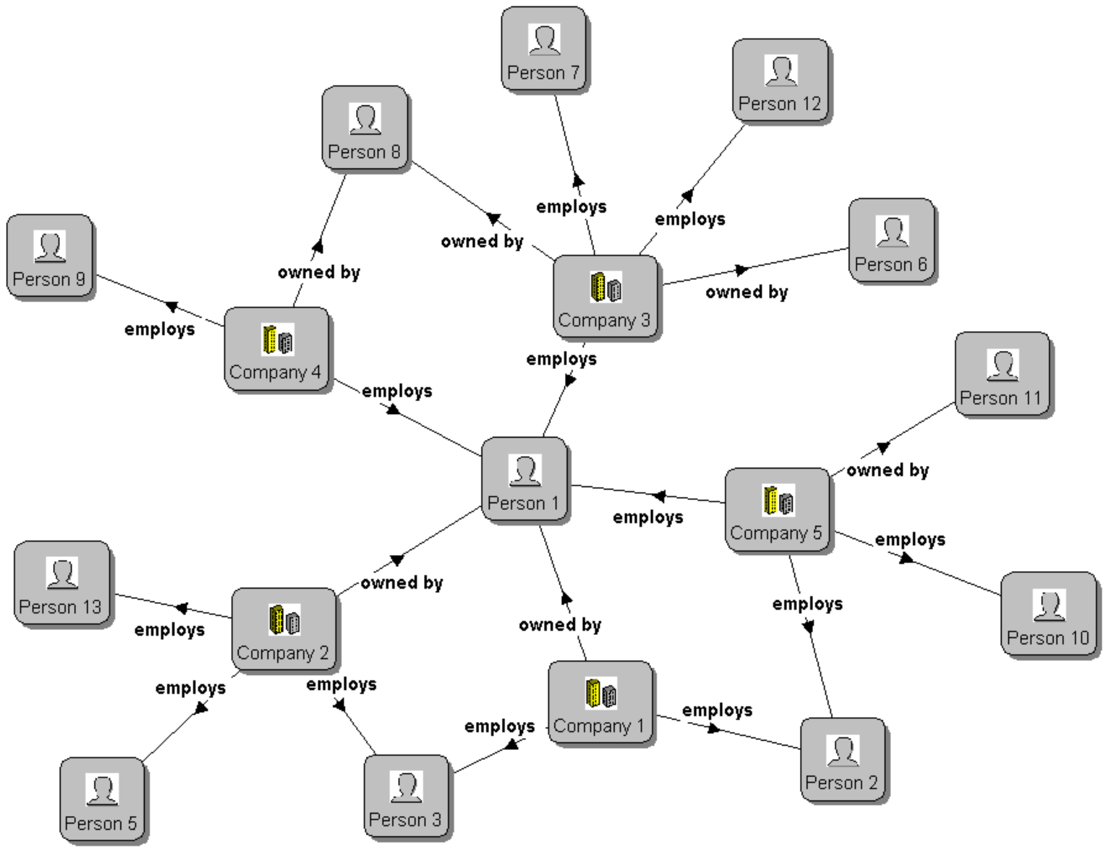
The Explorer Interface – Question 2



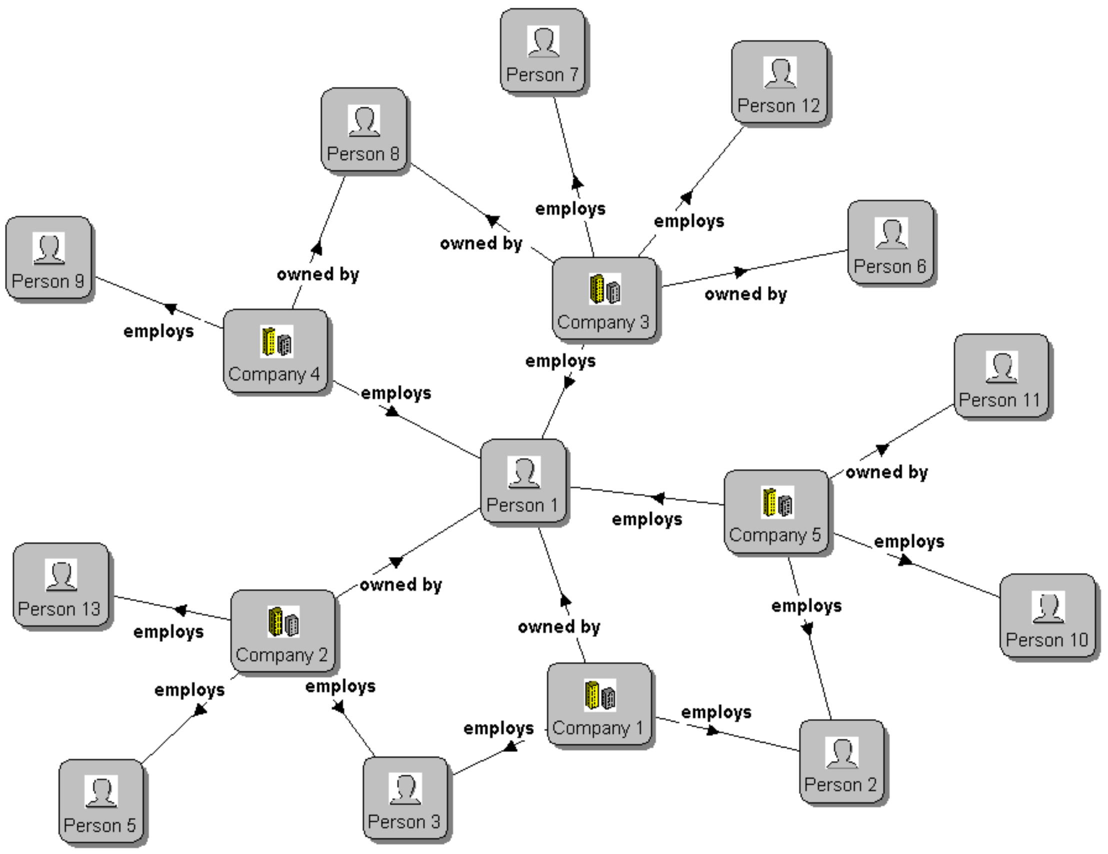
The Explorer Interface – Question 3



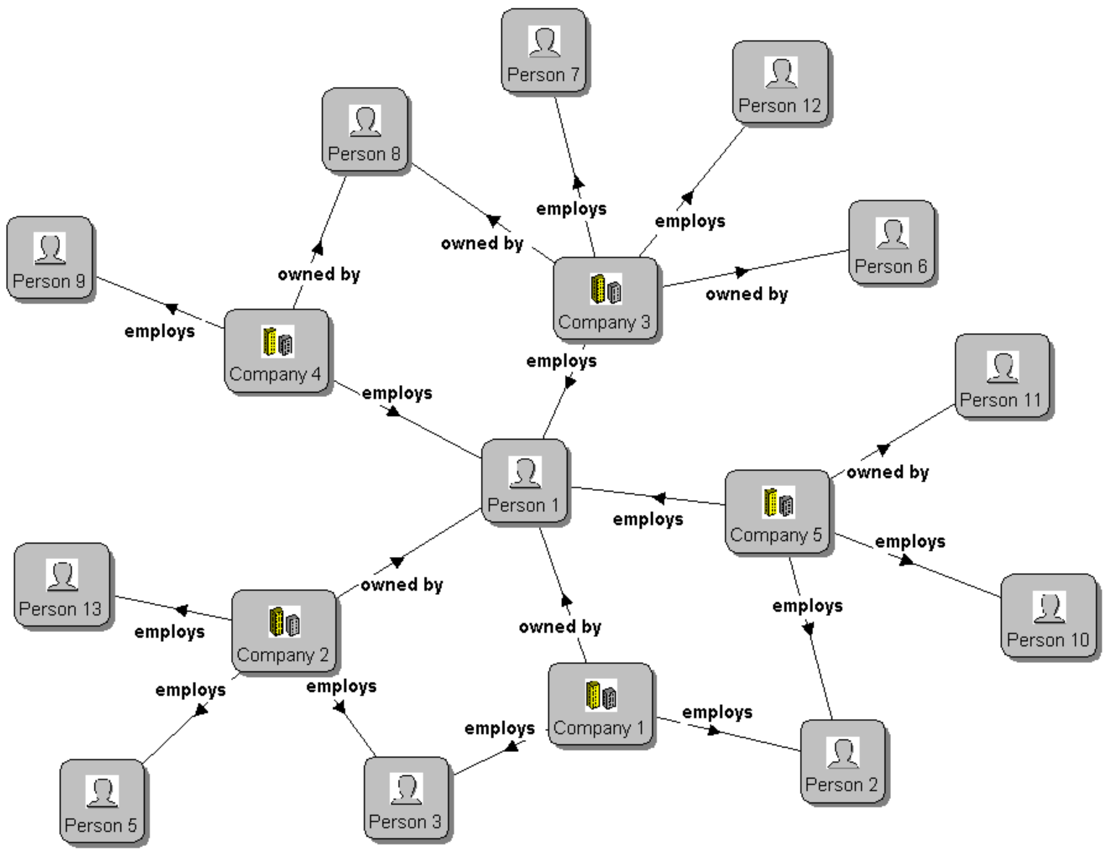
The Explorer Interface – Question 4



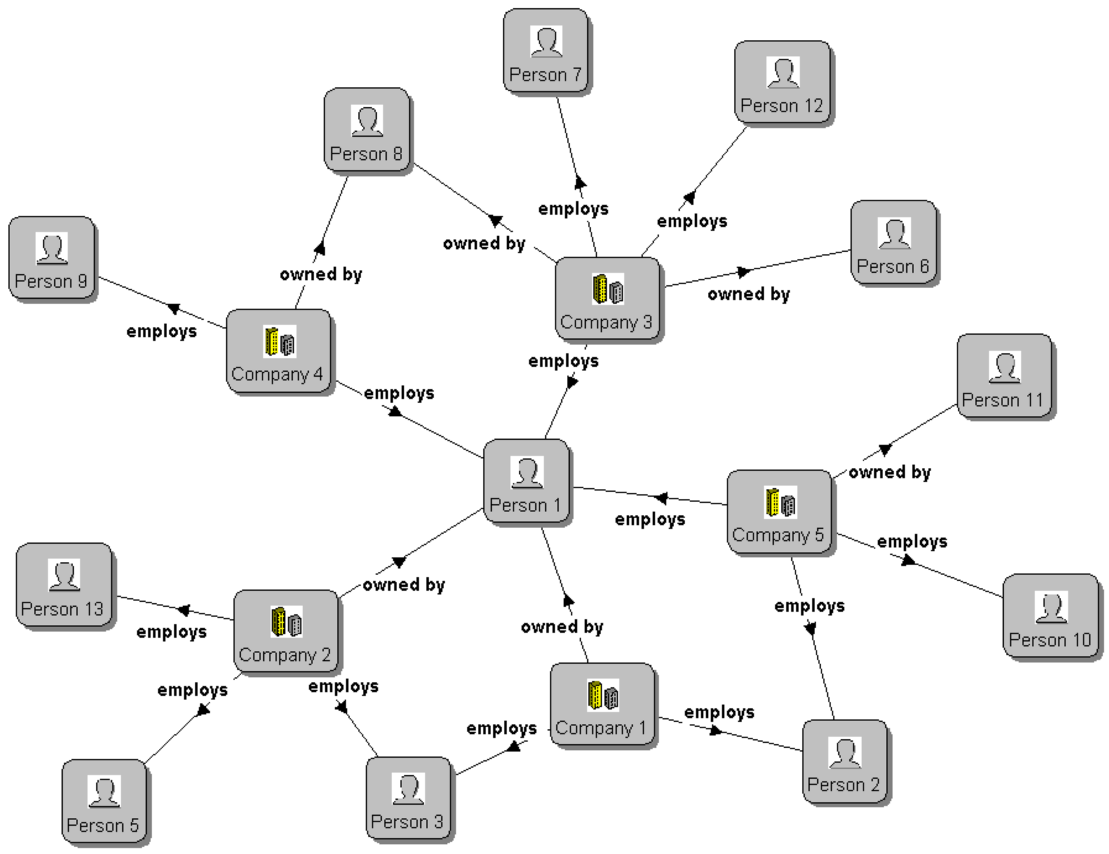
The Explorer Interface – Question 5



The Explorer Interface – Question 6



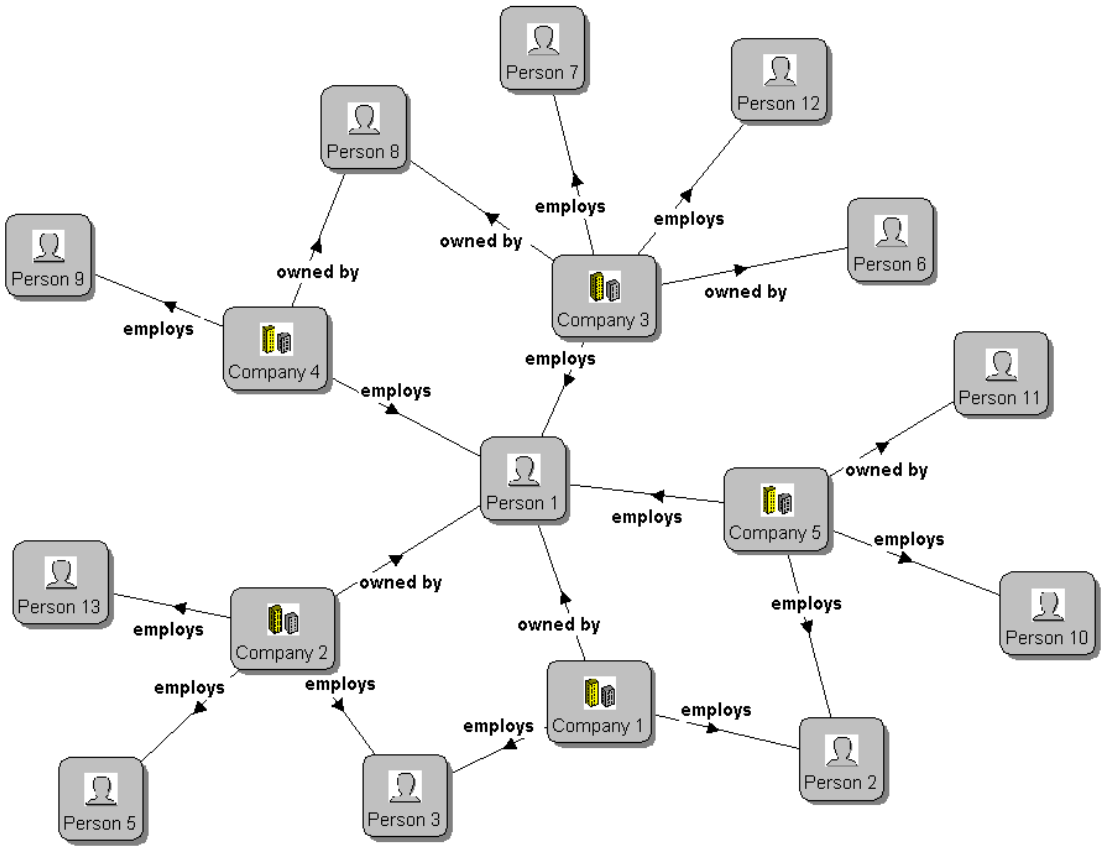
Filter Patterns – Question 1



Information Stored Directly about People

Identifier	Name	Age	Sex	Nationality	Profession
P1	Jane Smith	42	female	British	accountant
P2	David Gooldberg	33	male	American	ghost writer
P3	Sanjay	24	male	Indian	accountant
P5	Mathew Davies	36	male	British	author
P6	David Clennet	40	male	Canadian	businessman
P7	Harry Clean	52	male	American	publisher
P8	Helen Kite	27	female	British	businessman
P9	James Berry	32	male	British	publisher
P10	Peter Williams	44	male	British	author
P11	Kelly Townsend	52	female	British	business woman
P12	Michael Rees	38	male	American	author
P13	Heather Smith	23	female	British	accountant

