

# Enforcing Strong Object Typing in Flexible Hypermedia

*Pedro Furtado and H. Madeira*

Depto Eng. Informática Univ. Coimbra  
Pinhal Marrocos  
3030 Coimbra, Portugal  
pnf@dei.uc.pt

## ABSTRACT

The presentation layer of hypermedia systems could benefit from standard object querying functionality and this is most effective if strong typing is enforced. By strong typing we mean the direct representation of data semantics as object types in an object database as opposed to a “slotted frames” representation. On the other hand, the flexible emergent nature of structure must be considered in the authoring activity and in this sense premature typing and organizing is counterproductive. Reflecting on these apparently contradictory issues and the past proposals to handle the problem, we extend the strongly typed data model of a prototype hypermedia system, WorldView, to support semi-automatic object submission and type metamorphosis. Weak types are also necessary for some constructs, so they coexist with strong types, but these are enforced. We emphasize the benefits available to the presentation layer of keeping a uniform object oriented structure. In particular we implement a dynamic linking capability that uses queries to retrieve the objects related to some object attribute and suggest other improvements. We stress that most object oriented hypermedia systems are frame-based, especially in what concerns user-defined and emergent structure.

**KEYWORDS:** Hypermedia, Flexibility, Knowledge Structuring, Emergent Structures, Frame Model, OODBMS.

## 1 INTRODUCTION

In information gathering users collect, structure, taxonomize, index and query information items which are then useful for all kinds of analysis, measurements and drawing conclusions. The raw information may be directly entered by the user, gathered from the web, a file or some other device. It may be a body of information on something or a large quantity of scientific data. There is no predefined data structure and the user directs the system to emerge a certain structure from the input data based on visual and other clues. The user directs the system to build a schema from bottom up instead of the traditional top-down database design approach [23].

If a structured representation isn't pursued, the subsequent activities lack efficiency. Taxonomizing and indexing will be manual (impossible for large quantities of data), querying will not be possible except for full text searches and alike. These problems in turn hinder information analysis. The traditional hypertext <node, anchor, link> approach is not suited for this kind of structuring. It offers a great deal of freedom but little structure. A hypermedia system with an underlying object hyperbase is the appropriate tool to maintain this kind of structuring. Once the structured representation is achieved the taxonomizing, indexing and analysing activities can be supported by simply querying the database. Of course, directly querying a database requires the use of a non-intuitive language such as object query language (OQL) which is far less intuitive than clicking on an hyperlink. Therefore, it is the presentation layer that must use the query language as a powerful tool to implement dynamic linking and other possible features. On other words, query capabilities give the presentation layer the ability to work with objects based on their contents. The user is offered a graphical means to express the queries.

Our investigation focused on how to provide type emergent features that are easy to use and automate the process for repetitive data. But in contrast with previous work on this issue (Viki [10] [11], Dolphin[5], Cowfish [21]) we do not settle for slotted frames of information. Instead, the user is encouraged to create object types in an intuitive way. We are especially concerned with enforcing strong typing to achieve as much as possible database integration of the emergent structures. This way, we outline the path from unstructured data to formally structured one and set the basis for type metamorphosis. We also show that this optional strong typing brings useful full object query language functionality to the objects.

At this point we would like to point out that different hypermedia applications have different specific requirements. Dolphin [5] is targeted at conducting group meetings and therefore emphasizes the free expression of ideas. Viki [10] supports information gathering and interpreting. It explores spatial structures and automatic structure-finding algorithms to help the user organizing the materials. The applications envisioned in those systems did not stress the same needs that we stress here, that is, the necessity to query the information after at least a reasonably large body has been assembled, the importance of strong structure for that and how emergent structure can become

strong. This is the main contribution of our research here.

This paper is organized as follows: Section 2 situates this work in the research field. Section 3 presents the basic user interface and model of the WorldView system (WV) and also the automatic object submission capability. Section 4 describes the emergence, metamorphosis and type querying functionality with the help of an example and Section 5 concludes the paper.

## 2 INSIGHT INTO INFORMATION STRUCTURING

Information structuring and representation is discussed in a way that is related to our work in [10] and [5]. In [10] systems were classified as permissive, emergent, descriptive or prescriptive. Prescriptive systems constrain the user to a particular model or methodology. Permissive systems were the first tendency in the research field and allowed unconstrained linking but lacked structuring capabilities (ways of expressing regularities, abstractions or assumptions). After these systems, the research community turned to typed nodes and links, allowing a greater expression of semantics. Those are descriptive or meta-schematic approaches in which the user can express and manipulate abstractions and their instances. This is the case of MacWeb [14], Aquanet [7], etc. The underlying representation relied on slotted frames, where an object is a container with a type attribute and a set of slots which may also include type attributes. Experiences with Aquanet [8] report that users prefer informal spatial layout-based information relating to using more formal relations and premature structuring was counterproductive. This was further explored in [9]. The idea was that more flexible and informal knowledge organization was needed. This was achieved in Viki [10] which, besides giving a lot of flexibility to the user organizing the content holding entities, supports structure emergence and also some degree of structure recognition. Similarly, Dolphin [5] also supports structure emergence, the coexistence of different degrees of formality and mutual transformation between them. Cowfish [21] extends Sepia activity spaces [18] into flexible meta-model activity spaces in which the user flexibly specifies types. These systems also use a frame-based approach: Sepia uses a frame model with object oriented frame representations similar to Aquanet or MacWeb [13]. Dolphin [5] uses nodes as collections of atomic objects and other nodes (typed frames could be imported from Sepia).

