Interactive Query Formulation using Spider Queries

Confidential

Asymetrix Report 94-2

H.A. Proper
Asymetrix Research Laboratory
Department of Computer Science
University of Queensland
Australia 4072
E.Proper@acm.org

Version of June 23, 2004 at 10:29

PUBLISHED AS:

H.A. Proper. Interactive query formulation using spider queries. Asymetrix Research Report 94-2, Asymetrix Research Laboratory, University of Queensland, Brisbane, Australia, 1994.

Abstract

Effective information disclosure in the context of databases with a large conceptual schema is known to be a non-trivial problem. In particular the formulation of ad-hoc queries is a major problem in such contexts. Existing approaches for tackling this problem include graphical query interfaces, query by navigation, query by construction, and point to point queries. In this article we propose the spider query mechanism as a final corner stone for an easy to use computer supported query formulation mechanism for InfoAssisant.

The basic idea behind a spider query is to build a (partial) query of all information considered to be relevant with respect to a given object type. The result of this process is always a tree that fans out over existing conceptual schema (a spider).

We also provide a brief discussion on the integration of the spider quer mechanism with the existing query by navigation, query by construction, and point to point query mechanisms.

1 Introduction

Most present day organisations make use of some automated information system. This usually means that a large body of vital corporate information is stored in these information systems. As a result an essential function of information systems is the support of disclosure of this information. Without a set of adequate information disclosure avenues an information system becomes worthless since there is no use in storing information that will never be retrieved. An adequate support for information disclosure, however, is far from a trivial problem. Most query languages do not provide any support for the users in their quest for information. Furthermore, the conceptual schemata of real-life applications tend to be quite large and complicated. As a result, the users may easily become lost in conceptual space and they will end up retrieving irrelevant (or even wrong) objects and may miss out on relevant objects. Retrieving irrelevant objects leads to a low precision, missing relevant objects has a negative impact on the *recall* ([SM83]).

The disclosure of information stored in an information system has some clear parallels to the disclosure problems encountered in *document retrieval systems*. To draw this parallel in more detail, we quote the information retrieval paradigm as introduced in [BW92]. The paradigm starts with an individual or company having an *information need* they wish to fulfil. This need is typically a vague notion and needs to be made more concrete in terms of an *information request* (the query) in some (formal) language. The information request should be as good as possible a description of the information need. The information request is then passed on to an automated system, or a human intermediary, who will then try to fulfil the information request using the information stored in the system. This is illustrated in the *information disclosure*, or *information retrieval paradigm*, presented in figure 1 which is taken from [BW92].

We now briefly discuss why the information retrieval paradigm for document retrieval systems is also applicable for information systems. For a more elaborate discussion on the relation between information systems and document (information) retrieval systems in the context of the information retrieval paradigm, refer to [Pro94a]. In the paradigm, the retrievable information is modelled as a set $\mathcal{K}$ of *information objects* constituting the *information base* (or population).

In a document retrieval system the information base will be a set of documents ([SM83]), while in the case of an information system the information base will contain a set of facts conforming to a conceptual schema. Each information object $o \in \mathcal{K}$ is *characterised* by a set of descriptors $\mathcal{X}(o)$ that facilitates its disclosure. The characterisation of information objects is carried out by a process referred to as indexing. In an information system, the stored objects (the population or information base) can always be identified by a set of (denotable) values, the identification of the object. For example, an address may be identified as a city name, street name, and house number. The characterisation of objects in an information system is directly provided by the reference schemes of the object types.

The actual information disclosure is driven by a process referred to as *matching*. In document retrieval applications this matching process tends to be rather complex. The characterisation of documents is known to be a hard problem ([Mar77], [Cra86]), although newly developed approaches turn out to be quite successful ([Sal89]). In information systems the matching process is less complex as the objects in the information base have a more clear characterisation (the identification). In this case, the identification of the objects (facts) is simply related to the query formulation q by some (formal) query language.

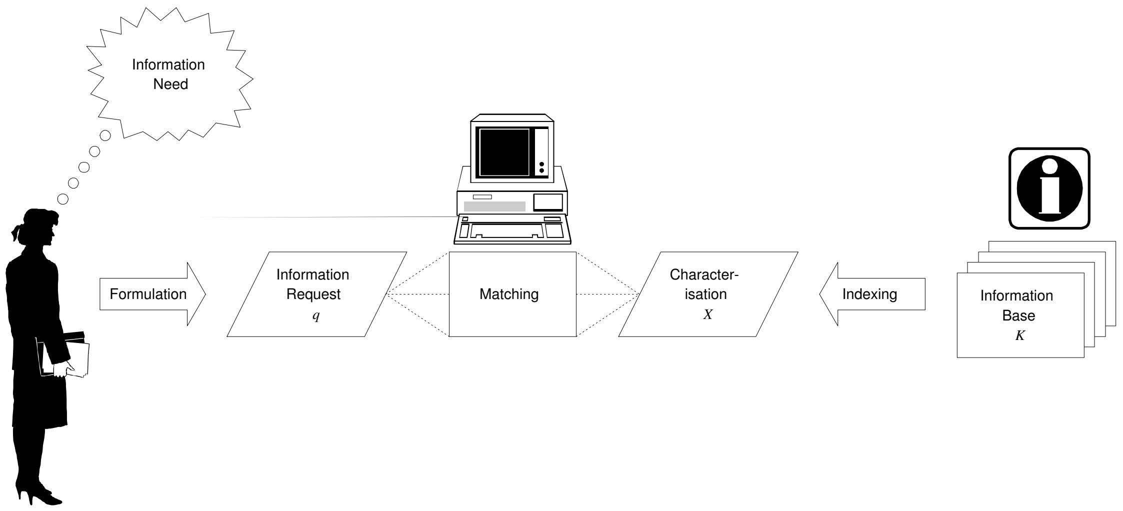


Figure 1: The information retrieval paradigm

The remaining problem is the query formulation process itself. An easy and intuitive way to formulate queries is absolutely essential for an adequate information disclosure. Quite often, the quest from users to fulfil their information need can be aptly described by ([Bru93]):

I don't know what I'm looking for, but I'll know when I find it.