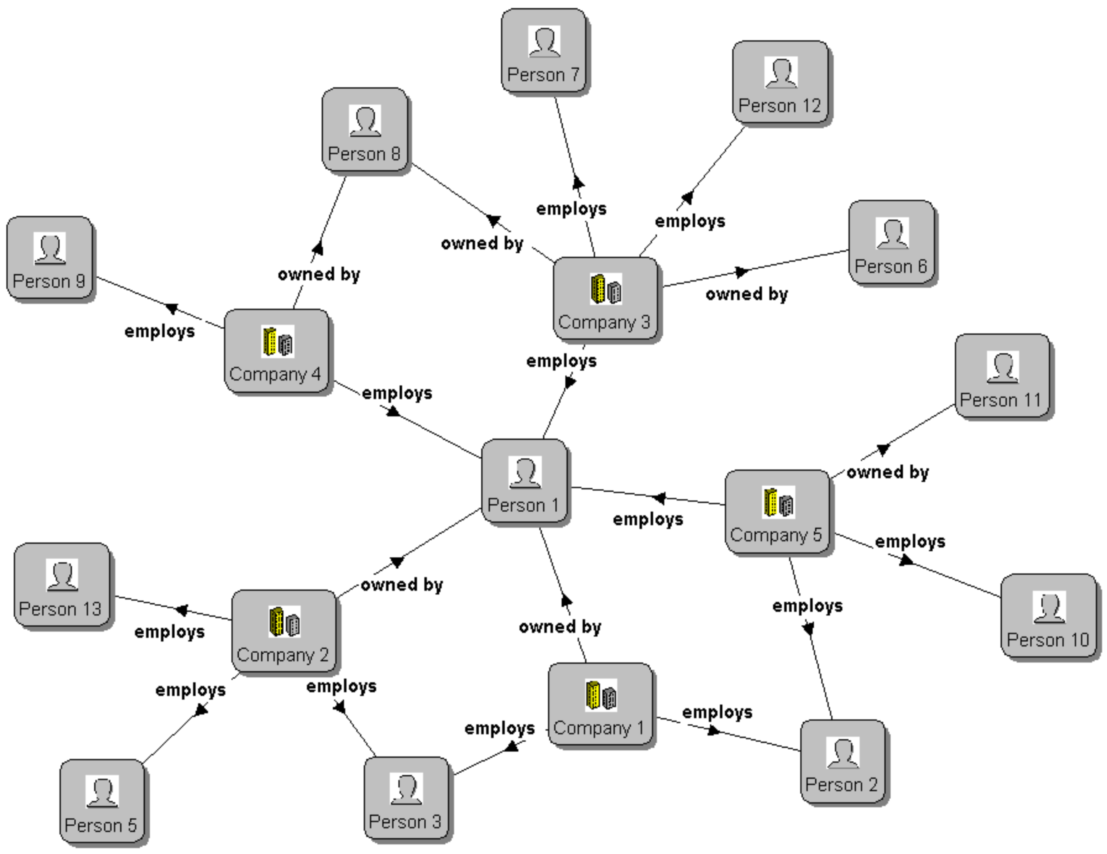
Filter Patterns – Question 2



Information Stored Directly about People

Identifier	Name	Age	Sex	Nationality	Profession
P1	Jane Smith	42	female	British	accountant
P2	David Gooldberg	33	male	American	ghost writer
P3	Sanjay	24	male	Indian	accountant
P5	Mathew Davies	36	male	British	author
P6	David Clennet	40	male	Canadian	businessman
P7	Harry Clean	52	male	American	publisher
P8	Helen Kite	27	female	British	businessman
P9	James Berry	32	male	British	publisher
P10	Peter Williams	44	male	British	author
P11	Kelly Townsend	52	female	British	business woman
P12	Michael Rees	38	male	American	author
P13	Heather Smith	23	female	British	accountant

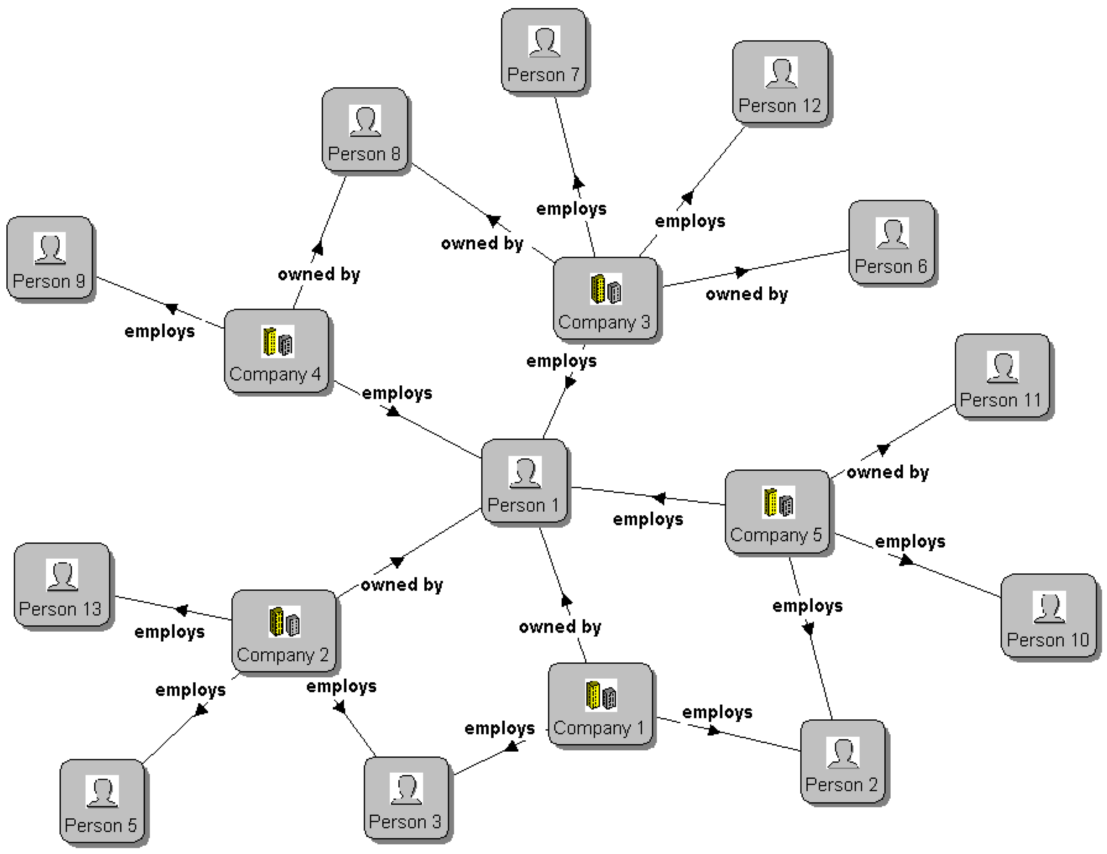
Filter Patterns – Question 3



Information Stored Directly about People

Identifier	Name	Age	Sex	Nationality	Profession
P1	Jane Smith	42	female	British	accountant
P2	David Gooldberg	33	male	American	ghost writer
P3	Sanjay	24	male	Indian	accountant
P5	Mathew Davies	36	male	British	author
P6	David Clennet	40	male	Canadian	businessman
P7	Harry Clean	52	male	American	publisher
P8	Helen Kite	27	female	British	businessman
P9	James Berry	32	male	British	publisher
P10	Peter Williams	44	male	British	author
P11	Kelly Townsend	52	female	British	business woman
P12	Michael Rees	38	male	American	author
P13	Heather Smith	23	female	British	accountant

Filter Patterns – Question 4

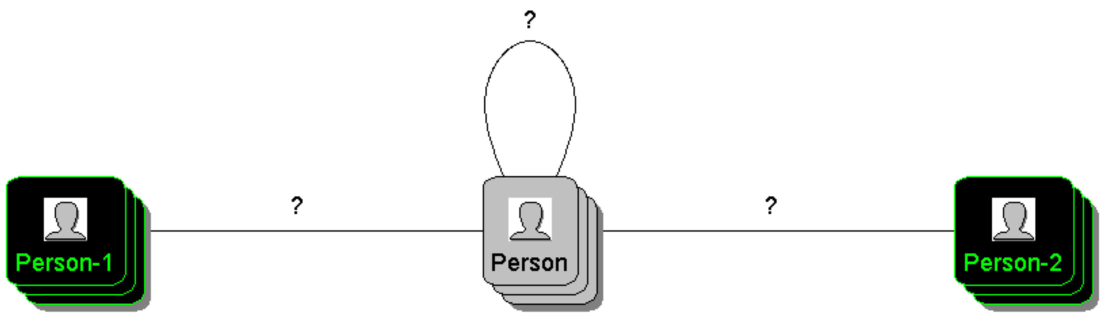


Information Stored Directly about People

Identifier	Name	Age	Sex	Nationality	Profession
P1	Jane Smith	42	female	British	accountant
P2	David Goolldberg	33	male	American	ghost writer
P3	Sanjay	24	male	Indian	accountant
P5	Mathew Davies	36	male	British	author
P6	David Clennet	40	male	Canadian	businessman
P7	Harry Clean	52	male	American	publisher
P8	Helen Kite	27	female	British	businessman
P9	James Berry	32	male	British	publisher
P10	Peter Williams	44	male	British	author
P11	Kelly Townsend	52	female	British	business woman
P12	Michael Rees	38	male	American	author
P13	Heather Smith	23	female	British	accountant