Frames are generic containers with an arbitrary number of data slots. Typed frames with typed slots restrict the information that can be put into the slots. On the contrary, an object type defines a specific structure (and behavior) and benefits directly from standard object manipulation languages like OQL for associative retrieval. Navigation by following hard-links is easy in frame-based systems, but associative data retrieval is not standard.

The importance of supporting strong object typing (instead of frame typing) in application tailoring of extensible hypermedia engines is emphasized in HyperStorm [1] and in the hypermedia document database application in [22]. It would also be important to support strong object typing for

user-defined and emergent structures. We show that by strong typing user-defined and emergent structures over an object database we are able to use the object querying functionalities of standard OQL in a way that would not be possible with a frame system.

In [5], hypermedia systems are compared with respect to two dimensions: the degree in which the user identifies object types and the degree in which the system represents those object types internally. We claim that our system improves the degree of system internal representation of object types as exposed before. Figure 1 shows this in the context of [5]. In addition to the weakly-typed slotted-frames representation, our system encourages the user to specify strong object type representations using intuitive visual tools and semi-automated submission. As a result, our system covers the region in the ellipsis, but emphasizes the black spot extremity with a high degree of system representation.

We show that this way the hypermedia engine strongly-typed object manipulation facilities become directly available.

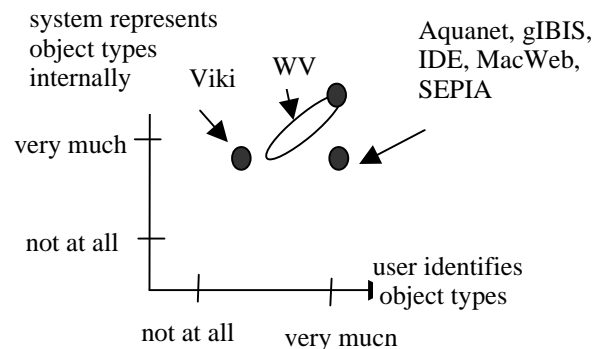


Figure 1 – Typing in WV and other systems

We will now discuss the argumentation for adopting a frame system instead of an object database. OVID [15] is a prototype video database system implemented on a Macintosh with HyperCard, coded in HyperTalk and including a video query language VQL. In [16] the authors argue that 1) a conventional database schema could not be defined for the application because attributes vary a lot in number and depending on the user perspective. 2) Users must describe the complete definition of attributes and their values. 3) Querying is also difficult because users would have to explore the object structure and objects vary a lot. This would mean that 4) frame systems are better suited for schemaless databases such as OVID.

We confront those arguments with ours. (1) We think there is a spectrum of types in what concerns object structure stability: This ranges from collections and frames to strongly structured objects. In a later section we show an example of structure emergence that could yield a large collection of regular structured objects of one type. The presentation layer can easily offer dynamic linking, indexing and information organizing capabilities for these new objects just by using OQL thanks to their object structure.

So, our system represents all the spectrum of objects, including frames. (2) Our system includes automatic object submission, which solves this difficulty. (3) When attributes are not completely known because they vary a lot, structure exploration is always necessary. This must be done in OVID as in any other system. We note that VQL is custom coded and the application is tailored for video data. The same VQL functionality could be implemented as an extension to OQL and the application could be accessing an OODBMS. There are interesting proposals for query language extensions to incomplete knowledge [2] which effectively automate the structure exploration. (4) Object databases represent frames easily and present other advantages because of the more strict object representation, namely, enforcing object structure when possible, standard OQL, scalability and concurrent access, transaction support, etc .

Another issue that we view as related but do not directly investigate here is indexing the emerged structured data for information organization, access and navigation. This is an important issue for our future work. We believe that the strong typing we enforce and the possibility to use the query language facilitates this task. One example is categorizing (taxonomizing). A simple OQL query with a group clause can do this on strongly typed data! Examples of interesting indexing schemes are the semantic indexes in [3], the information organizing axis with the mapping function in the Frame-Axis model [12] or the knowledge layer and semantic walk expressions in MacWeb [14].