In document retrieval systems this problem is attacked by using *query by navigation* ([BW92], [Bru93]) and *relevance feedback* mechanisms ([Rij89]). The query by navigation interaction mechanism between a searcher and the system is well-known from the Information Retrieval field, and has proven to be useful. It shall come as no surprise that these mechanisms also apply to the query formulation problem for information systems. In [BPW93], [BPW94], [HPW94b], [Pro94a] such applications of the *query by navigation* and *relevance feedback* mechanisms have been described before. When combining the query by navigation and manipulation mechanisms with the ideas behind visual interfaces for query formulation as described in e.g. [ADD$^+$92] and [Ros94] powerfull and intuitive tools for computer supported query formulation become feasible. Such tools will also heavily rely on the ideas of direct manipulation interfaces ([Sch83]) as used in present day computer interfaces.

One important step in the improvement of the information disclosure of information systems, is the introduction of query languages on a conceptual level. Examples of such conceptual query languages are RIDL ([Mee82]), LISA-D ([HPW93], [HPW94a]), and FORML ([HHO92]). By letting users formulate queries on a conceptual level, users are safeguarded from having to know the exact mapping to internal representations (e.g. a set of tables which conform to the relational model) to be able to formulate queries in a non conceptual language such as SQL. The next step is to introduce ways to support users in the formulation of queries in such conceptual query languages (CQL).

In line with the above discussed information retrieval paradigm and the notion of relevance feedback, a query formulation process (both for a document retrieval system, and an information system) can be said to roughly consist of the following four phases:

1. The *explorative phase*. What information is there, and what does it mean?
2. The *constructive phase*. Using the results of phase 1, the actual query is formulated.
3. The *feedback phase*. The result from the query formulated in phase 2 may not be completely satisfactory. In this case, phases 1 and 2 need to be re-done and the result refined.
4. The *presentation phase*. In most cases, the result of a query needs to be incorporated into a report or some other document. This means that the results must be grouped or aggregated in some form.

Depending on the user's knowledge of the system, the importance of the respective phases may change. For instance, a user who has a good working knowledge of the structure of the stored information may not require an elaborate first phase and would like to proceed with the second phase as soon as possible.

In this report, we discuss an additional mechanism to support automated disclosure of information stored in information systems, the spider query mechanism. As stated before, the related notions of *query by navigation* and *query by construction* have already been discussed in [BPW93], [PW95], [Pro94a]. The point to point query mechanism was already discussed in [Pro94b].

The idea behind spider queries is to start out from one object type, and to associate all information that is relevant to this object type. The essential part of a spider query is selecting the object types in the direct suroundings of the initial object type that are considered to be relevant. This style of querying corresponds to a situation where users only know about the existance of some object types in the conceptual schema about which they would like to be informed.

The structure of this report is as follows. In section 2, we discuss an example spider query session, and elaborate briefly on the integration with the existing query by navigation, query by construction, and point to point querie mechanisms. Section 3 deals with the representation of conceptual schema as a graph. Building a spider query (essentailly also a graph) is covered in section 4. Before concluding, section section 5 discusses the representation of a spider query as a path expression. For the reader who is unfamiliar with the notation style used in this report, it is advisable to first read [Pro94c].

2 An Example Spider Query Session

In this section we discuss a sample session involving a spider query, and also discuss briefly the relationship to the existing query by navigation, query by construction and point to point queries. The discussed example operates on a conceptual schema for the administration of the election of American presidents. The example schema itself is not shown; the structure of the domain will become clear from the sample session. Note that the quality of the verbalisations of the paths in the examples used in this section should be improved, however, this is subject of further research. In figure 2 a possible screen is depicted for building queries using a point to point query mechanism. No special window is needed for a spider query (see also [Pro94b]).

We start out from an existing query in a query by construction window. Note that this could also be single object type, e.g. politician. The spider query mechanism adds one important aspect to the query by construction window, the spider button: ➢. When a user presses this button, the system calculates the spider query of the object type directly to the right of the button. This is illustrated in figure 3. The system allows for the removal of parts of the resulting spider query that are not considered to be relevant by the user. Suppose the user is not interested in administration is headed by and election won by, then these paths can be deleted, which leads to the screen depicted in figure 4.

It is now interesting to see that a query essentially is a double tree with a shared root (politician in the example). Furthermore, the leaves on the tree resulting from the spider query can be extended further if desired by commencing new spider queries. Finally, since the result of a spider query is constructed from path expressions as well, these expressions have the ⬇ associated that can be used to select alternative paths between the head and tail object types. Furthermore, the paths can also be used as a starting point of a query by navigation session. This latter posibility is illustrated in figure 5.

As stated before in [Pro94b], the query by construction window is basically a syntax directed editor. In the left part of the window all possible constructs from the query language are listed. In our examples we have used the constructs defined in LISA-D. Once the FORML and LISA-D languages have been merged, a more complete language for the query by construction part will result.

3 A Conceptual Schema as a Graph

For the purpose of finding a path between object types in a conceptual schema, the schema first needs to be translated to a graph. This translation is exactly the same as provided in [Pro94b], but for reasons of completeness we provide it again. We start out from a formalisation of ORM based on the one used in ([HP95]). However, since only a very limited part of the formalisation is needed, we do not cover the formalisation in full detail.