Filter Patterns – Question 5

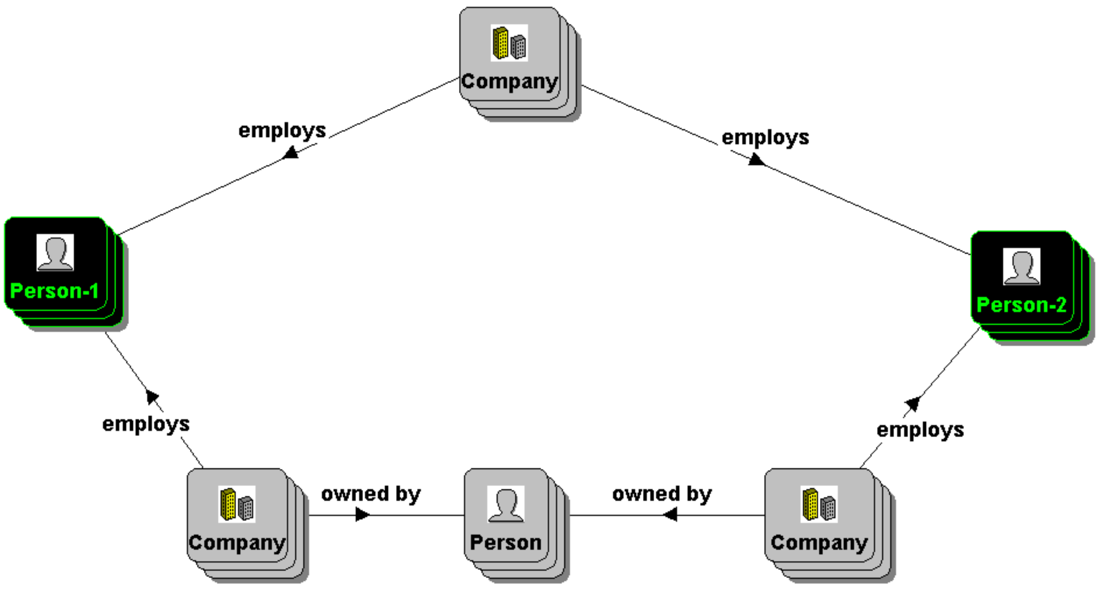
The Path Filter



Description

Filter Patterns – Question 6

Path Filter



Description

A.7 The Post-Evaluation Questionnaire

Did you find the concept of the grids for combining filter conditions difficult to understand?

Did you find it difficult to combine filter conditions using these grids?

Did you find the concept of a filter pattern difficult to understand?

Did you find the concept of a path filter difficult to understand?

Did you find it difficult to construct path filters? If so, were there any types of path that it was particularly difficult to construct a path filter for?

Do you have any general comments regarding the system, such as the style of interaction provided or difficulty of use?

Appendix B – Similarity Conditions

B.1 Introduction

The syntax of a valid similarity condition is dependant on whether the link type selected for the grid column associated with a given condition is defined as being single-valued or multivalued.

B.2 When the Selected Link Type is Multivalued

When the link type selected in the grid column associated is multivalued the syntax of a valid similarity condition is that given below in Extended Backus Naur Form (EBNF) for the element `<condition>`:

```
<condition>    ::= <comparison> | <in common> | <length>

<comparison>   ::= <operator> [<bag>]
<operator>     ::= “includes “ | “does not include “ | “equals ” |
                   “does not equal “ | “any in common “ | “distinct “
<bag>          ::= <data value> {“, “ <data value>}
<data value>   ::= <object> | <value>

<in common>    ::= [<conditional> “ “] <integer> “ in common”
<conditional>  ::= “< ” | “<= ” | “> ” | “>= ”

<length>       ::= (<conditional> | <equality>) <integer>
<equality>     ::= “= “ | “not = “

<object>       ::= “’” {<digit> | <letter> | <white space>} ”’”
<value>        ::= <text> | <number> | <truth> | <date> | <time> | <timestamp>
<text>         ::= “”” {<ASCII char>} “””
<number>       ::= <digit>{<digit>}[“.” <digit> {<digit>}]
<truth>        ::= “true” | “false”
<date>         ::= dependent on the Java runtime system used
<time>         ::= dependent on the Java runtime system used
<timestamp>    ::= dependent on the Java runtime system used
<ASCII char>   ::= any printable character in the ASCII character set
<white space>  ::= a character representing white space
<letter>       ::= “a”..“z” | “A”..”Z”
<integer>      ::= <digit>{<digit>}
<digit>        ::= “0”..”9”
```

The definition of the element `<condition>` is comprised of three distinct sub-elements, each of which represents a different type of condition. These are discussed below.

`<comparison>` represents a comparison between two bags (sets that may have duplicate elements) of objects or values using a specified operator. The first bag, DOV, represents those objects or values linked to an object that may be displayed by instances of the selected link type defined in the appropriate direction. The second bag, COV, represents those objects or values specified in the condition, or if no such bag was specified, those linked to the object selected by the user by instances of the selected link type defined in the appropriate direction.

The operators for comparing these bags have the following semantics: *includes* requires that DOV $\subseteq$ COV, *equals* that (DOV = COV), *any in common* that (DOV $\cap$ COV) $\neq$ {}, and the operator *distinct* that (DOV $\cap$ COV) = {}. The operators *does not include* and *does not equal* represent the negation of the operators *includes* and *equals* respectively.

The elements `<in common>` and `<length>` represent comparisons between an integer N, and the integer specified in the condition. For the element `<in common>` N represents the cardinality of (DOV $\cap$ COV). For the element `<length>` N represents the cardinality of DOV. Both elements require that the comparison operator specified in the condition should be satisfied.

B.3 When the Selected Link Type is Single-Valued

B.3.1 Introduction

When the link type selected in the grid column is single-valued the syntax of a valid similarity condition is dependant on the type of value or object that may be linked by instances of the selected link type defined in the appropriate direction. The valid syntaxes for the different types of values, and for objects, are defined in the subsections below.

B.3.2 Syntax for Text Values

If a text value may be linked to an object that may be displayed the syntax of a valid condition is that given in EBNF notation below for the element `<t condition>`, which like all elements defined in the following sections uses those elements defined previously in section B.2:

```
<t condition>  ::= <existence> | <sounds> | <contains> | <text equality>

<text equality>::= <equality> [<text>]
<sounds>       ::= (“sounds like ” | “does not sound like “) [<text>]
<contains>     ::= (“contains ” | “does not contain ”) [<text>]
<existence>    ::= “exists” | “does not exist”
```

The definition of the element `<t condition>` is comprised of four distinct sub-elements, each of which represents a different type of condition. These are discussed below.

`<existence>` represents an existence test. The operator *exists* tests if there is a value linked to an object that may be displayed, whereas *does not exist* tests if there is no such value. For either of the operators to return true the associated test must be satisfied. This element is also included in the syntax definitions provided in the sections below. As the element has an identical syntax and semantics in these definitions it will not be described again.

All other conditions compare two values, DOV and COV. DOV represents the value linked to an object that may be displayed by an instance of the selected link type defined in the appropriate direction. COV represents the value specified in the condition, or if no such value is specified the value linked to the object selected by the user by an instance of the selected link type defined in the appropriate direction. If no value is linked to an object that may be displayed, or to the object selected by the user, the respective values of DOV and COV are null.

