# An Information System organized as Stratified Hypermedia

C.A.J. Burgers[1], H.A. Proper[2] and Th.P. van der Weide[1]

[1]Computing Science Institute, University of Nijmegen
Toernooiveld, NL-6525 ED Nijmegen, The Netherlands

[2]Department of Computer Science, University of Queensland
Queensland 4072, Australia
E.Proper@acm.org

PUBLISHED AS:

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.

**Abstract**

In this paper we investigate the relation between modern hypertext approaches and conventional data modelling techniques, such as PSM. We show how query formulation in a traditional information system can be regarded as a stratified hypermedia featuring two levels of abstraction. The first level of abstraction covers the structure of the stored information, and the second level the information itself.

The investigations provide us with the mechanism of *query by navigation* as a novel avenue for improved query formulation in information systems.

## 1 Introduction

The information disclosure problem begins with persons having an information need they wish to fulfill (figure 1). Formulation of this need leads to a *request* $q$, which has to be matched against the *characterization* of *information objects* which are available in the information base (also referred to as information carriers or documents). This process of *matching* yields a measure for the relevance of objects to the request. The formulation of a request is a known cause of problems ([SM83]).

In information modelling, the analysts and users have to deal with information structures of sizes, matching that of wallpaper (even for medium sized modelling problems). Such structures easily result in diagrams which are unreadable for both domain experts and system analysts, giving them a feeling of being *lost in conceptual space*. If a user wants to know something about objects in this information structure, it is very likely that they have problems in formulating an adequate request. As a consequence, they will retrieve 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]).

A way to overcome the query formulation problem is the concept of *query by navigation*, allowing the user to navigate through the information characterizations, and meanwhile formulating their request interactively. This

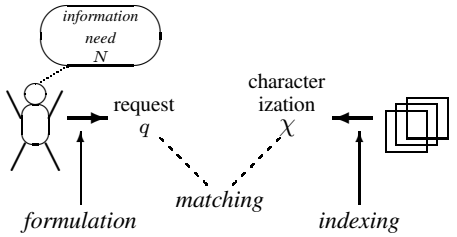


Figure 1: The information disclosure paradigm

interaction mechanism between a searcher and the system is well-known from Information Retrieval, and has proven to be useful ([BW92, Bru93]).

The Predicator Set Modelling technique (PSM) has been introduced as a common denominator for object-role modelling techniques (such as NIAM [NH89]), and ER ([Che76]) based modelling techniques. This makes the presented ideas applicable to a broad range of modelling techniques.

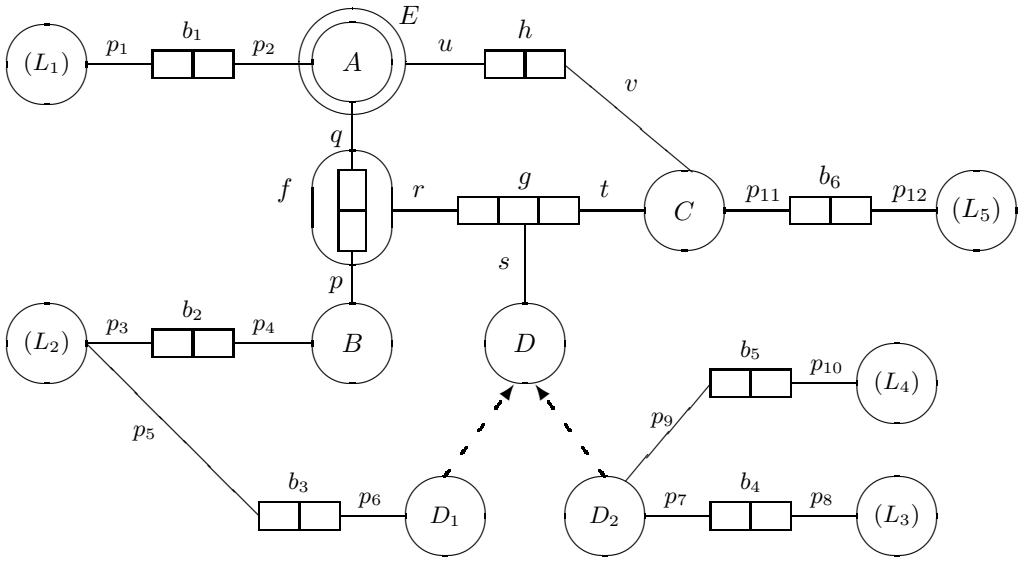


Figure 2: A sample PSM schema

In this paper a PSM information structure is represented as a stratified hypermedia architecture, leading to an implementation of the concept of query by navigation in the context of information modelling. Stratified hypermedia architecture ([BW92]) features a descriptive level, the so-called *hyperindex*, comprising a hypertext of characterizations, indexing the lower level, the so-called *hyperbase*. The hyperbase contains the actual information. Stratified hypermedia architecture offers, by separating description and instantiation, the possibility to formulate information needs by an interactive process of navigation through the descriptive level.

As a result, the hyperindex provides a domain, both for the characterization and query language. The characterization function $\chi$ relates hyperindex and hyperbase, and forms the basis for determining the relevancy (matching) of information objects (hyperbase) to a given query (descriptor from the hyperindex).

Before representing PSM as a stratified hypermedia system, we briefly introduce both PSM and the stratified hypermedia architecture. This paper only provides a brief discussion of the underlying ideas, a more elaborate treatment can be found in chapter 8 of [Pro94], and in [PW95].

## 1.1 Introduction to PSM

The Predicator Set Modelling technique (PSM) [HW93] is a formal model for describing complex data structures. PSM is an extension of the Predicator Model as presented in [BHW91], which was designed as a formalisation of NIAM.

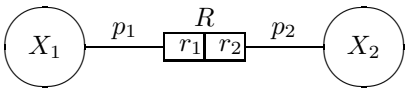


Figure 3: A binary fact type

One of the key concepts in data modelling is the concept of relation type or fact type. In figure 3 the graphical representation of a binary relation $R$ between object types $X_1$ and $X_2$ in the NIAM style is shown. The basic building element of a fact type is the connection between an object type and a role, the so-called *predicator*. In figure 3, $p_1$ is the predicator connecting $X_1$ to $r_1$. In PSM a fact type is considered to be a set of predicators. A relation type is therefore considered as an association between predicators rather than between object types. In PSM, and some versions of ER and NIAM, fact types can also be considered as entity types, this is called *objectification*.

In [HW93] an information structure is defined, over a set of label types $\mathcal{L}$, as a structure consisting of the following basic components:

1. A finite set $\mathcal{P}$ of *predicators*
2. A set $\mathcal{O}$ of *object types*, $\mathcal{L} \subseteq \mathcal{O}$
3. A partition $\mathcal{F}$ of the set $\mathcal{P}$. The elements of $\mathcal{F}$ are called *fact types*
4. A set $\mathcal{G}$ of *power types*
5. A set $\mathcal{S}$ of *sequence types*
6. A set $\mathcal{C}$ of *schema types*
7. A function $\mathsf{Base} : \mathcal{P} \rightarrow \mathcal{O}$. The base of a predicator is the object part of that predicator
8. An auxiliary function $\mathsf{Fact} : \mathcal{P} \rightarrow \mathcal{F}$, defined by: $\mathsf{Fact}(p) = f \Leftrightarrow p \in f$.
9. A function $\mathsf{Elt} : \mathcal{G} \cup \mathcal{S} \rightarrow \mathcal{O}$. This function yields the element type of a power or sequence type
10. A partial order $\mathsf{Spec} \subseteq \mathcal{E} \times \mathcal{O} \setminus \mathcal{L}$ on object types, capturing specialisation
11. A partial order $\mathsf{Gen} \subseteq \mathcal{E} \times \mathcal{O} \setminus \mathcal{L}$ on object types, expressing generalisation. To detect if an object $a$ is generalised $\mathsf{gen}(a)$ is introduced:
$$\mathsf{gen}(a) \Leftrightarrow \exists_{x \in \mathcal{O}}[a \ \mathsf{Gen} \ x]$$

An information structure is now captured formally as: $\mathcal{I} = \langle \mathcal{P}, \mathcal{O}, \mathcal{F}, \mathcal{G}, \mathcal{S}, \mathcal{C}, \mathsf{Gen}, \mathsf{Spec}, \mathsf{Base}, \mathsf{Elt} \rangle$. The set of atomic object types is determined by: $\mathcal{A} = \mathcal{E} \cup \mathcal{L}$. There are two different kinds of atomic object types: *entity types* ($\mathcal{E}$) and *label types* ($\mathcal{L}$). The difference lies in the fact that label types can, in contrast to entity types, be represented (directly) on a communication medium. An instantiation (population) $\mathsf{Pop}$ of an information structure assigns instances to all object types in a consistent way. The instantiation of object type $x$ is denoted as $\mathsf{Pop}(x)$. It is important to note that instances of object types are *not* part of the information structure.