A conceptual schema is presumed to consist of a set of types $\mathcal{TP}$. Within this set of types two subsets can be distinguished: the relationship types $\mathcal{RL}$, and the object types $\mathcal{OB}$. Furthermore, let $\mathcal{RO}$ be the set of roles in the conceptual schema. The fabric of the conceptual schema is then captured by two functions and two predicates. The set of roles associated to a relationship type are provided by the partition: $\mathsf{Roles} : \mathcal{RL} \to \wp(\mathcal{RO})$. Using this partition, we can define the function Rel which returns for each role the relationship type in which it is involved: $\mathsf{Rel}(r) = f \iff r \in \mathsf{Roles}(f)$. Every role has an object type at its base called the player of the role, which is provided by the function: $\mathsf{Player} : \mathcal{RO} \to \mathcal{TP}$. Subtyping and polymorphy of object types are captured by the predicates $\mathsf{SpecOf} \subseteq \mathcal{OB} \times \mathcal{OB}$ and $\mathsf{HasMorph} \subseteq \mathcal{OB} \times \mathcal{OB}$ respectively. For any ORM conceptual schema the following (undirected) labelled graph $G = \langle N, E \rangle$ can then be defined:

$$N \triangleq \mathcal{TP} \tag{1}$$

$$E \triangleq \left\{ \langle \{\mathsf{Player}(r), \mathsf{Rel}(r)\}, r \rangle \;\middle|\; r \in \mathcal{RO} \right\} \tag{2}$$

$$\bigcup \left\{ \langle \{x, y\}, \mathsf{SpecOf} \rangle \;\middle|\; x \,\mathsf{SpecOf}\, y \right\} \tag{3}$$

$$\bigcup \left\{ \langle \{x, y\}, \mathsf{HasMorph} \rangle \;\middle|\; x \,\mathsf{HasMorph}\, y \right\} \tag{4}$$

The edges in the resulting graph have the format $\langle \{x, y\}, l \rangle$, where x and y are the source/destination (no order) of the edge, and l is the label of the edge. The labels on the edges either result from the roles in the relationship types (2), or they result from specialisation or polymorphism (3,4). In the remainder, the graph G will be used as an implicit parameter for all introduced functions and operations. As a convention, the nodes of graph G are accessed by $G.N$, and the edges by $G.E$.

As an example, consider the conceptual schema depicted in figure 6. For this schema we have:

$\mathcal{TP} = \{A, B, C, D, f, g\}$	$\mathsf{Roles}(f) = \{r, s\}, \mathsf{Roles}(g) = \{t, u\}$
$\mathcal{RL} = \{f, g\}$	$\mathsf{Player}(r) = A, \mathsf{Player}(s) = B, \mathsf{Player}(t) = C, \mathsf{Player}(u) = A$
$\mathcal{OB} = \{A, B, C, D\}$	$A \,\mathsf{HasMorph}\, C, A \,\mathsf{HasMorph}\, g$
$\mathcal{RO} = \{r, s, t, u\}$	$D \,\mathsf{SpecOf}\, B$

From this schema the graph as depicted in figure 7 can be derived.

Note that for spider queries, it might be usefull to limit the edges resulting from roles to those roles which are *not* used in the reference schemas of other object types. We cannot yet decide on this until we have finallised the path expression language, but it is simply a matter of filtering the proper edges.

4 Building a Spider Query

The construction of a spider query corresponds to the construction of a graph. The nodes and edges in this graph are based on the nodes (object types) and edges from the original conceptual schema graph. A *spider query graph* is a labelled tree (connected directed acyclic graph). It is not just a subgraph of the conceptual schema graph, since one object type can be visited more than once on different branches of the graph (legs of the spider query graph). Let $\mathbb{N}$ be the set of all nodes that can occur in a spider query, and let $\mathbb{L} \triangleq \mathcal{RO} \cup \{\mathsf{SpecOf}, \mathsf{HasMorph}\}$ be the set of labels that can occur in a spider query graph. The spider query graph itself is now constructed by the function:

$$\mathsf{SpiderQuery} : \mathbb{N} \rightarrow (\mathbb{N} \rightarrowtail \mathcal{TP}) \times \wp(\mathbb{N} \times \mathbb{N} \times \mathbb{L})$$

The result of a spider query $\mathsf{SpiderQuery}(x)$ is a tuple $\langle O, S \rangle$, where $O : \mathbb{N} \rightarrowtail \mathcal{TP}$ provides the relation between the spider query graph and the conceptual schema, and the spider query graph itself is defined by the (directed!) edges in $S \subseteq \wp(\mathbb{N} \times \mathbb{N} \times \mathbb{L})$.

The construction process itself is driven by a recursive function σ, which is activated as follows:

$$\mathsf{SpiderQuery}(x) \triangleq \sigma(\{\langle \mathsf{NewNode}(x), x \rangle\}, \varnothing, \{x\})$$

where $\mathsf{NewNode}$ is a function returning a new node each time it is called. The σ function is the actual engine of the construction process. This function tries to extend the spider query graph in a number of steps. In each step the possible extensions of the existing graph at that moment are calculated by the ε function. The ε function takes as parameters the current spider query graph: O, S, and the nodes which are allowed to be extended: T. It returns a number of tuples of the form $\langle n, t, l \rangle$, where n is an extendable node in the existing spider query graph, and t is a type in the conceptual schema that is reachable in (the conceptual schema graph) from object type $O(n)$. A restriction on the returned tuples is that no cycles may be formed, i.e. no revisiting of object types on one path in the spider query graph. The formal definition is given by:

$$\varepsilon(O, S, T) \triangleq \left\{ \langle n, t, l \rangle \;\middle|\; \langle \{O(n), t\}, l \rangle \in G'.E \wedge n \in T \wedge t \notin \mathsf{Top}(n) \right\}$$

where $\mathsf{Top}(n) \triangleq \bigcup_{m:\langle m,n,l \rangle \in S} (\mathsf{Top}(m) \cup \{m\})$ is the set of types on the path in the spider query graph leading from the root to n.