`<text equality>` represents a comparison between the values of DOV and COV using a given operator. `<sounds>` represents a phonetic comparison between the values of DOV and COV using the SOUNDEX algorithm. The operator *sounds like* specifies that they must sound alike, whereas the operator *does not sound like* specifies that they must not sound alike.

`<contains>` represents a comparison between the values of DOV and COV to determine if COV is a substring of DOV. The operator *contains* specifies that COV must be a substring of DOV, whereas the operator *does not contain* specifies that COV must not be a substring.

With the exception of the operator *does not exist*, which is defined above, all operators when used to compare a null value evaluate to false.

B.3.3 Syntax for Number Values

If a number value may be linked to an object that may be displayed the syntax of a valid condition is that given in EBNF notation below for the element `<n condition>`:

```
<n condition>  ::= <num comp> | <within> | <existence>
<num comp>     ::= (<conditional> | <equality>) [<number>]
<within>       ::= (“within ” | “not within “) <number>
```

The definition of the element `<n condition>` is comprised of three distinct sub-elements, each of which represents a different type of condition. `<num comp>` represents a comparison between the values of DOV and COV using a specified operator. `<within>` represents that DOV must be, or must not be, within N of COV, where N is the number specified in the condition. This specifies that the following equation must, or must not, be satisfied: ((DOV >= (COV – N)) and (DOV <= (COV + N))). The operator *within* specifies that the equation must be satisfied, whereas the operator *not within* specifies that the equation must not be satisfied.

B.3.4 Syntax for Time, Date &Timestamp Values

The syntax for valid similarity conditions if a time, date or timestamp value may be linked to an object that may be displayed is identical to that described in section B.3.3 for number values, apart from the type of value that may be included in the condition. The semantics of the operators is also the same, except that when the *within* and *not within* operators are used the integer argument specified in the condition represents the number of seconds.

B.3.5 Syntax for Truth Values

If a truth value may be linked to an object that may be displayed the syntax of a valid condition is that given in EBNF notation below for the element `<tr condition>`:

```
<tr condition> ::= <truth comp> | <existence>
<truth comp>   ::= <equality> [<truth>]
```

The definition of the element `<tr condition>` is comprised of two distinct sub-elements, each of which represents a different type of condition. The syntax and semantics of these elements are identical to those defined in section B.3.3 for number values.

B.3.6 Syntax for Objects

If an object, regardless of type, may be linked to an object that may be displayed, the syntax of a valid condition is that given in EBNF notation below for the element `<o condition>`:

```
<o condition>  ::= <object comp> | <existence>
<object comp>  ::= <equality> [<object>]
```

With the exception of the type of data value that may be specified in the condition the syntax and semantics of the conditions that may be formed is identical to that given in section B.3.5 for truth values.

Appendix C – Filter Conditions for Objects

C.1 Introduction

The syntax of a valid object constraint filter condition is dependant on whether the link type selected for the grid column associated with a given condition is defined as being single-valued or multivalued.

C.2 When the Selected Link Type is Multivalued

When the link type selected in the grid column is multivalued, the syntax of a valid object constraint filter condition is that given below in Extended Backus Naur Form (EBNF) for the element `<condition>`:

```
<condition>    ::= <comparison> | <length>

<comparison>   ::= <operator> <bag>
<operator>     ::= “includes “ | “does not include “ | “equals ” |
                   “does not equal “ | “any in common “ | “distinct “
<bag>          ::= <data value> {“, “ <data value>}
<data value>   ::= <object> | <value> | <reference>

<reference>    ::= <simple> | <cast>
<simple>       ::= <label> ”.”  <link type>
<cast>         ::= “((“ <object type> “)” <label> “).” <link type>
<label>        ::= label of an object constraint defined in the filter pattern
<object type>  ::= the name of an object type defined in the database schema
<link type>    ::= the name of a link type defined in the database schema

<length>       ::= (<conditional> | <equality>) <integer>
<conditional>  ::= “< ” | “<= ” | “> ” | “>= ”
<equality>     ::= “= “ | “not = “

<object>       ::= “’” {<digit> | <letter> | <white space>} ”’”
<value>        ::= <text> | <number> | <truth> | <date> | <time> | <timestamp>
<text>         ::= “”” {<ASCII char>} “””
<number>       ::= <digit>{<digit>}[“.” <digit> {<digit>}]
<truth>        ::= “true” | “false”
<date>         ::= dependent on the Java runtime system used
<time>         ::= dependent on the Java runtime system used
<timestamp>    ::= dependent on the Java runtime system used
<ASCII char>   ::= any printable character in the ASCII character set
<white space>  ::= a character representing white space
<letter>       ::= “a”..“z” | “A”..”Z”
<integer>      ::= <digit>{<digit>}
<digit>        ::= “0”..”9”
```

The definition of the element `<condition>` is comprised of two distinct sub-elements, each of which represents a different type of condition. These are discussed below.

`<comparison>` represents a comparison between two bags (sets that may have duplicate elements) of objects or values using a specified operator. The first bag, MOV, represents those objects or values linked to an object that may be matched by instances of the selected link type defined in the appropriate direction. The second bag, COV, represents those objects or values specified in the condition.

The operators for comparing these bags have the following semantics: *includes* requires that MOV $\subseteq$ COV, *equals* that (MOV = COV), *any in common* that (MOV $\cap$ COV) $\neq$ {}, and the operator *distinct* that (MOV $\cap$ COV) = {}. The operators *does not include* and *does not equal* represent the negation of the operators *includes* and *equals* respectively.

The element `<length>` represents a comparison between an integer N and the integer specified in the condition, where N represents the cardinality of MOV. For the operator to return true the comparison operator specified in the condition should be satisfied.

C.3 When the Selected Link Type is Single-Valued

C.3.1 Introduction

When the link type selected in the grid column is single-valued the syntax of a valid object constraint filter condition is dependant on the type of value or object that may be linked by an instance of the selected link type defined in the appropriate direction. The valid syntaxes for the different types of values, and for objects, are defined in the sections below.

C.3.2 Syntax for Text Values

If a text value may be linked to an object that may be matched the syntax of a valid condition is that given in EBNF notation below for the element `<t condition>`, which like all elements defined in the following subsections uses those elements defined previously in section C.2:

```
<t condition>  ::= <existence> | <sounds> | <contains> | <text equality>

<text equality>::= <equality> <t reference>
<sounds>       ::= (“sounds like ” | “does not sound like “) <t reference>
<contains>     ::= (“contains ” | “does not contain ”) <t reference>
<existence>    ::= “exists” | “does not exist”
<t reference>  ::= <text> | <reference>
```

The definition of the element `<t condition>` is comprised of four distinct sub-elements, each of which represents a different type of condition. These are discussed below.

`<existence>` represents an existence test. The operator *exists* tests if there is a value linked to an object that may be matched, whereas *does not exist* tests if there is no such value. For either of the operators to return true the associated test must be satisfied. This element is also included in the syntax definitions provided in the sections below. As the element has an identical syntax and semantics in these definitions it will not be described again.

All other conditions compare two values, MOV and COV. MOV represents the value linked to an object that may be matched by an instance of the selected link type defined in the appropriate direction. COV represents the value specified in the condition. If no value is linked to an object that may be matched the value of MOV is null.

`<text equality>` represents a comparison between the values of MOV and COV using a given operator. `<sounds>` represents a phonetic comparison between the values of MOV and COV using the SOUNDEX algorithm. The operator *sounds like* specifies that they must sound alike, whereas the operator *does not sound like* specifies that they must not sound alike.

`<contains>` represents a comparison between the values of MOV and COV to determine if COV is a substring of MOV. The operator *contains* specifies that COV must be a substring of MOV, whereas the operator *does not contain* specifies that COV must not be a substring.

With the exception of the operator *does not exist*, which is defined above, all operators when used to compare a null value evaluate to false.

C.3.3 Syntax for Number Values

If a number value may be linked to an object that may be matched the syntax of a valid condition is that given in EBNF notation below for the element `<n condition>`:

```
<n condition>  ::= <num comp> | <existence>
<num comp>     ::= (<conditional> | <equality>) (<number> | <reference>)
```

The definition of the element `<n condition>` is comprised of two distinct sub-elements, each of which represents a different type of condition. `<existence>` is defined in section C.3.2. `<num comp>` represents a comparison between the values of MOV and COV using a specified operator.