Object types can have values in common in some instantiation, this is formalised in the concept of *type relatedness*. Formally, type relatedness is captured by a binary relation $\sim$ on $\mathcal{O}$. Two objects are type related if and only if this can be proven from the following derivation rules [HPW93]:

1. $\vdash x \sim x$
2. $x \sim y \vdash y \sim x$
3. $x \,\mathsf{Spec}\, y \wedge y \sim z \vdash x \sim z$
4. $x \,\mathsf{Gen}\, y \wedge y \sim z \vdash x \sim z$
5. $x, y \in \mathcal{G} \wedge \mathsf{Elt}(x) \sim \mathsf{Elt}(y) \vdash x \sim y$
6. $x, y \in \mathcal{S} \wedge \mathsf{Elt}(x) \sim \mathsf{Elt}(y) \vdash x \sim y$
7. $\mathcal{I}_x \sim \mathcal{I}_y \vdash x \sim y$

where $\mathcal{I}_x$ denotes the underlying information structure of schema type $x$. The function $\sqcap : \mathcal{O} \to \mathcal{O}$, $\sqcap$ is called the *pater familias* [BHW91]. This function is similar to the top operation from lattice theory.

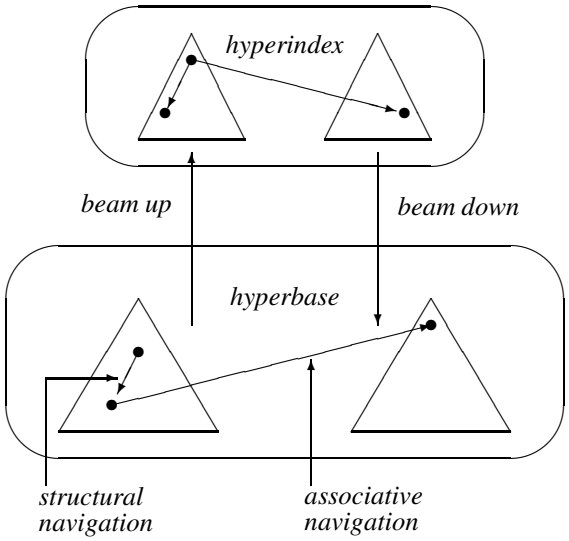


Figure 4: operations of the hypermedia

## 1.2 Introduction to a stratified hypermedia architecture

Stratified hypermedia architecture in its simplest form is a two level hypermedia architecture, as introduced in [BW92]. This architecture features a descriptive level (hyperindex) of indexing information which indexes the lower level, the hyperbase. The hyperbase contains the actual information. An advantage of this architecture is that searchers can navigate within the descriptive level to a description of their information need and then transfer to the lower level via interlayer navigation. This process is called *query by navigation*.

Formally a layer is a structure $\mathsf{L} = (\mathbb{F}, \mathbb{N}, \mathbb{G}, \mathbb{V})$ where

- $\mathbb{F}$ is a set of information fragments. This set is called the *fragment base*.
- $\mathbb{N}$ is a set of presentation units (or nodes). $\mathbb{N}$ is called the *node base*.
- $\mathbb{G}$ is a structure $(\mathbb{E}, \mathbb{P})$, where $\mathbb{E}$ is a set of symbols denoting structural elements, and $\mathbb{P}$ is a set of context-free production rules. $\mathbb{G}$ is referred to as the *schema* of the layer.
- $\mathbb{V}$ is a set of *views*, called the *mask*.

Fragments are the elementary parts of a document, which can not be decomposed structurally into smaller components. Nodes are units of presentation and are used to present the structural elements to the user. Formally, a node is a partially ordered set of fragments. $\mathbb{G}$ is used to structure the information in a layer, and will usually contain a set of context-free grammar rules. In stratified hypermedia architecture, a view is a structure $\mathsf{V} = (\mathbb{S}, \omega, \mathbb{M}, \pi, \mathbb{L})$ where

- $\mathbb{S} \in \mathbb{E}$ is the *start symbol*.
- $\omega$ is a set of parse trees generated from $\mathbb{S}$ using $\mathbb{G}$. $\omega$ is referred to as the *actual structure*.
- $\mathbb{M}$ is the set of vertices within $\omega$. A vertex is also called a *molecule*.
- $\pi : \mathbb{M} \rightarrow \mathbb{N}$ maps each molecule from $\mathbb{M}$ to a presentation unit.
- $\mathbb{L}$ is a set of *associative link* schemata.

In figure 4 two kinds of navigations between molecules are presented. The movement from one molecule to another, using the underlying structure, is called *structural navigation*. Selecting an associative link initiates the traversal of such a link. This is called *associative navigation*. Associative navigation leads to a change in context. Associative links are used to feature cross-references between a fragment in one node, and a fragment in another node.

The basic concept in the interaction with the searcher is a *context*. A context in the hyperindex represents (part of) the information need of the searcher, and corresponds to a path through the information structure. By operating on one context, the searcher will inevitably reach a point where no improvement of this context is possible. The searcher will then put a hold on this context, and commence a new search for the missing part of the information need. During a search searchers will usually have a number of contexts activated, one of which is selected as *focus* for further processing. This set of activated contexts is called the *guide*. The partial information descriptions in the guide can later be combined into a complex query by means of operators such as the set operations, and calculations (sum, average, etc).

# 2 Exploring an information structure

Before formally describing a data modelling technique such as PSM as a stratified hypermedia architecture, we present a demonstration of the benefits of the resulting system. In this paper, we concentrate on query by navigation, and therefore *omit* how this architecture can be helpful during the construction process of information systems. For a description of this latter use, see [HPW92] and [HPW93].

The examples in this section are based on the so-called presidential database, which served as a unified example in a special issue of Computing Surveys [FS76]. The example was first enuntiated in [WBGW73]. An excerpt of the presidential database is shown in figure 5. The examples show how the system supports the formulation of queries. The process of query formulation corresponds to a search through the information system in order

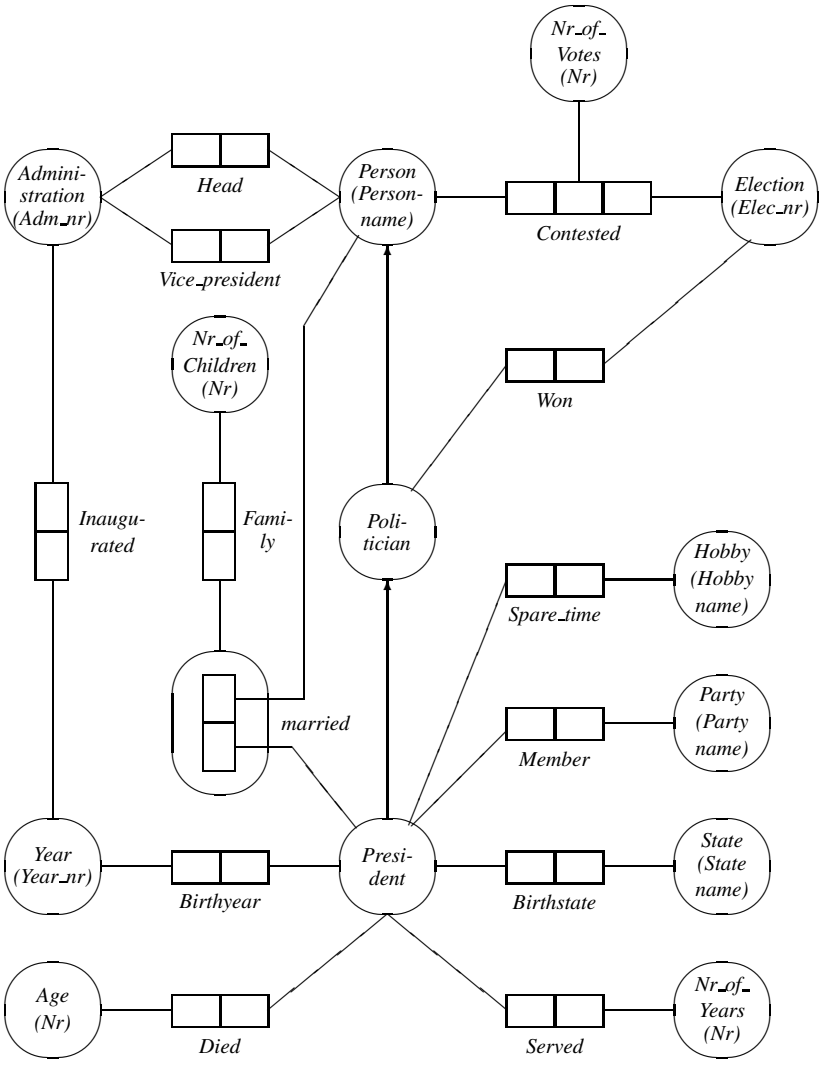


Figure 5: Part of the American president information structure

to fulfill some information need. The request of a searcher is formulated by stepwise refining or enlarging the current description (the *focus*) of this need, until the searcher recognises the current description as sufficiently describing (part of) the information need. Usually the best description will result from a set of such descriptions (the *guide*).