The actual driver function σ evolves around three parameters. These parameters are updated in every step of the function. In the definition we use O, S and T as variable names, where T contains the nodes in the spider query graph that may be used for further extensions, and O and S represent the spider query graph so-far. The σ function is identified by:

$$\sigma(O,S,T) \triangleq \begin{cases} \sigma(O',S',T') & \text{if } \varepsilon(O,S,T) \neq \varnothing \\ \langle O,S \rangle & \text{otherwise} \end{cases}$$

where:

$$O' = O \cup \big\{ \langle \mathsf{NewNode}(t), t \rangle \;\big|\; t \in \pi_2\, \varepsilon(O,S,T) \big\} \tag{5}$$

$$S' = S \cup \big\{ \langle n,m,l \rangle \;\big|\; \langle n,t,l \rangle \in \varepsilon(O,S,T) \wedge O'(m) = t \big\} \tag{6}$$

$$T' = \big\{ m \;\big|\; \langle n,m,l \rangle \in \varepsilon(O,S,T) \wedge \mathsf{CWeight}(O'(m)) \leq \mathsf{CWeight}(O'(n)) \big\} \tag{7}$$

In 4, the newly found object types in $\varepsilon(O,S,T)$ are assigned a new node so that they can be added to the existing spider query graph. The set of edges of the spider query graph is updated in 5. The new set of nodes that will be considered for further extensions in the next step of σ are determined by 6. In this definition, the conceptual weight function CWeight is the same function as used in [Pro94b], and should provide the conceptual importance of each object type. This importance could for instance be based on the abstraction level at which the object type occurs ([CH94]). The rationale behind the use of the CWeight function is that as soon as the conceptual importance increases, any new neigbouring object type from the last added object type (the one with the increased CWeight) is not relevant for the root of the current spider query graph, i.e. we have left the relevance scope of the current root. Note, however, the neighbours of the last added object type could quite well be relevant for a spider query starting out from this latter object type. These nodes will be added if the user presses the spider query button that will be associated to this node.

As an example of the operation of the SpiderQuery function, consider the graph depicted in figure 8. Each edge in this graph is labelled, and each node has associated its name (object type), and the conceptual weight. In figure 9 the tree is depicted with which the σ function is started. The double circle around node B is used to indicate that node B is in the set of extendible nodes T.

Figure 10 depicts the spider query graph after one incremental step of σ. All neighbours of B are added to the graph, and since they do not have a higher conceptual weight than B, they can be used for further extensions.

The next step of the algorithm is illustrated in figure 11. Node A has two neigbours: B and F. However, since adding B to the spider query graph would lead to a repetition of an existing node on the path to the root of the spider query graph, B is not added. Similarly, B is not added as a neighbour of C and F. Nodes D and E are not marked as points of further extensions since they both have a higher conceptual weight then node C. As a resuolt, node H is not part of node B's scope of rellevance. However, node B would be part of a spider query starting from H. Note that nodes D and E will both have associated a spider query button when the result is presented to the user, so a user can always explicitly decide to further 'climb the conceptual importance mountain'.

The result of the last step of σ is shown in figure 12. The second node A, and node G, do not lead to further extensions. All neighbours of these two nodes are already present on the paths to the root of the graph. Only the leftmost node F leads to a further extension with a G. After this extension no further extensions are possible.

Finally, it is good to realise that the σ function always terminates:

Lemma 4.1 The σ function always terminates.

Proof:

This corresponds to saying that the resulting graph is finite.

The number of object types in an ORM schema is (presumed to be) finite, and the paths of the spider query will not contain cycles (follows from the definition of ε).

As a result, each node only has a finite number of outgoing arcs, and each path from the root to a leaf is finite. Hence the resulting graph is finite. □

5 The Resulting Path Expressions

In this section we discuss how to transfer a spider query graph into a path expression. We use the spider query graph O, S as an implicit parameter for all definitions in this section.

Given a node x in the spider query graph, then the following path expression can be associated to this node:

$$\mathsf{NodeExpr}(x) \triangleq \begin{cases} [a_1 : \mathsf{PathSeg}(y_1, l_1, x), \ldots, a_n : \mathsf{PathSeg}(y_n, l_n, x); O(x)] & \text{if } S \neq \varnothing \\ O(x) & \text{otherwise} \end{cases}$$

where $S = \{\langle y_1, l_1\rangle, \ldots, \langle y_n, l_n\rangle\}$ is a set such that $S = \{\langle y, l\rangle \mid \langle x, y, l\rangle \in S\}$, and $a_1, \ldots, a_n$ is a set of fresh attribute names. A good choice for these latter names are the names of the object types where the PathSegs end (suffixed with a number to make the name unique if needed). The $\circ$ operation is the concatenation operation for path expressions, and the $[X_1, \ldots, X_n]$ construct is the path confluence operation. It allows us to combine a variety of path expressions. For more details about the path expression operators, refer to [HPW93] and the forthcomming Asymetrix report on path expressions. One single edge from the spider query graph is converted to a path expression as follows:

$$\mathsf{PathSeg}(y, l, x) \triangleq \mathsf{NodeExpr}(y)\ \mathsf{Connector}(l, x)\ O(x)$$

where

$$\mathsf{Connector}(l, x) \triangleq \begin{cases} \circ & \text{if } l \in \{\mathsf{HasMorph}, \mathsf{SpecOf}\} \\ \circ\, l\, \circ & \text{if } x \in \mathcal{RL} \wedge l \in \mathsf{Roles}(x) \\ \circ\, l^{\leftarrow}\, \circ & \text{otherwise} \end{cases}$$

The linear path expressions are for internal use only. They can be mapped to proper SQL queries on the one hand, and verbalised as semi-natural language sentences using the verbalisation information as provided in the conceptual schema on the other hand. As stated before, the verbalisation of path expressions is subject of further research.

Finally, the (unique) root of a spider query graph is determined as follows:

$$\mathsf{IsRoot}(r) \iff O{\downarrow}r \wedge \neg\exists_{x,l}\left[\langle x,r,l\rangle \in S\right]$$

If r is the (unique) root of a spider query graph, then $\mathsf{NodeExpr}(r)$ results in the complete path expression for this spider query graph.