C.3.4 Syntax for Time, Date &Timestamp Values

The syntax for valid conditions if a time, date or timestamp value may be linked to an object that may be matched is identical to that described in section C.3.3 for number values apart from the type of value that may be included in the condition. The semantics of the operators is also the same.

C.3.5 Syntax for Truth Values

If a truth value may be linked to an object that may be matched then the syntax of a valid condition is that given in EBNF notation below for the element `<tr condition>`:

```
<tr condition> ::= <truth comp> | <existence>
<truth comp>   ::= <equality> (<truth> | <reference>)
```

The definition of the element <tr condition> is comprised of two distinct sub-elements, each of which represents a different type of condition. The syntax and semantics of these elements are identical to those defined in section C.3.3 for number values.

C.3.6 Syntax for Objects

If an object, regardless of type, may be linked to an object that may be matched the syntax of a valid condition is that given in EBNF notation below for the element `<o condition>`:

```
<o condition>  ::= <object comp> | <existence>
<object comp>  ::= <equality> (<object> | <reference>)
```

With the exception of the type of data value that may be specified in the condition the syntax and semantics of the conditions that may be formed is identical to that given in section C.3.5 for truth values.

Appendix D - Filter Conditions for Links

D.1 Introduction

The syntax of a valid link constraint filter condition is dependant on the value type of the attribute selected for the grid column associated with a given condition. The valid syntaxes for the value types defined in EDVC are provided in the sections below, which all use the following Extended Backus Naur Form (EBNF) definitions:

```
<conditional>  ::= “< ” | “<= ” | “> ” | “>= ”
<equality>     ::= “= “ | “not = “
<text>         ::= “”” {<ASCII char>} “””
<number>       ::= <digit>{<digit>}[“.” <digit> {<digit>}]
<truth>        ::= “true” | “false”
<date>         ::= dependent on the Java runtime system used
<time>         ::= dependent on the Java runtime system used
<timestamp>    ::= dependent on the Java runtime system used
<ASCII char>   ::= any printable character in the ASCII character set
<white space>  ::= a character representing white space
<letter>       ::= “a”..“z” | “A”..”Z”
<integer>      ::= <digit>{<digit>}
<digit>        ::= “0”..”9”
```

D.2 Syntax for the Text Value Type

If the value type of the attribute selected in the grid column is the text value type then the syntax of a valid link constraint filter condition is that given in EBNF notation below for the element `<t condition>`:

```
<t condition>  ::= <existence> | <sounds> | <contains> | <text equality>

<text equality>::= <equality> <text>
<sounds>       ::= (“sounds like ” | “does not sound like “) <text>
<contains>     ::= (“contains ” | “does not contain ”) <text>
<existence>    ::= “exists” | “does not exist”
```

The definition of the element `<t condition>` is comprised of four distinct sub-elements, each of which represents a different type of condition. These are discussed below.

`<existence>` represents an existence test. The operator *exists* tests if there is a value defined for the attribute selected for the link currently being evaluated, whereas *does not exist* tests if there is no such attribute value. For either of the operators to return true the associated test must be satisfied. This element is also included in the syntax definitions provided in the sections below. As the element has an identical syntax and semantics in these definitions, it will not be described again.

All other conditions compare two values, LAV and COV. LAV represents the value of the attribute selected in the grid column for the link currently being evaluated. COV represents the value specified in the condition. If the link currently being evaluated does not have a value defined for that attribute the value of LAV is null.

`<text equality>` represents a comparison between the values of LAV and COV using a given operator. `<sounds>` represents a phonetic comparison between the values of LAV and COV using the SOUNDEX algorithm. The operator *sounds like* specifies that they must sound alike, whereas the operator *does not sound like* specifies that they must not sound alike.

`<contains>` represents a comparison between the values of LAV and COV to determine if COV is a substring of LAV. The operator *contains* specifies that COV must be a substring of LAV, whereas the operator *does not contain* specifies that COV must not be a substring.

With the exception of the operator *does not exist*, which is defined above, all operators when used to compare a null value evaluate to false.

D.3 Syntax for the Number Value Type

If the value type of the attribute selected in the grid column is the number value type then the syntax of a valid condition is that given in EBNF notation below for the element `<n condition>`:

```
<n condition>  ::= <num comp> | <existence>
<num comp>     ::= (<conditional> | <equality>) <number>
```

The definition of the element `<n condition>` is comprised of two distinct sub-elements, each of which represents a different type of condition. `<existence>` is defined in section D.2. `<num comp>` represents a comparison between the values of LAV and COV using a specified operator.

D.4 Syntax for the Time, Date & Timestamp Value Types

If the value type associated with the attribute selected in the grid column is either the time, date or timestamp value types then the syntax of a valid condition is, apart from the type of value that may be included in the condition, identical to that described in section D.3 for the number value type. The semantics of the operators is also the same.

D.5 Syntax for the Truth Value Type

If the value type of the attribute selected in the grid column is the truth value type then the syntax of a valid condition is that given in EBNF notation below for the element `<tr condition>`:

```
<tr condition> ::= <truth comp> | <existence>
<truth comp>   ::= <equality> <truth>
```

The definition of the element `<tr condition>` is comprised of two distinct sub-elements, each of which represents a different type of condition. The syntax and semantics of these elements are identical to those defined in section D.3 for the number value type.

Appendix E – A User Manual for DMT

A User Manual for the Database Manager Tool

Contents

1 Introduction

1.1 Introduction

This booklet is a user manual for the Database Manger Tool (DMT), an application for creating database files for the database query interface the Exploratory Database View Constructor (EDVC). DMT is written in version 1.3 of the Java programming language and will run on any platform that supports version 2 of the standard edition of the Java 2 runtime system.

It is assumed that the reader has a working knowledge of the operating system installed on their computer and understands how to use menus and a mouse. The term *click* is used throughout this booklet to refer to a single click of the leftmost mouse button. If a double-click or the rightmost mouse button is needed it will be explicitly stated.

1.2 Installing DMT

Copy the *edvc* directory from the installation disk to your computer. Set the CLASSPATH environment variable to point to the *edvc* directory (see the user documentation for your platform for instructions on how to perform this operation). Open the *edvc.pro* file located within the *edvc* directory with a text editor. At the end of the first line append the full pathname of the directory in which the *edvc* directory is located. For example, if the *edvc* directory was copied into the *C:\Program Files* directory on a Microsoft® Windows® platform the first line of the file should be:

```
installation directory=C:\Program Files
```

1.3 Starting DMT

If you are using a Microsoft® Windows® platform double-click on the **Database Manager Tool** icon on the desktop. If you are using a UNIX based platform type `java edvc.Dmt` at the command line and press enter.

1.4 Creating a New Database File

Select **new** from the **database** menu. If changes have been made to the database file currently open, if any, then a dialog asking if these changes should be saved will be displayed; after any changes have been saved, if needed, a dialog similar to the one in figure 1.4.1 will be displayed.

Enter the name of the database associated with the new database file in the text field in the **new database name** panel then click the **OK** button, the main panel will turn white and the pan window (described in section 4) will appear.

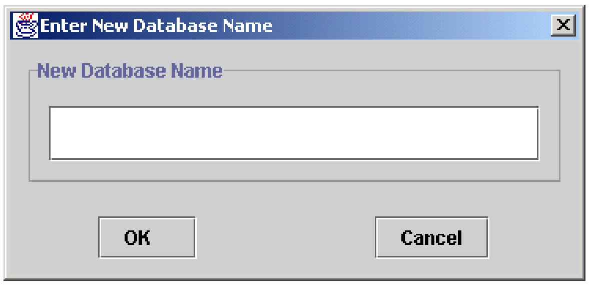
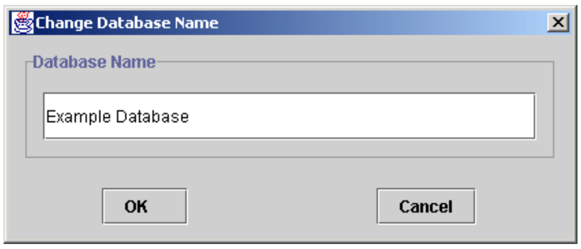


Figure 1.4.1. A dialog for entering the name of the database associated with a new database file

1.5 Changing the Name of a Database

1. Select **change name** from the **database** menu, a dialog similar to the one in figure 1.5.1 will be displayed



Example Database

Figure 1.5.1. The change database name dialog

2. Enter the new name in the text field in the **database name** panel

3. Click the **OK** button

2 Creating Schema Elements

2.1 Creating a New Object Type

1. Select the ▭ option from the toolbar[1]

2. Click on the main panel where the object type should initially be placed, a dialog similar to the one in figure 2.1.1 will be displayed

3. Enter the name of the object type in the text field located in the **name** panel

The name must not be the same as any other object type currently defined in the database schema, and must consist only of letters, digits and white space.