The examples, in this paragraph, give an idea of what a searcher subsequently has to do, and what screens they will encounter, when formulating their information need. Each screen contains, in its header, the current focus, and in its body the direct environment of the current focus.

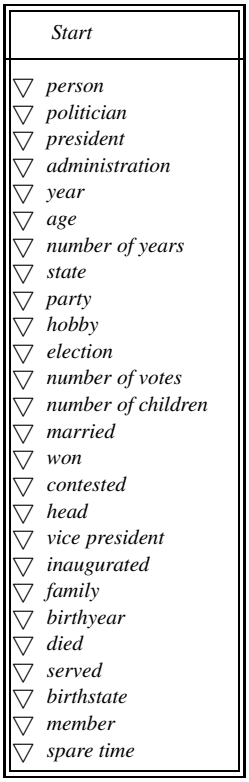


Figure 6: The starting node

The first example, figure 6, shows the screen which corresponds to the starting point of a search in which the searcher has not yet revealed anything. This screen will be referred to as the *starting node* of the system. The starting node contains all object types of the information structure at hand. The searcher can now choose one of the objects as focus for further processing. This selection then is the first refinement of the searcher's information need. A selection of an item in a node is denoted by the symbol $\leftarrow$ in the figures. The symbol $\triangledown$ is a button for a refinement step, while $\triangle$ is used for an enlargement step. Finally $\triangleright$ is a button used for an associative link.

In the second example, figure 7, the searcher wants to find those presidents married with someone involved in politics. The search begins with the starting node. We describe only one path leading to a descriptor of his information need in which the searcher is directly heading for the goal, without any backtracking. The searcher starts with selecting *president* as focus. The associated screen shows about the direct environment of *president*. Thereupon the searcher selects *president is married* as next focus. In the resulting screen the searcher continues with *president is married with person*. The screen associated with this focus shows the sentence *president is married with politician*, which is a proper description of the original information need. Now the searcher can satisfy this information need by requiring the result of the request. The system will present the answer in a

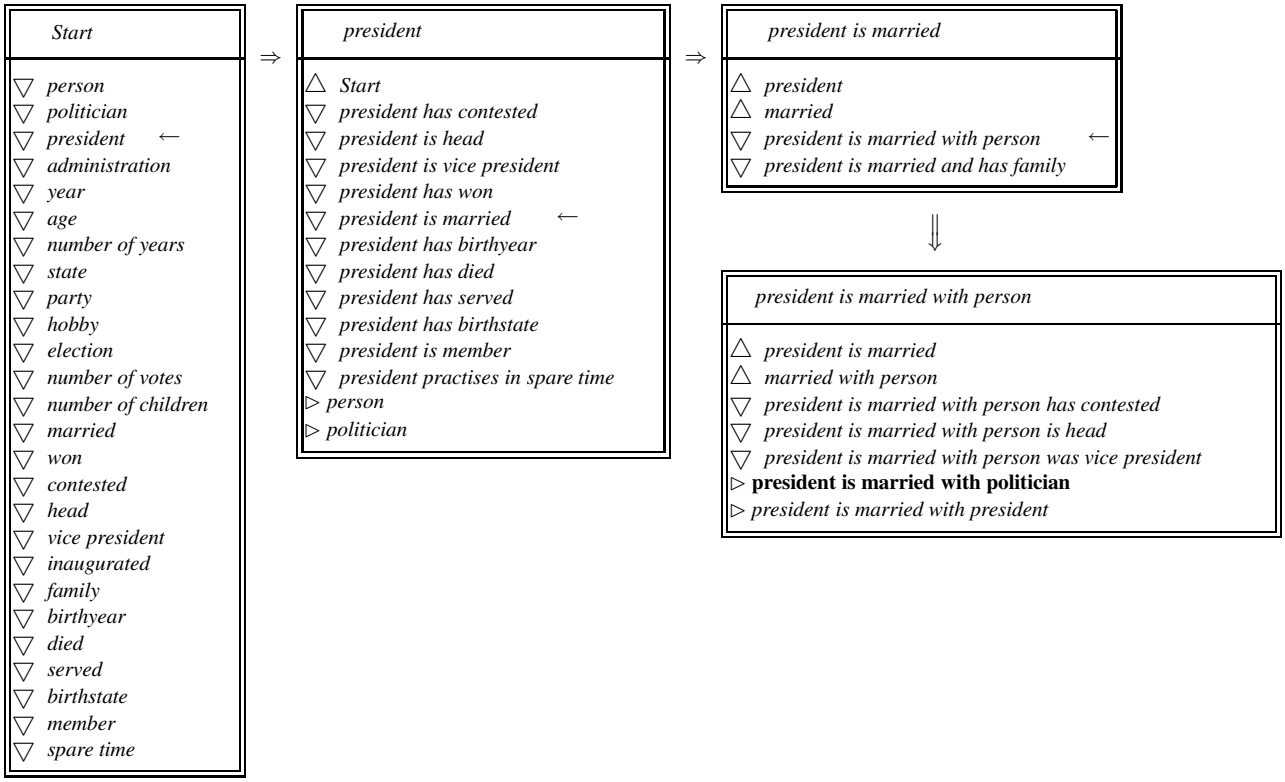


Figure 7: The quest for a president who is married with a politician

standard tabular format, or in the following more verbose form:

president $x$ is married with politician $y$

where $x$ denotes a president and $y$ a politician.