6 Conclusions

In this article we introduced a novel way to computer supported query formulation called spider queries. We provided a sample session with a provisional tool supporting spider queries, and briefly discussed the relationship to query by navigation, query by construction, and point to point queries. Together with these existing mechanisms a powerfull query formulation tool can now indeed be build.

As a next step, the path expressions should be further developed to suit our needs. Furthermore, elegant verbalisations of the path expressions should be catered for.

asy

References

[ADD$^+$92] A. Auddino, Y. Dennebouy, Y Dupont, E. Fontana, S. Spaccapietra, and Z. Tari. SUPER - Visual Interaction with an Object-based ER Model. In G. Pernul and A.M. Tjoa, editors, *11th International Conference on the Entity-Relationship Approach*, volume 340–356 of *Lecture Notes in Computer Science*, pages 423–439. Springer-Verlag, 1992.

[BPW93] C.A.J. Burgers, H.A. Proper, and Th.P. van der Weide. Organising an Information System as Stratified Hypermedia. In H.A. Wijshoff, editor, *Proceedings of the Computing Science in the Netherlands Conference*, pages 109–120, Utrecht, The Netherlands, EU, November 1993.

[BPW94] C.A.J. Burgers, H.A. Proper, and Th.P. van der Weide. An Information System organized as Stratified Hypermedia. In N. Prakash, editor, *CISMOD94, International Conference on Information Systems and Management of Data*, pages 159–183, Madras, India, October 1994.

[Bru93] P.D. Bruza. *Stratified Information Disclosure: A Synthesis between Information Retrieval and Hypermedia*. PhD thesis, University of Nijmegen, Nijmegen, The Netherlands, EU, 1993.

[BW92] P.D. Bruza and Th.P. van der Weide. Stratified Hypermedia Structures for Information Disclosure. *The Computer Journal*, 35(3):208–220, 1992.

[CH94] L.J. Campbell and T.A. Halpin. Abstraction Techniques for Conceptual Schemas. In R. Sacks-Davis, editor, *Proceedings of the 5th Australasian Database Conference*, volume 16, pages 374–388, Christchurch, New Zealand, January 1994. Global Publications Services.

[Cra86] T.C. Craven. *String Indexing*. Academic Press, London, United Kingdom, 1986.

[HHO92] T.A. Halpin, J. Harding, and C-H. Oh. Automated Support for Subtyping. In B. Theodoulidis and A. Sutcliffe, editors, *Proceedings of the Third Workshop on the Next Generation of CASE Tools*, pages 99–113, Manchester, United Kingdom, May 1992.

[HP95] T.A. Halpin and H.A. Proper. Subtyping and Polymorphism in Object-Role Modelling. *Data & Knowledge Engineering*, 15:251–281, 1995.

[HPW93] A.H.M. ter Hofstede, H.A. Proper, and Th.P. van der Weide. Formal definition of a conceptual language for the description and manipulation of information models. *Information Systems*, 18(7):489–523, October 1993.

[HPW94a] A.H.M. ter Hofstede, H.A. Proper, and Th.P. van der Weide. A Conceptual Language for the Description and Manipulation of Complex Information Models. In G. Gupta, editor, *Seventeenth Annual Computer Science Conference*, volume 16 of *Australian Computer Science Communications*, pages 157–167, Christchurch, New Zealand, January 1994. University of Canterbury. ISBN 047302313

[HPW94b] A.H.M. ter Hofstede, H.A. Proper, and Th.P. van der Weide. Supporting Information Disclosure in an Evolving Environment. In D. Karagiannis, editor, *Proceedings of the 5th International Conference DEXA'94 on Database and Expert Systems Applications*, volume 856 of *Lecture Notes in Computer Science*, pages 433–444, Athens, Greece, EU, September 1994. Springer Verlag, Berlin, Germany, EU. ISBN 3540584358

[Mar77] M.E. Maron. On Indexing, Retrieval and the Meaning of About. *Journal of the American Society for Information Science*, 28(1):38–43, 1977.

[Mee82] R. Meersman. The RIDL Conceptual Language. Research report, International Centre for Information Analysis Services, Control Data Belgium, Inc., Brussels, Belgium, 1982.

[Pro94a] H.A. Proper. *A Theory for Conceptual Modelling of Evolving Application Domains*. PhD thesis, University of Nijmegen, Nijmegen, The Netherlands, EU, 1994. ISBN 909006849X

[Pro94b] H.A. Proper. Interactive query formulation using point to point queries. Asymetrix Research Report 94-1, Asymetrix Research Laboratory, University of Queensland, Brisbane, Australia, 1994.

[Pro94c] H.A. Proper. Introduction to formal notations. Asymetrix Research Report 94-0, Asymetrix Research Laboratory, University of Queensland, Brisbane, Australia, 1994.

[PW95] H.A. Proper and Th.P. van der Weide. Information Disclosure in Evolving Information Systems: Taking a shot at a moving target. *Data & Knowledge Engineering*, 15:135–168, 1995.

[Rij89] C. J. van Rijsbergen. Towards an information logic. In *Proceedings of the 12th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 77–86, Cambridge, Massachusetts, United States, June 1989. ACM Press.

[Ros94] P. Rosengren. Using Visual ER Query Systems in Real World Applications. In G.M. Wijers, S. Brinkkemper, and T. Wasserman, editors, *Proceedings of the Sixth International Conference CAiSE'94 on Advanced Information Systems Engineering*, volume 811 of *Lecture Notes in Computer Science*, pages 394–405, Utrecht, The Netherlands, June 1994. Springer-Verlag.

[Sal89] G. Salton. *Automatic Text Processing–The Transformation, Analysis, and Retrieval of Information by Computer*. Addison-Wesley, Reading, Massachusetts, 1989.