[1] A short description of what a toolbar option represents can be obtained by leaving the pointer stationary over the correpsonding icon for a short period of time.

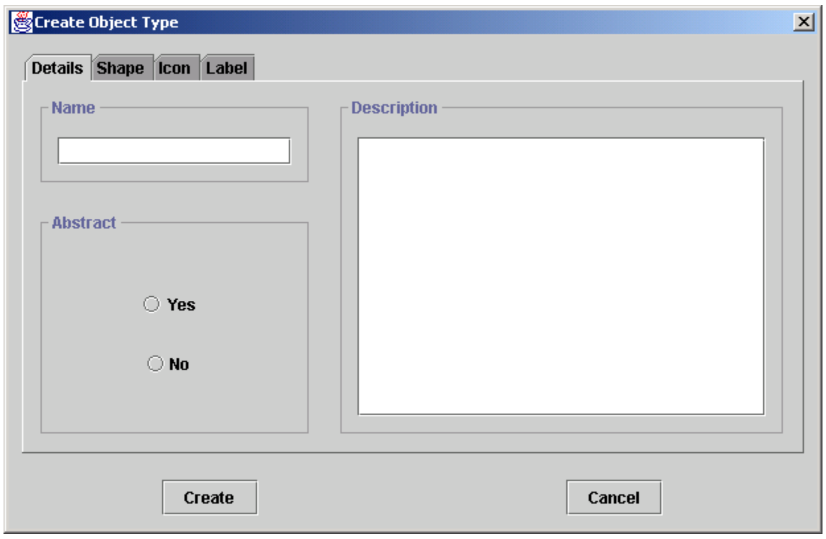


Figure 2.1.1. The create object type dialog

4. If the object type is abstract, select the **yes** button in the **abstract** panel, otherwise select the **no** button

All other information concerning the object type is optional. The object type may be given a description by entering one in the text field in the **description** panel. The visual properties of the object type, such as the shape, icon or label, may be specified using the components in the corresponding tabbed panes in the dialog.

5. Click the **create** button

The object type will be created but the dialog will persist until cancelled so as additional object types can be created.

2.2 Creating an Object Type Hierarchy

1. Select the option from the toolbar

2. Click on the sub-type

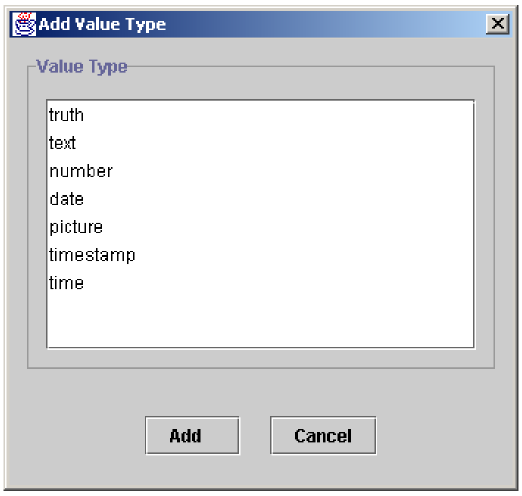
3. Click on the super-type

Providing that a cycle would not be introduced in the object type hierarchy, in which case a dialog will be displayed stating that the operation could not be performed, an is-a relationship will be added between the object types selected.

2.3 Adding a Value Type

1. Select the option from the toolbar

2. Click on the main panel where the value type should initially be placed, a dialog similar to the one in figure 2.3.1 will be displayed



truth
text
number
date
picture
timestamp
time

Figure 2.3.1. The add value type dialog

3. Select the appropriate value type from the list in the **value type** panel

4. Click the **Add** button

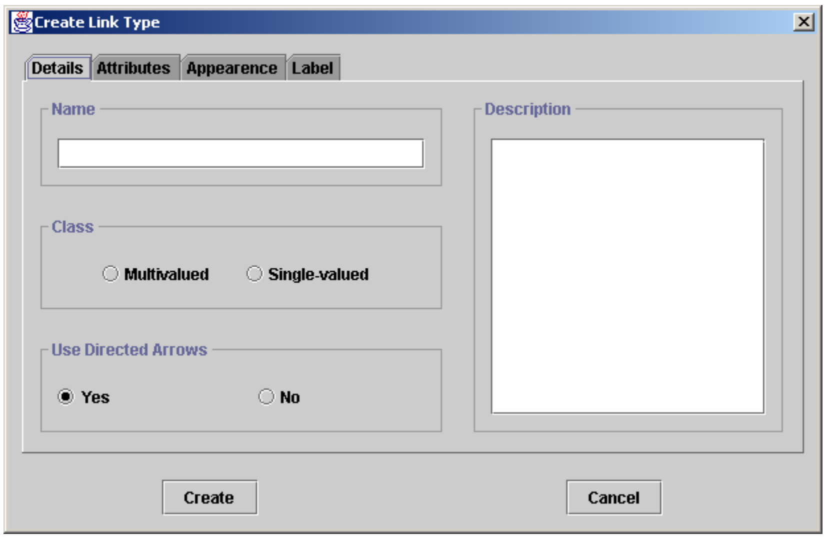
The value type will be added but the dialog will persist until cancelled so as additional value types can be added.

2.4 Creating a New Link Type

1. Select the → option from the toolbar

2. Click on the data type that the link type is to be defined from

3. Click on the data type that the link type is to be defined to, a dialog similar to that in figure 2.4.1 will be displayed



Multivalued Single-valued

Use Directed Arrows: [✓ Yes] No

Figure 2.4.1. The create link type dialog

4. Enter the name of the link type in the text field in the **name** panel

The name must not be the same as any other link type currently defined in the database schema that is defined from the same object type and must consist only of letters, digits and white space.

5. Select the class of the link type by selecting either the **single-valued** button or the **multivalued** button in the **class** panel

To add an attribute select the **attributes** tabbed pane then click the **add** button, a dialog similar to the one in figure 2.4.2 will be displayed.

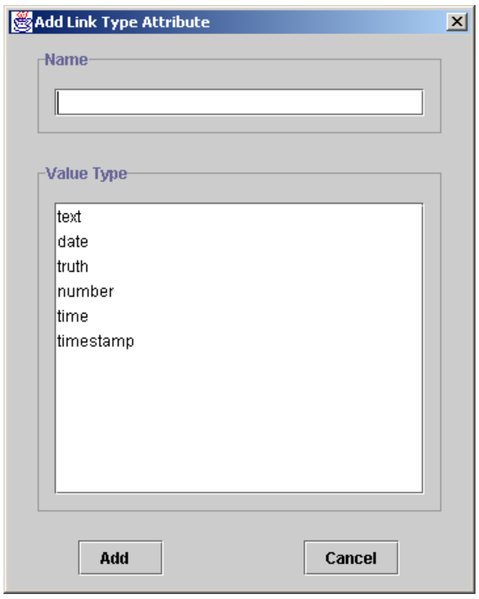
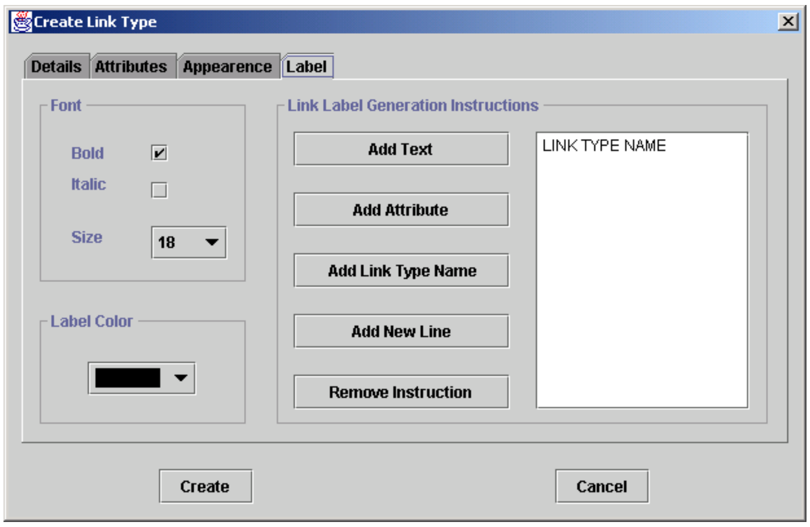


Figure 2.4.2. The add link type attribute dialog

Enter the name of the attribute in the text field in the **name** panel. The name must not be the same as any other attribute currently defined for the link type and must consist only of letters, digits and white space. Select the associated value type from the list in the **value type** panel then click the **add** button.