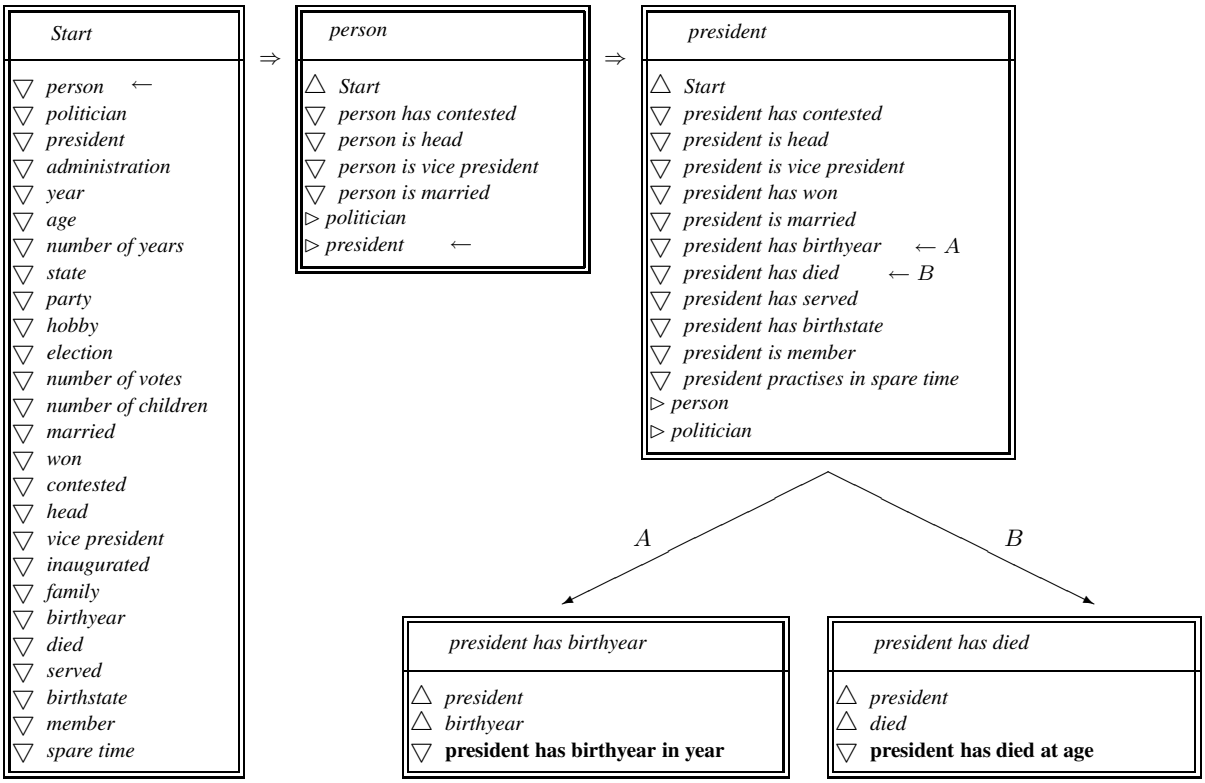


Figure 8: The quest for a persons birthyear and age of death

In the third example, figure 8, the searcher is interested in the birthyear and age of death of some person. In order to find these properties of some persons the searcher first has to choose *president*, as the birthyear and age of death are only recorded for presidents. Now the searcher can refine their focus by choosing either *president has birthyear* or *president has died* as focus for further processing. When selecting the former as focus, using $A$, the sentence *president has birthyear in year* appears. This satisfies the first part of the information need. In order to satisfy the second part, the searcher must create another context. The searcher now focuses, using $B$, on the age of death of persons. After choosing *president has died at age*, the searcher has fulfilled the information need and can require the response by this guide. The system will present all answers in the following verbose way:

president $x$ has birthyear $y$, president $x$ has died at age $z$

where $x$ denotes a president, $y$ a birthyear and $z$ an age of death.

# 3 PSM represented as stratified hypermedia

To represent PSM as a stratified hypermedia architecture, we will derive the hyperindex from the underlying information structures, and construct the hyperbase in terms of the instantiations of the information structure (its population).

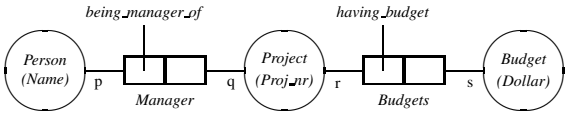


Figure 9: Projects, members and budgets

Object role models allow for the verbalisation of queries by so-called path expressions, built from names of object types and role names. These path expressions are used to construct the hyperindex. For example in figure 9, the expression

Person being_manager_of Project having_budget 10000

describes all persons which are manager of some project with a budget of 10000 dollars. Rather than role names, we will use predicator names. A role name can be seen as a representation mechanism for (binary) fact types. For example, role name being_manager_of corresponds to the representation of fact type Manager by deep structure sentences [NH89] of the form:

$$\langle \mathsf{Person} \rangle \ \text{being\_manager\_of} \ \langle \mathsf{Project} \rangle$$

A role name also corresponds to a connector [HPW93] through fact type Manager. In non-binary fact types, however, connectors can not be uniquely derived from a unique predicator, as predicators in this case does not have a unique co-role. For this reason predicator names are used to uniquely identify paths. Predicators and object types are the elementary parts from which the set of *linear path expressions* ($\mathcal{PE}_{lin}$) is constructed. The hyperindex is constructed from this set $\mathcal{PE}_{lin}$ in the obvious way:

- the elementary paths form the fragments ($\mathbb{F}_i$) of the hyperindex,
- the others form the molecules ($\mathbb{M}_i$).

In the remainder of this section, we describe this in more detail.

## 3.1 Constructing the hyperindex

In [HPW93] the concept of path expression is introduced. A path expression corresponds to a path through the information structure, via predicators, beginning and ending in some object type. In this paper, we restrict ourselves to linear paths. Linear path expressions are constructed from object types $O \in \mathcal{O}$ and predicators $p \in \mathcal{P}$.

Predicator $p$ corresponds to a path from object type $\mathsf{Base}(p)$ to object type $\mathsf{Fact}(p)$. The reverse path is denoted as $p^{\leftarrow}$. Note that a predicator forms a connection between two object types. Linear path expressions are concatenated using the composition operator $\circ$. For instance in figure 2

$$C \circ v \circ h \circ u^{\leftarrow} \circ E$$

is a path from $C$ to $E$.

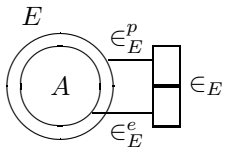


Figure 10: Implicit fact connecting $E$ and $A$

The information structure in figure 2 is represented according to the modelling technique PSM. In this example, object type $E$ is a so-called power type with underlying *element type* $A$. A power type is instantiated with subsets of the (current) instantiation of the underlying element type. The fact type connecting a power type with its element type is usually omitted in the information structure, this fact type is called an implicit fact type (figure 10). In the hyperindex however implicit fact types (and their implicit predicators) are taken into account.

Formally, the hyperindex is introduced as a structure $\mathsf{L}_i = (\mathbb{F}_i, \mathbb{N}_i, \mathbb{G}_i, \mathbb{V}_i)$. Before going deeper into detail, the hyperindex layer is introduced beginning with its fragment base.

**fragment base:** The fragment base ($\mathbb{F}_i$) of the hyperindex contains the elementary parts of the information structure, i.e., its object types ($\mathcal{O}$) and its predicators ($\mathcal{P}$). So we define:

$$\mathbb{F}_i = \mathcal{O} \cup \mathcal{P} \cup \mathcal{P}^{\leftarrow}$$

**node base:** Next we focus on the construction of the hyperindex node base ($\mathbb{N}_i$). Navigation through the information structure corresponds to the construction of a linear path expression. This path expression represents the search process. The associated molecule and its direct environment are presented by a node (figure 11). This environment shows the searcher how they may continue their search, either by enlargement or refinement of the current expression.

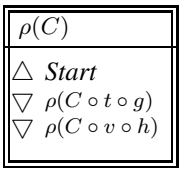


Figure 11: Presentation of $C$ and its direct environment

In this figure, the function $\rho$ is a representation function for linear path expressions in the form of readable sentences (this function will be introduced in section 4). The node contains a header $\rho(C)$, showing the current focus, all possible refinements ($\triangledown$ ) and enlargements ($\triangle$ ).

**schema:** The following step in defining the hyperindex is the introduction of its schema, $\mathbb{G}_i = (\mathbb{E}_i, \mathbb{P}_i)$. We consider $\mathcal{PE}_{lin}$ as the only syntactic category, so:

$$\mathbb{E}_i = \{\mathcal{PE}_{lin}\}$$

The context-free production rules $\mathbb{P}_i$ define the way in which linear path expression can be extended. Suppose $p \in \mathcal{P}$ and $O \in \mathcal{O}$ then the following rules belong to $\mathbb{P}_i$:

$$\begin{array}{lcl} \mathcal{PE}_{lin} & \rightarrow & O \\ \mathcal{PE}_{lin} & \rightarrow & \mathcal{PE}_{lin} \circ p \circ \mathsf{Fact}(p) \\ \mathcal{PE}_{lin} & \rightarrow & \mathcal{PE}_{lin} \circ p^{\leftarrow} \circ \mathsf{Base}(p) \end{array}$$

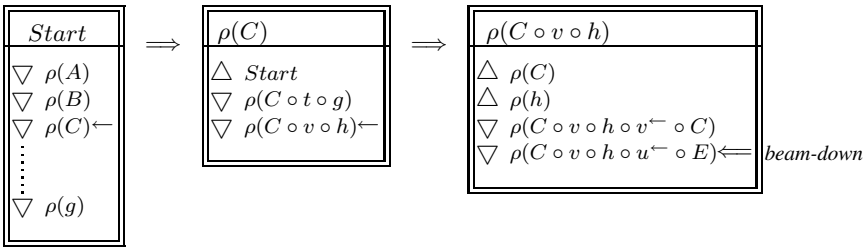


Figure 12: An example hyperindex session

**views:** The hyperindex will contain only one single view. A view is formally introduced as a structure $\mathsf{V}_i = (\mathbb{S}_i, \omega_i, \mathbb{M}_i, \pi_i, \mathbb{L}_i)$. The starting point of this view is $\mathbb{S}_i$. The presentation $\pi(\mathbb{S}_i)$ of this molecule presents all object types $O \in \mathcal{O}$. The molecules $\mathbb{M}_i$ are formed by the composed linear path expressions. We already discussed the presentation of the molecules (figure 11). The actual structure $\omega_i$ then are the linear path expressions that are possible in the actual information structure.