[Sch83] B. Schneiderman. Direct Manipulation: A Step Beyond Programming Languages. *IEEE Computer*, 16(8):57–69, 1983.

[SM83] G. Salton and M. J. McGill. *Introduction to Modern Information Retrieval*. McGraw-Hill New York, NY, 1983.

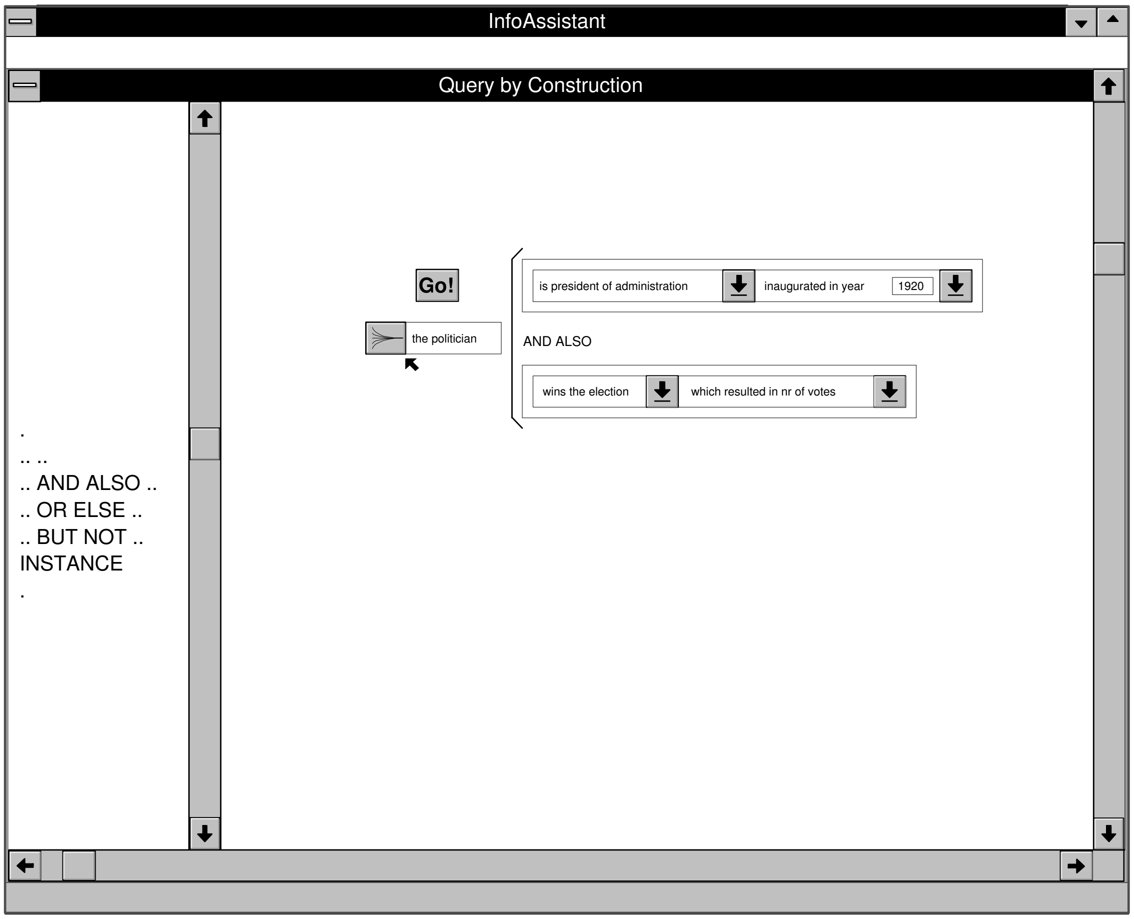


Figure 2: Start of a spider query

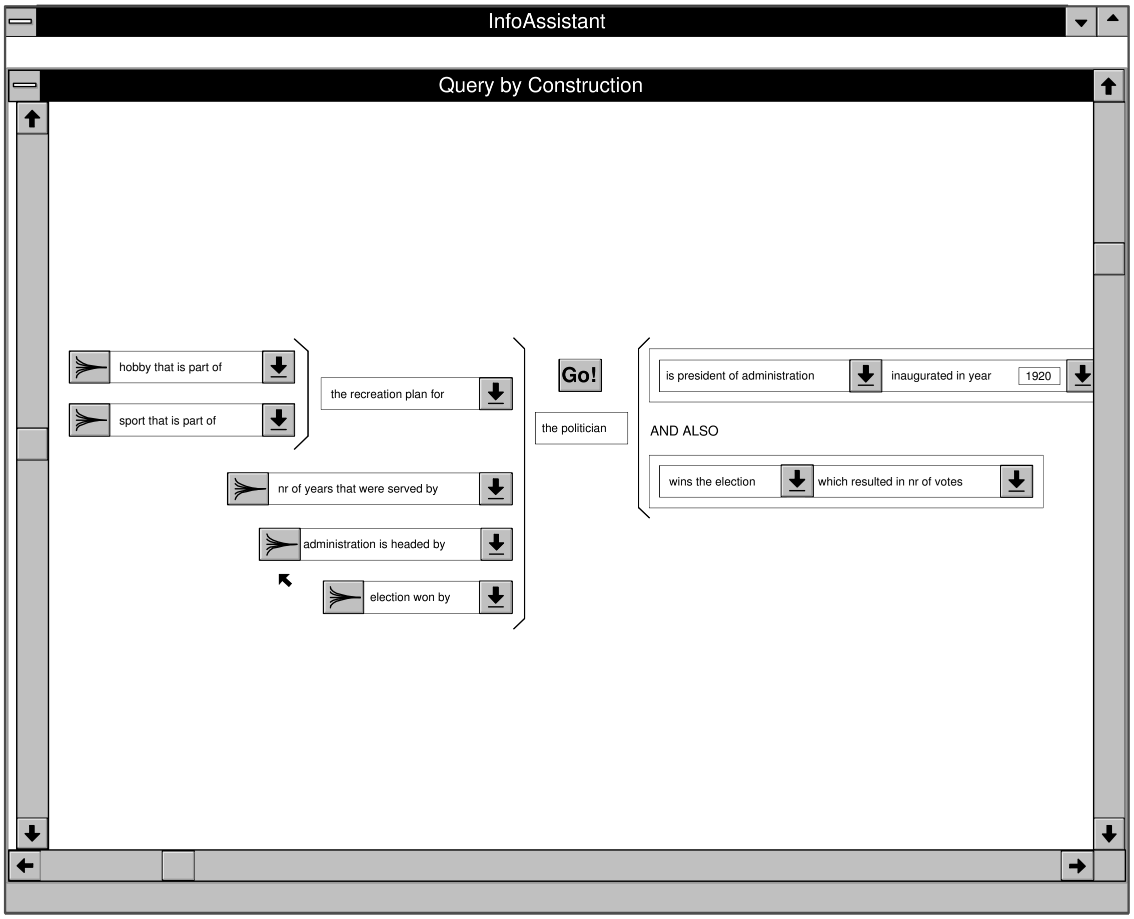


Figure 3: Result of a Spider Query

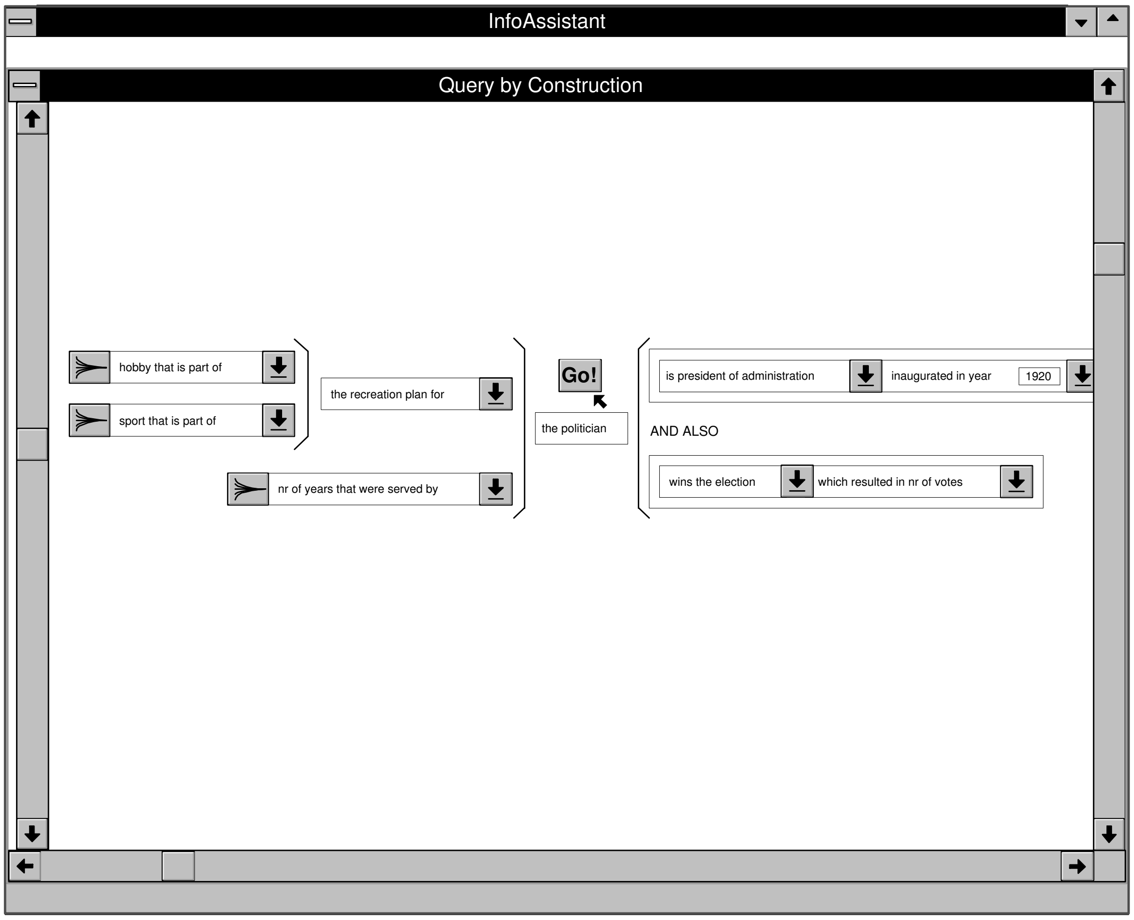
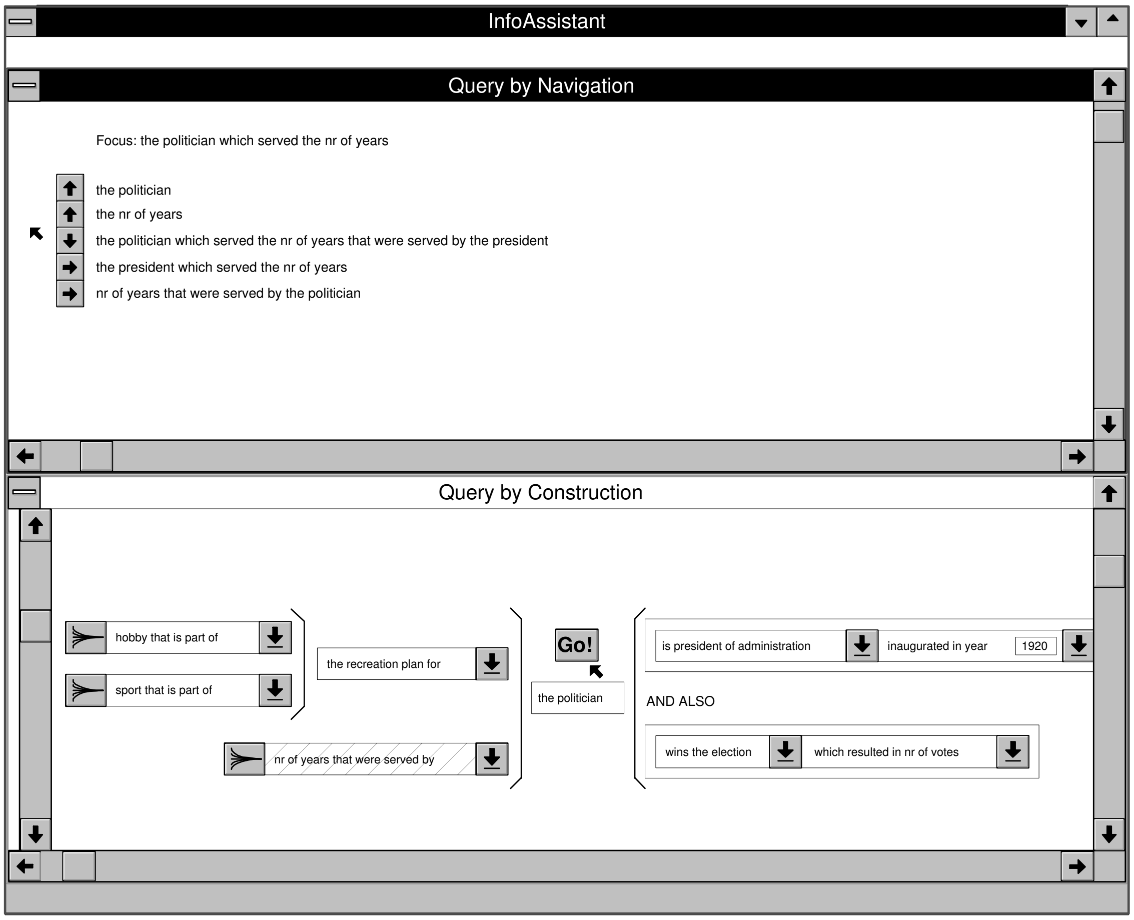