An attribute may be edited by selecting the attribute from the list in the **attributes** panel then following a similar procedure to that described above for adding an attribute, except that the **edit** button should be clicked. An attribute may be removed by selecting the attribute from the list in the **attributes** panel then clicking the **remove** button.

The label given to a link that is an instance of this link type when added to a link chart in EDVC is by default the name of the link type. To alter this select the **label** tabbed pane, an example of which is displayed in figure 2.4.3.



LINK TYPE NAME

Figure 2.4.3. The label tabbed pane in the create link type dialog

The instructions used by EDVC to generate the label of a link of this type when added to a link chart are listed in the **link label generation instructions** panel. These instructions are executed from top till bottom by EDVC to generate the label of a link.

To add an instruction that will add a piece of literal text to the label click the **add text** button, a dialog similar to the one in figure 2.4.4 will be displayed.

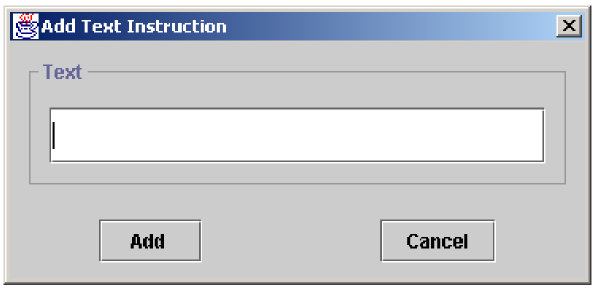
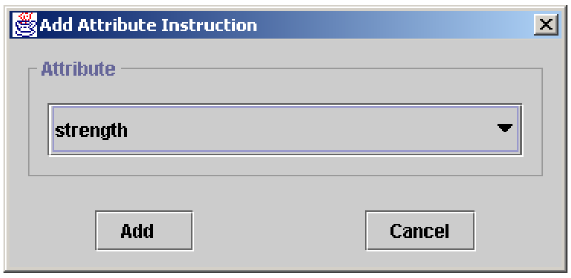


Figure 2.4.4. The add text instruction dialog

Enter the text to be used in the text field in the **text** panel then click the **add** button, the instruction will be appended to the end of the list. To add an instruction that will add the value of one of the link's attributes click the **add attribute** button, a dialog similar to that in figure 2.4.5 will be displayed.



strength

Figure 2.4.5. The add attribute dialog

Select the desired attribute from the list in the **attribute** panel then click the **add** button, the instruction will be appended to the end of the list. If the link added to a link chart does not have a value defined for this attribute then no action is taken for such an instruction.

To add an instruction that will add a new-line character or the name of the link type click the button with the corresponding label. To remove an instruction, select the instruction then click the **remove** button.

To create the link type click the **create** button.

3 Editing the Database Schema

3.1 Editing a Schema Element

1. Select the option from the toolbar

2. Click on the schema element that you wish to edit[2], an edit dialog similar to the one used to create the element will be displayed

Apart from the data types connected by link types and is-a relationships any property of the element selected may be altered. To do so follow the procedure described in section 2 for setting that property during the creation of the element.

If an object type is edited then the instructions used to generate the label of an object that is an instance of the object type when added to a link chart in EDVC may also be set. The manner in which this is done is analogous to the way in which such instructions are specified for generating the labels of links (described in section 2.4).

4. Click the **edit** button

3.2 Moving a Data Type

1. Select the option from the toolbar

2. Drag the data type to the desired location

3.3 Removing a Schema Element

1. Select the option from the toolbar

2. Click on the element

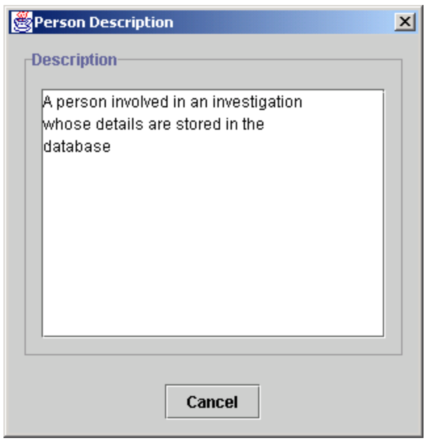
If the element removed was a data type then all link types and is-a relationships that involved the data type are also removed.

[2] Link types and is-a relationships are selected by clicking on their label or arrow.

3.4 Displaying the Properties of a Schema Element

1. **Right-click** on the element

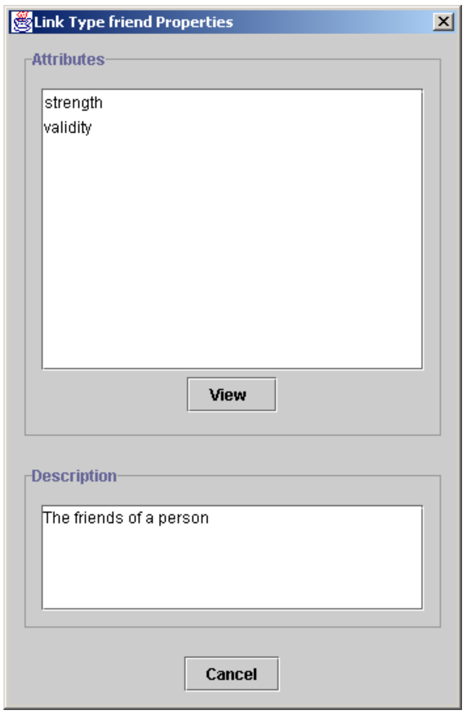
If the element selected is a data type and a description has been provided then a dialog similar to the one in figure 3.4.1 will be displayed.



A person involved in an investigation whose details are stored in the database

Figure 3.4.1. A dialog displaying the description of a data type

If the data type selected does not have a description then a dialog will be displayed conveying this information. If the element selected is a link type and a description has been provided, and/or the link type has attributes defined, then a dialog similar to the one in figure 3.4.2 will be displayed.



strength
validity

The friends of a person

Figure 3.4.2. A dialog displaying the properties of a link type

If the link type selected does not have a description and the link type does not have attributes defined then a dialog will be displayed conveying this information.

3.5 Editing a Schema's Description

1. Select **edit description** from the **schema** menu, a dialog similar to the one in figure 3.5.1 will be displayed

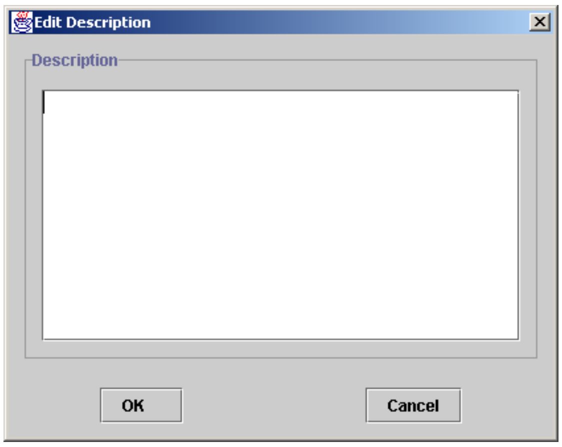


Figure 3.5.1. A dialog for editing a schema's description

2. Enter the description in the text field in the **description** panel

3. Click the **OK** button

4 Panning & Printing the Database Schema

The pan window, located in the top right hand corner of the main panel, displays a miniature version of the schema constructed. The part of the schema visible in the main panel appears within the red rectangle located in the pan window. The part displayed may be altered by dragging the red rectangle to the appropriate location. A copy of the whole of the schema, not just the part displayed in the main panel, may be obtained by selecting **print** from the **schema** menu.