Next the associative links ($\mathbb{L}_i$) of the hyperindex layer are introduced. Traversing an associative link results in a change of context [BW92]. The links from $\mathbb{L}_i$ are used to handle specialisation and generalisation as they occur in the actual information structure. They are defined in the following way:

- if $A$ Spec $B$, then we provide an associative link from each node that contains $A$ (as last occurrence) in its header, to the node in which this $A$ is replaced by $B$. This replacement is effectuated by a special entry ($\triangleright$) in the presentation node.
- if $A$ Gen $B$ and the header contains $B$ as last occurrence, then a special entry is included where $B$ is replaced by $A$

The difference in assignment of the associative links (figure 13) origines from the difference in identification of specialised and generalised objects [HW93].

**Sample sessions in the hyperindex**

At this point the basic concepts of the hyperindex layer are defined. In the remainder of this subsection some examples are given, demonstrating the mechanism of query by navigation.

In a hyperindex session, we use the symbol $\leftarrow$ to denote the path expression that is selected by the user to continue the search process. The symbol $\Longleftarrow$ denotes a beam-down operation (see subsection 3.4).

A possible hyperindex session, for the PSM-schema in figure 2, is shown in figure 12. The example shows the screens that the searcher will subsequently encounter. The session starts with the empty path expression, corresponding to the state in which the searcher has not revealed anything of their interest. The system offers the searcher the opportunity to select one of the object types as an initial point for departure. The symbol $\leftarrow$ shows that the searcher selected object type $C$. So the header of the second node is $\rho(C)$. Now the searcher may choose to step back to a previous state. Or they can refine their information need further, by extending the formed linear path, to reach some other object type. After making another selection, the searcher is satisfied with $\rho(C \circ v \circ h \circ u^{\leftarrow} \circ E)$ as a description of their information need, and decides to retrieve the associated information by performing a beam-down operation.

Two PSM schema's, containing generalisation and specialisation respectively, are shown in figure 13. In order to demonstrate the differences between specialisation and generalisation, figure 13 shows a part of the hyperindex,

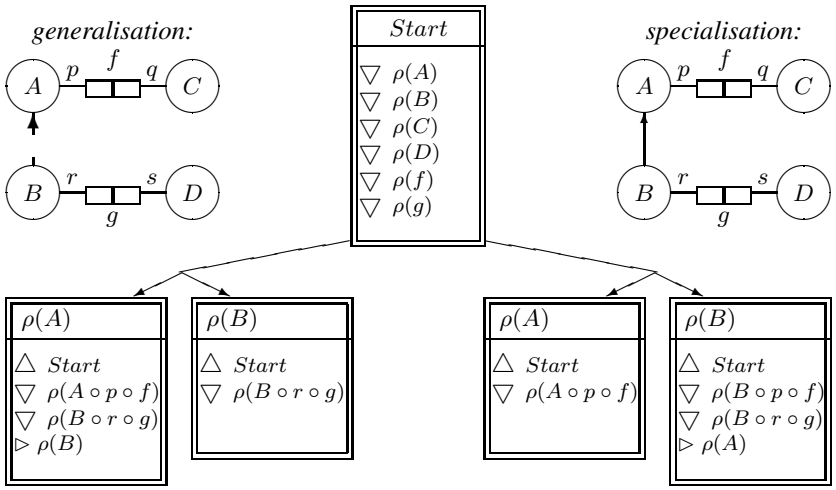


Figure 13: Hyperindex dealing with generalisation and specialisation

located around the starting molecule. From this example, the assignment of the associative links, and the constructed path expressions will become clear. When generalisation is involved, the extension of $A$ with $r \circ g$ results in a substitution of $A$ by $B$. This substitution denotes that instances of $A$, playing $g$, are actually instances of $B$. This substitution origines from the fact that generalised objects inherits all of their properties of its specifiers [HW93].

In specialisation, subtypes inherit their properties from its associated supertype(s) [HW93]. A path expression however is represented by the most specialised name, so in the representation $\rho(B \circ p \circ f)$, $B$ is *not* substituted by $A$.

## 3.2 Constructing the hyperbase

The translation of an instantiation of a PSM information structure into a hyperbase will be done bottom-up. The fragment base ($\mathbb{F}_b$) is defined first, followed respectively by the node base ($\mathbb{N}_b$), the schema ($\mathbb{G}_b$) and the views ($\mathbb{V}_b$).

Instances of abstract object types are composed values. Such instances are represented as structural elements in the hyperbase, thus contributing to the set $\mathbb{M}_b$ of molecules. These, however, are not the only molecules in the hyperbase. For fact types with arity $> 2$, it will be useful to have disposal of all non-empty projections $\wp^+(f)$ of each instance of $f$. This is motivated by the possibility that a path expression can run via a connector through part of a fact type. This will occur when the searcher is not interested in the complete fact instance.

Besides, it will be useful to have disposal of joins of (different) molecules. Two molecules can be joined if they share some sub-molecule. This makes it possible to deal with the construction of linear paths over multiple facts. If a searcher traverses several fact types subsequently, and then performs a beam-down, the hypermedia will return the combination of the associated fact type instances as a result. Note that molecules, formed by projections and joins, will (probably) only be virtually available in the hyperbase.

$$\mathbb{M}_b \quad = \quad \bigcup_{\mathcal{O}\setminus\mathcal{L}} \mathsf{Pop}(x) \;\cup\; \textit{joins of molecules} \;\cup\; \textit{projections of fact type instances}$$

**fragment base:** Label types play a special role in data modelling, as instances of label types, in contrast to other object types, are considered to be directly presentable. Let Repr be the presentation function for label types.

The presentations of instances of the population of label types then form the fragment base ($\mathbb{F}_b$) of the hyperbase. $\mathbb{F}_b$ is defined as:

$$\mathbb{F}_b \quad = \quad \bigcup_{L \in \mathcal{L}} \big\{ \mathsf{Repr}(x) \;\big|\; x \in \mathsf{Pop}(L) \big\} \;\cup\; \big\{ \text{‘,’}, \{, \}, (, ), \langle, \rangle \big\} \;\cup\; \big\{ \omega_i \;\big|\; \omega_i \textit{ occurs in identification rule} \big\}$$

Identification rules are discussed later in this section. The special symbols : $\{, \}$, $(, )$, $\langle, \rangle$ and ‘,’ are included in $\mathbb{F}_b$ for the presentation of object types. For instance the $\{$ and $\}$ are used to present power types. The readability fragments $\big\{ \omega_i \;\big|\; \omega_i \textit{ occurs in identification rule} \big\}$ provide us with the possibility to present molecules in readable sentences.

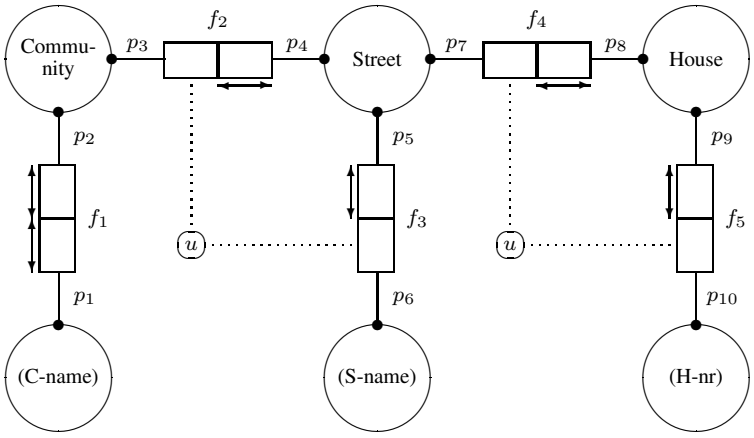


Figure 14: Schema with complex identification

**node base:** The node base consists of a set of nodes. Nodes are used as units of presentation. Using $\pi_b$, each molecule is mapped to a unique node.

According to the *Conformity Rule* [BHW91], an instance of a fact type is a mapping from its predicators into values of the appropriate types. Let $\big\{r : 1026, s : 10000\big\}$ be an instance of fact type Budgets $= \big\{r, s\big\}$, in figure 9. This instance assigns value 1026 to predicator $r$, and value 10000 to predicator $s$. According to our construction, this tuple is a molecule in the hyperbase. In the remainder of this paragraph the presentation of object types instances in readable sentences is discussed. We assume each fact type $f = \big\{p_1, \ldots, p_n\big\}$ has associated a grammar of the following form: $\langle f \rangle \rightarrow \omega_0 \;\langle p_1 \rangle \ldots \omega_{n-1} \;\langle p_n \rangle \;\omega_n$ for readability fragments $\omega_0, \ldots, \omega_n$. These grammar rules denote the structure of the fact type sentences, the so called sentence type [NH89]. For example, fact type Budgets in figure 9 has associated the following rule: $\langle$Budgets$\rangle \rightarrow$ project $\langle r \rangle$ has a budget of $\langle s \rangle$ dollars. Note that $\omega_0 =$ ‘project’, $\omega_1 =$ ‘has a budget of’, while $\omega_2 =$ ‘dollars’. Now let $x \in \mathsf{Pop}(f)$, then this instance is presented by: $\pi_b(x) = \omega_0\, \pi_b(x(p_1)) \ldots \omega_{n-1}\, \pi_b(x(p_n))\, \omega_n$. So the example instance $\big\{r : 1026, s : 10000\big\}$ is presented as: project 1026 has a budget of 10000 dollars. The function $\pi_b$ works recursively when presenting an objectified fact type instance.