Figure 4: Pruning the Spider Query



Focus: the politician which served the nr of years

- the politician
- the nr of years
- the politician which served the nr of years that were served by the president
- the president which served the nr of years
- nr of years that were served by the politician

hobby that is part of

sport that is part of

the recreation plan for

nr of years that were served by

the politician

is president of administration inaugurated in year 1920

AND ALSO

wins the election which resulted in nr of votes

Figure 5: Switching to query by navigation

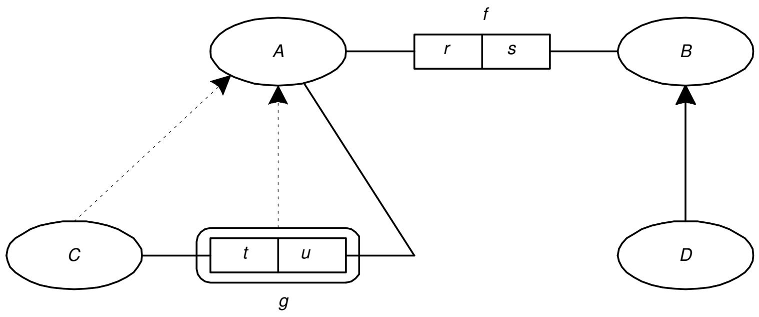


Figure 6: Example Conceptual Schema

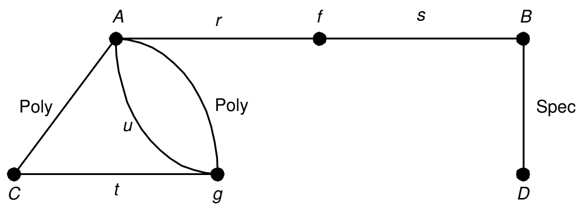


Figure 7: Example Graph

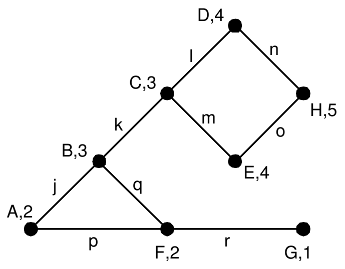


Figure 8: Example graph for a spider query

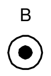


Figure 9: Initial spider query graph

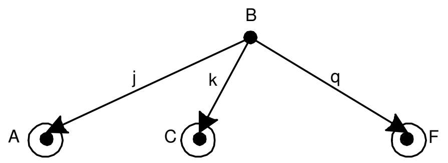


Figure 10: First step in building the spider query graph

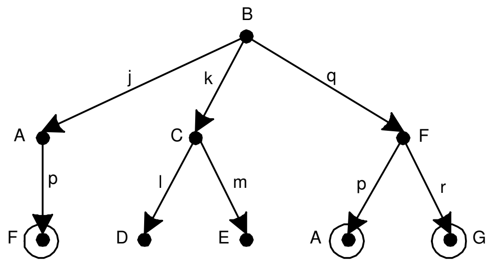


Figure 11: Second step in building the spider query graph

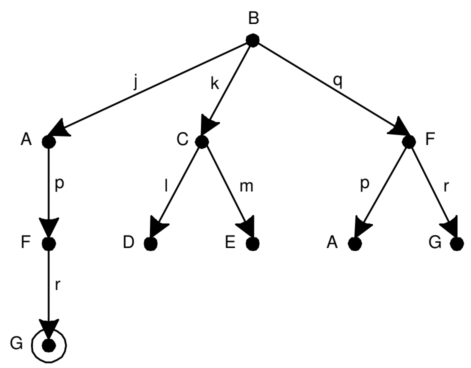


Figure 12: Last step in building the spider query graph