The presentation of entity types uses the standard names, as laid down via the identification paths. These standard names denote the way in which abstract entity type instances are presented. For example, an instance of entity type House in figure 14 is presented, using its identification path, as: house $\langle$H-nr$\rangle$ in street $\langle$S-name$\rangle$ in community $\langle$C-name$\rangle$ This because House is identified using H-nr and Street. Street is identified using S-name and Community. Community is identified by C-name. For more details about standard names, see [HPW93] and [Hof93].

The presentation of power types and sequence types can be done according to mathematical conventions. Let $\big\{x_1, \ldots, x_n\big\}$ be an instance of power type $G$, then this instance is presented, using the special symbols $\{, \}$ and

‘,’ as: $\pi_b(\{x_1,\ldots,x_n\}) = \mathsf{ONm}(G)\{\pi_b(x_1),\ldots,\pi_b(x_n)\}$. Note that this presentation requires some order on the elements of power type instances. For an explanation of the function ONm see section 4.

Let $\langle x_1,\ldots,x_n\rangle$ be an instance of sequence type $S$. Then the presentation of $\langle x_1,\ldots,x_n\rangle$, using the special symbols $\langle$, $\rangle$ and ‘,’ yields: $\pi_b(\langle x_1,\ldots,x_n\rangle) = \mathsf{ONm}(S)\langle\pi_b(x_1),\ldots,\pi_b(x_n)\rangle$. An instance $v$ of schema type $C$, consisting of object types $X_1,\ldots,X_n$, is a mapping: $v : \{X_1,\ldots,X_n\} \to \wp(\text{values})$. The presentation ($\pi_b$) of $v$ takes into account that the concept of schema types is not elementary. According to [HW93] and [Hof93], schema types can be defined using the concepts of power type and fact type. So $v$ is presented, using **(**, **)** and **‘,’** as follows:

$$\pi_b(v) \quad = \quad \mathsf{ONm}(C)\mathbf{(}\mathsf{ONm}(X_1) : \pi_b(v(X_1))\mathbf{,}\ldots\mathbf{,}\,\mathsf{ONm}(X_n) : \pi_b(v(X_n))\mathbf{)}$$

where the sets $v(X_i)$ are presented according to the guidelines for instances of power types.

Joins are used to represent combinations of instances, formed by the construction of linear paths over multiple fact types. These molecules are presented by the concatenation of the presentations of the two joined instances. The presentations of these two instances are concatenated using the symbol **‘,’**. For example, in figure 9 the instances $\{p : \text{Johnson}, q : 1026\}$ and $\{r : 1026, s : 10000\}$, can be joined because they share instance 1026. The molecules, representing these instances, are presented as:

$$\begin{aligned}\pi_b\{p : \text{Johnson}, q : 1026\} &= \text{person Johnson is manager of project 1026}\\ \pi_b\{r : 1026, s : 10000\} &= \text{project 1026 has a budget of 10000 dollars}\end{aligned}$$

The presentation of the join of these two molecules yields:

person Johnson is manager of project 1026 **,** project 1026 has a budget of 10000 dollars

The presentation of molecules, representing projections of fact types instances, shows parts of the sentences of the original instance. For example, an instance of a ternary fact type is presented as:

person $\langle x\rangle$ is married with person $\langle y\rangle$ in year $\langle z\rangle$

The molecule representing this instance $\langle x, y, z\rangle$ can be divided into molecules: $\langle x, y\rangle$, $\langle x, z\rangle$ and $\langle y, z\rangle$. These molecules then are presented as:

$$\begin{aligned}\pi_b\,\langle x, y\rangle &= \text{person } \langle x\rangle \text{ is married with person } \langle y\rangle\\ \pi_b\,\langle x, z\rangle &= \text{person } \langle x\rangle \text{ is married in year } \langle z\rangle\\ \pi_b\,\langle y, z\rangle &= \text{person } \langle y\rangle \text{ is married in year } \langle z\rangle\end{aligned}$$

**schema:** The structure of the hyperbase is described by the grammar $\mathbb{G}_b = (\mathbb{E}_b, \mathbb{P}_b)$. From the construction of molecules it will be clear that each non-label object type forms a syntactic category. Besides, projections and joins of fact types are syntactic categories. We will write $\langle x\rangle$ if we want to emphasize the usage of object type $x$ as a syntactic category (both terminal and non-terminal symbol).

The context-free rules ($\mathbb{P}_b$) specifying the structure of these object classes will form, together with the rules specifying projections and joins, the actual structure $\omega_b$ of the hyperbase. These context-free rules represent the way in which instances of object types can be decomposed into smaller structural elements.

The entity types form the first class of molecules in the hyperbase. The instances of entity types can not be decomposed into smaller structural elements. So these molecules can be seen as terminal symbols.

Next we look at the instances of fact types. A fact type $f = \{p_1,\ldots,p_n\}$ corresponds to the syntactic category $\langle\{p_1,\ldots,p_n\}\rangle$. For syntactic categories over sets of predicators we introduce the following context-free production rule:

$$\langle\{p_1,\ldots,p_n\}\rangle \to \langle\{p_1,\ldots,p_n\} - \{p_i\}\rangle\,,\ \text{for each } p_i, 1 < i \leq n$$

If the set of predicators contains only one element, the production rule yields:

$$\langle \{p\} \rangle \rightarrow \langle \mathsf{Base}(p) \rangle$$

Note that these two rules also define the projections of a fact type instance.

The instances of power types and sequence types are formed by composing sets and tuples of the instances of their element type respectively. Because these two are structurally the same, assuming some arbitrary order on the elements of power type instances, the context-free rule yields for both: $\langle X \rangle \rightarrow \langle \mathsf{Elt}(X) \rangle^*$. Construction of the production rules dealing with instances of a schema type $\langle C \rangle$ is less obvious. Suppose schema type $C$ is decomposed into the object types $x_1, \ldots, x_k$. The first production rule for $C$ yields the domain of populations of $C$: $\langle C \rangle \rightarrow \langle \overline{x_1} \rangle \ldots \langle \overline{x_k} \rangle$. The range of the instantiantion of object type $x_i$ is further refined by the rule: $\langle \overline{x_i} \rangle \rightarrow \langle x_i \rangle^*$. There are three production rules dealing with joins of molecules. The first production rule yields the join of two fact type instances. The second one yields the join of two already joined molecules. The third production rule yields the join of a joined molecule with a fact type instance. Note that this production rule is only applicable if that fact type instance has an arity larger then two. This results in the following three production rules: $\langle J \rangle \rightarrow \langle F_1 \rangle \langle F_2 \rangle$, $\langle J \rangle \rightarrow \langle J_1 \rangle \langle J_2 \rangle$, and $\langle J \rangle \rightarrow \langle J_1 \rangle \langle F_1 \rangle$, where $F_i$'s and $J_i$'s molecules represent fact type instances and joined molecules respectively.

**view:** Thus far, all parts of the hyperbase view have been discussed, except for associative links. In the minimal setting, as introduced here, we will omit associative links.

## 3.3 Characterization

In this section, we relate hyperbase and hyperindex by the introduction of a characterization $\chi$. This function maps information objects (fragments, nodes, molecules) from the hyperbase onto the fragments of the hyperindex [BW92].

The characterization function will be defined bottom up, starting with the fragments of the $\mathbb{F}_b$. Fragment $y \in \mathbb{F}_b$ is characterized by all entity types that are associated to $y$ via a bridge type ($\mathcal{B}$):

$$\chi(y) \quad = \quad \{X \mid \exists_{x \in \mathsf{Pop}(X), f=\{p,q\} \in \mathcal{B}} [\langle p\!:\!x, q\!:\!y \rangle \in \mathsf{Pop}(f)]\}$$

As a corollary, note that subtyping and generalisation is covered by this rule as follows:

- $X \; \mathsf{Spec} \; Y \wedge X \in \chi(x) \Rightarrow Y \in \chi(x)$
- $X \; \mathsf{Gen} \; Y \wedge Y \in \chi(x) \Rightarrow X \in \chi(x)$

The characterization of node $N = \omega_0 \ldots \omega_n$ is defined in terms of fragments it contains. If, however, $N$ presents a molecule containing an instance of a power type, a sequence type or a schema type then the characterization has to deal with it differently. This because these object types are presented using fragments who are concretisations of their elementary types. The characterization of a node $N$ can be defined using the following four rule. The characterization of a node containing an instance of a power type $G$ yields: $\chi(\sigma \; \pi_b(\{x_1, \ldots, x_n\})\tau) = \chi(\sigma) \cup \{G\} \cup \chi(\tau)$. If a node contains an instance of a sequence type $S$ then the characterization yields: $\chi(\sigma \; \pi_b(\langle x_1, \ldots, x_n \rangle)\tau) = \chi(\sigma) \cup \{S\} \cup \chi(\tau)$. The characterization of a node containing an instance of a schema type $C$ yields: $\chi(\sigma \; \pi_b(v)\tau) = \chi(\sigma) \cup \{C\} \cup \chi(\tau)$. In all the other cases the characterization of a node yields: $\chi(N) = \bigcup_{f \in N} \chi(f)$. For instance, the characterization of

project 1026 has a budget of 10000 dollars yields $\{$Project, Budget$\}$, because:

$$\begin{aligned}
\chi(\text{`project'}) &= \varnothing \\
\chi(1026) &= \{\text{Project}\} \\
\chi(\text{`has a budget of'}) &= \varnothing \\
\chi(10000) &= \{\text{Budget}\} \\
\chi(\text{`dollars'}) &= \varnothing
\end{aligned}$$

Note that the structure of the presentation of a molecule is not taken into account. The characterization of molecule $x$, representing an instance of object type $X$ is found in two steps. Firstly, the characterization of the presentation $\pi_b(x)$ is taken into account: $\chi_w(x) = \chi(\pi_b(x))$. This is known as the weak characterization of $x$. Secondly, the properties of $x$ are included (strong characterization). These properties are found by the predicators based on $X$. $\chi_s(x) = \{p \in \mathcal{P} \mid t(p) = X \vee p \in X\}$, where $t$ is an instance of some fact type $f$. Molecules representing joins are characterized by the union of the characterizations of their sub-molecules.

## 3.4 The beam-down operation

Now that the characterization of the hyperbase molecules is completed, we focus on the beam-down operation. The beam-down operation transfers the searcher from an expression in the hyperindex to the result of that expression, i.e. molecule(s) in the hyperbase. The hyperbase molecules are characterized using the fragments within the hyperindex ($\mathbb{F}_i$), so, in order to establish a beam-down result, the molecules of the hyperindex ($\mathbb{M}_i$) must also be described using these fragments.

The function *dissect* (Di) yields for each linear path expression its associated set of fragments:

$$\mathsf{Di} : \mathcal{PE}_{lin} \rightarrow \wp(\mathbb{F}_i)$$

The function dissect is defined inductively according to the structure of linear path expressions.

- An object $X$ is dissected in its own name and the entity types on its identification path. The entity types on an identification path of object type $X$ is denoted by $\mathsf{Id}(X)$. A fact type $X$ however is dissected in its predicators and the bases of those predicators. If a fact type $X$ is connected to an objectified fact type $Y$ then the dissection of $Y$ is also included in the dissection of $X$, so:

$$\mathsf{Di}(X) = \begin{cases} \bigcup\limits_{p \in X} \big( \{p\} \cup \mathsf{Di}(\mathsf{Base}(p)) \big) & \text{if } X \text{ is a fact type} \\ \{X\} \cup \bigcup\limits_{y \in \mathsf{Id}(X)} \{y\} & \text{otherwise} \end{cases}$$

- The dissection of a linear path must deal with the possibility that this path contains objectified fact types (used as entity types). If some path $Y$ contains an objectified fact type $Z$, used as an entity type, then the dissection of $Y$ includes the dissection of $Z$ also, so:

$$\mathsf{Di}(P \circ p \circ \mathsf{Fact}(p)) = \begin{cases} \mathsf{Di}(P) \cup \{p\} \cup \mathsf{Di}(\mathsf{Base}(p)) & \text{if } \mathsf{Base}(p) \in \mathcal{F} \\ \mathsf{Di}(P) \cup \{p\} & \text{otherwise} \end{cases}$$

- If a linear path $P$ is extended with a predicator $p$ and an object type $\mathsf{Base}(p)$, then $p^{\leftarrow}$ and the dissection of $\mathsf{Base}(p)$ are added to the dissection of $P$.

  Because there is no difference between connecting two object types using a $p$ or $p^{\leftarrow}$, $p^{\leftarrow}$ can be substituted (in the result) by $p$, so:

$$\mathsf{Di}(P \circ p^{\leftarrow} \circ \mathsf{Base}(p)) = \mathsf{Di}(P) \cup \{p\} \cup \mathsf{Di}(\mathsf{Base}(p))$$

The BeamDown operator, on some guide $G$, retrieves hyperbase molecules in the following way. The function Rel calculates the relevancy of hyperbase molecules for the actual guide. This function yields a pair as a result, the first component denoting the relevancy of the object types from the dissection of the guide, the second that of the predicates. Only hyperbase molecules that exceed some treshold $\epsilon$ will be retrieved, and are presented in their (alphabetic) order of relevance.

$$\mathsf{BeamDown}(G) = \left\{ M \in \mathbb{M}_b \;\middle|\; \mathsf{Rel}(M, G) > \epsilon \right\}$$

The calculation of the relevancy of some hyperbase molecule $M$ for some guide $G$ yields:

$$\mathsf{Rel}(M, G) \quad = \quad (\text{relevance object types},\ \text{relevance predicators})$$

The calculation of the relevance of the object types takes type relatedness into account. For each object type $X$, in some characterization or dissection, only $\sqcap(X)$ is regarded relevant for the calculation. This leads to the following rule:

$$\text{relevance object types} \quad = \quad \frac{\left| \left( \mathsf{Cr}(M) \ \cap \ \mathsf{Pr}(G) \right) \ \cup \ \mathsf{Corr}(M) \right|}{\left| \mathsf{Cr}(M) \ \cup \ \mathsf{Pr}(G) \right|}$$

The function Cr is based on the weak characterization (containing object types):

$$\mathsf{Cr}(M) = \sqcap(\chi_w(M))$$

Similarly, the function Pr yields the set containing the pater familias of each object type from the dissection of $G$:

$$\mathsf{Pr}(G) = \sqcap \left( \bigcup_{p \in G} \mathsf{Di}(p) \cap \mathcal{O} \right)$$

The function Corr corrects for noise, resulting from multiple entity type instances being represented by a single label type instance. For example, in figure 2 object types $B$ and $D_1$ are both identified by label type $L_2$.

$$\mathsf{Corr}(M) = \left\{ X \;\middle|\; \exists_{x \in \mathsf{Pop}(X) \wedge y \in \pi_b(M)} [\mathsf{Concr}(x) = y] \right\}$$

where $\mathsf{Concr}(x)$ denotes the concrete value used for representing abstract instance $x$.

The calculation of the relevance of the predicators is done analogously, with the exception that no noise occurs in this case.

$$\text{relevance predicators} = \frac{\left| \mathsf{Cp}(M) \ \cap \ \mathsf{Pp}(G) \right|}{\left| \mathsf{Cp}(M) \ \cup \ \mathsf{Pp}(G) \right|}$$

The function Cp is defined by:

$$\mathsf{Cp}(M) = \chi_s(M)$$

The predicators, associated with the dissection of a guide, are found by:

$$\mathsf{Pp}(G) = \bigcup_{p \in G} \mathsf{Di}(p) \cap \mathcal{P}$$

# 4 The presentation function $\rho$

In this section the function $\rho$ is introduced. This function maps linear path expressions into readable sentences. Before defining $\rho$ we introduce the concepts of *object type naming* and *predicator naming*.

## 4.1 Object type and predicator naming

In order to verbalise linear path expressions, we must first verbalise object types and predicators. Explicit object types are referenced by a unique name: $\mathsf{ONm} : \mathcal{O} \rightarrow \mathcal{N}_s$. Explicit predicators may have assigned a so-called *predicator name* via the (partial) function: $\mathsf{PNm} : \mathcal{P} \cup \mathcal{P}^{\leftarrow} \rightarrow \mathcal{N}_s$. These naming functions, however, do not assign names for implicit object types and implicit predicators. For the latter categories, a general naming mechanism (using keywords) is introduced. For instance in figure 9, the predicator $q$ could be verbalised in two directions as follows:

$$\begin{aligned} \mathsf{PNm}(q) &= \text{has as} \\ \mathsf{PNm}(q^{\leftarrow}) &= \text{of} \end{aligned}$$

The predicator name of a predicator may yield an empty string. The motivation is that an empty string sometimes enables us to present better readable sentences.

## 4.2 Verbalising linear paths

Linear path expressions are presented as readable sentences via the function: $\rho : \mathcal{PE}_{lin} \rightarrow \mathcal{N}_s$. This function is defined recursively as follows:

$$\rho(O) = \mathsf{ONm}(O)$$

$$\rho(P \circ p \circ X) = \begin{cases} \rho(P)\ \mathsf{PNm}(p)\ \mathsf{ONm}(X) & \text{if } p \text{ and } X \text{ have names} \\ \rho(P)\ \textit{involved in}\ \mathsf{ONm}(X) & \text{if } p \text{ has no, and } X \text{ has a name} \\ \text{some keyword, see below} & \text{if both } p \text{ and } X \text{ have no names} \end{cases}$$

$$\rho(P \circ p^{\leftarrow} \circ X) = \begin{cases} \rho(P)\ \mathsf{PNm}(p^{\leftarrow})\ \mathsf{ONm}(X) & \text{if } p^{\leftarrow} \text{ and } X \text{ have names} \\ \rho(P)\ \textit{of}\ \mathsf{ONm}(X) & \text{if } p^{\leftarrow} \text{ has no, and } X \text{ has a name} \\ \text{some keyword} & \text{if both } p^{\leftarrow} \text{ and } X \text{ have no names} \end{cases}$$

For instance in figure 9:

$$\begin{aligned} \rho(\text{Project} \circ q \circ \text{Manager}) &= \mathsf{ONm}(\text{Project})\ \mathsf{PNm}(q)\ \mathsf{ONm}(\text{Manager}) \\ &= \text{project has as manager} \end{aligned}$$

Now we will introduce keywords. Keywords serve as a naming mechanism for handling implicit object types and predicators. In figures 15, 16, 17, and 18 the keywords are summarised. For each keyword an arrow denotes the path associated with that keyword. (For more information about keywords see [HPW93].)

These keywords also can be used when no predicator names are defined. For instance, if in figure 9 $\mathsf{PNm}(q^{\leftarrow})$ is not defined:

$$\rho(\text{Manager} \circ q^{\leftarrow} \circ \text{Project}) = \text{manager } \textit{of} \text{ project}$$

When dealing with implicit facts it can be useful to use keywords instead of defining predicator names and the name of the implicit fact. If in figure 16:

$$\begin{aligned} \mathsf{ONm}(B) &= \text{convoy} \\ \mathsf{ONm}(A) &= \text{ship} \end{aligned}$$

Then the denotation of the following linear paths yields:

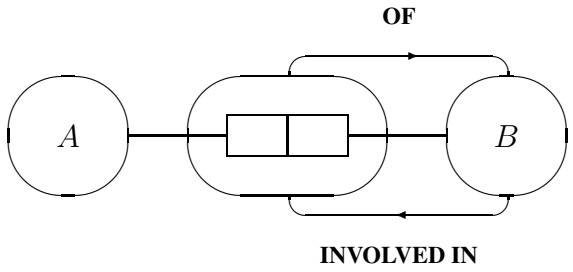


Figure 15: Keywords used in fact verbalisation

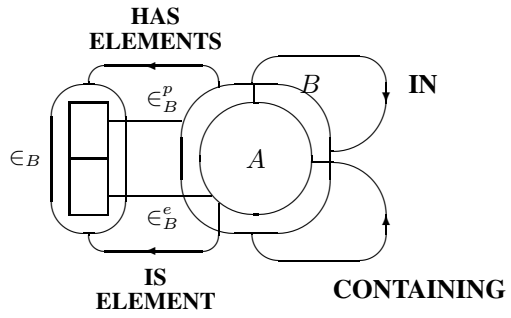


Figure 16: Keywords used in power type verbalisation

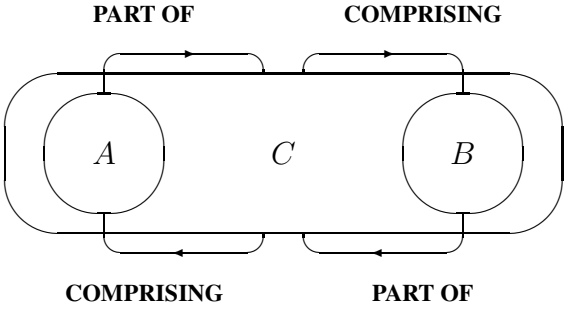


Figure 17: Keywords used in schema type verbalisation

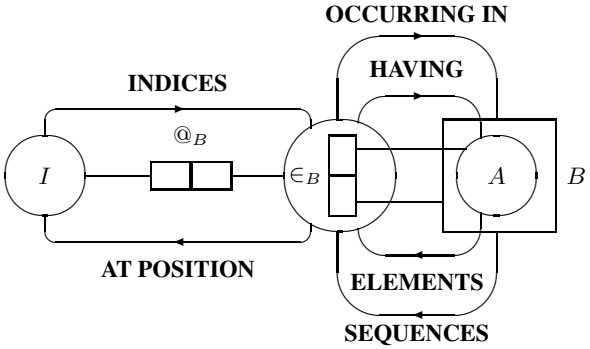


Figure 18: Keywords used in sequence type verbalisation

$$\rho(B \circ \in_B^p \circ \in_B) = \text{convoy } \textit{has elements}$$
$$\rho(A \circ \in_B^e \circ \in_B \circ \in_B^{p\leftarrow} \circ B) = \text{ship } \textit{in} \text{ convoy}$$

# 5 Conclusions

In this paper we investigated an integration between stratified hypermedia architecture and information systems. This integration makes it possible to use the concept of query by navigation in information structures based on object-role modelling techniques. Due to this integration a user can interactively formulate a query.

The presented mechanism offers several opportunities for further refinement. Firstly, at the moment a guide connects the comprising linear path expressions like conjunction for boolean expressions. It might be worthwile to investigate the effects of an extension to full boolean retrieval for combining linear path expressions. Finally, the query by navigation mechanism should be extended with supporting mechanisms that allow for the formulation of more complex queries.

# References


[BHW91] P. van Bommel, A.H.M. ter Hofstede, and Th.P. van der Weide. Semantics and verification of object-role models. *Information Systems*, 16(5):471–495, October 1991.

[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.

[Che76] P.P. Chen. The entity-relationship model: Towards a unified view of data. *ACM Transactions on Database Systems*, 1(1):9–36, March 1976.

[FS76] J.P. Fry and E.H. Sibley. Evolution of Data-Base Management Systems. *Computing Surveys*, 8(1):7–42, 1976.

[Hof93] A.H.M. ter Hofstede. *Information Modelling in Data Intensive Domains*. PhD thesis, University of Nijmegen, Nijmegen, The Netherlands, 1993.

[HPW92] A.H.M. ter Hofstede, H.A. Proper, and Th.P. van der Weide. Data Modelling in Complex Application Domains. In P. Loucopoulos, editor, *Proceedings of the Fourth International Conference CAiSE'92 on Advanced Information Systems Engineering*, volume 593 of *Lecture Notes in Computer Science*, pages 364–377, Manchester, United Kingdom, EU, May 1992. Springer Verlag, Berlin, Germany, EU. ISBN 3540554815

[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.

[HW93] A.H.M. ter Hofstede and Th.P. van der Weide. Expressiveness in conceptual data modelling. *Data & Knowledge Engineering*, 10(1):65–100, February 1993.

[NH89] G.M. Nijssen and T.A. Halpin. *Conceptual Schema and Relational Database Design: a fact oriented approach*. Prentice-Hall, Sydney, Australia, 1989. ASIN 0131672630

[Pro94] H.A. Proper. *A Theory for Conceptual Modelling of Evolving Application Domains*. PhD thesis, University of Nijmegen, Nijmegen, The Netherlands, EU, 1994. ISBN 909006849X

[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.

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

[WBGW73] S.E. Willner, A.E. Bandurski, W.C. Gorhan, and M.A. Wallace. COMRADE data management system. In *Proceedings of the AFIPS National Computer Conference*, pages 339–345, Montvale, New Jersey, 1973. AFIPS Press.