### 3 OVERVIEW OF WORLDVIEW

WorldView (WV) is a research project in knowledge structuring and hypermedia tools for information merging and sharing. The base user interface is structured as a collection of nested folders to guide users organizing the information they insert and comprehensive user permissions and object privilege features to allow them to control the access in a cooperative environment. Our system has a specific data model that will be presented shortly after and supports the hypermedia-like linking capabilities. Flexibility was not our main concern in the first prototype. Still, it was recognized that a flexible cooperative hypermedia system must support user-defined information. For this reason the system is extended to support first semi-automatic user submission of objects and relationships and emergent structures from information gathered from user workspaces, files, the web or any other source.

#### Basic User Access Interface

The basic user access interface of WV is based on folders containing the information items. The system is characterized by having accounts and security features. Figure 2 shows the interface.

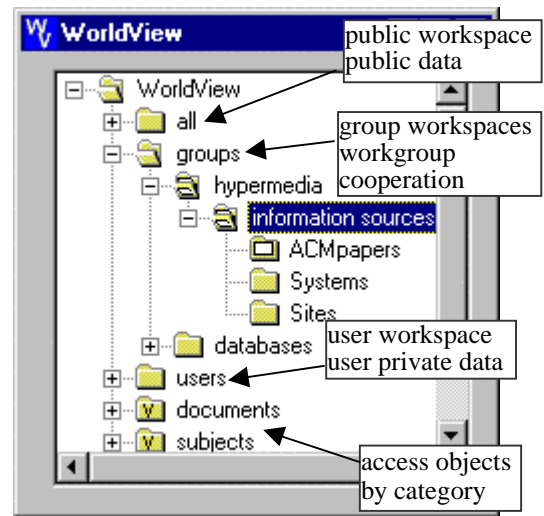


Figure 2 – User interface to WV

Anyone can access the “all” folder, but only users possessing an account can access their own subfolders in the user folder. Group subfolders are accessible by users who belong to the corresponding group. Objects belong to a user and also have privilege properties managed by their owner. The folders marked with a ‘V’ are views showing certain categories of objects.

Figure 3 shows a Workspace. It is a flexible interface allowing users to organize information in a spatial layout as in Dolphin [5] , Sepia [18], Cowfish [21] and Viki [10]. In our approach, workspace collection representations are similar to folder representations but include spatial layout information for the items in the collection.

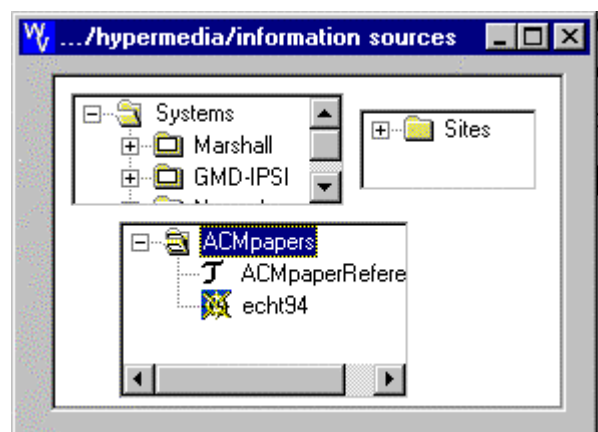


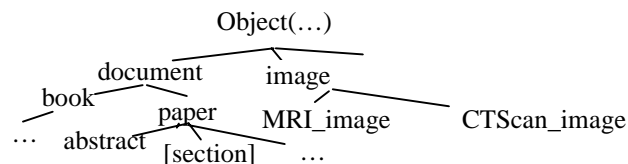
Figure 3 – Information sources workspace

#### Basic WV Data Model

Our model is based on a dynamic object-oriented philosophy relying on the triplet  $\langle O, R_f, R_r \rangle$  of object, referencing and relating objects. The Dexter reference model [6] is sufficiently rich to support our data model. Now we succinctly present the components of the model :

- Objects (O) - An object oriented database management system is used, representing inheritance, composition, association, etc. The object definition and manipulation capabilities of OQL are used to manage the objects. The object types can be: information items (predefined objects carrying information, e.g. a SGML document is a composite information item which includes chapter information items); User-defined objects; Nodes, sets and collections;
- Referencing objects ( $R_f$ ) are generic anchors, then further specialized in specific referencing types: Image contours, text regions, etc.
- Relating objects ( $R_l$ ) are generic n-ary links between object instances, further specialized in specific relating types. The relationship can be expressed by a dynamic OQL query or enumeration of objects and implement a basic navigation mechanism.

Figure 4 shows the objects in part of an application hierarchy example. Object types can be designed for a specific application. Given the appropriate privileges, the user can extend those types and create new ones.



**Figure 4 – Exemplifying an object type hierarchy**

Any object can be referenced and related. But to support within-component access and linking for navigation (hyperenabled objects), objects must have extended attributes (default) for referencing (refs:Set[Referencing]) and relating (rels:Set[Relationship], drels:Set[DynRel]);

#### Extending the Model to Handle Automatic Submission

The first step to extend the system is supporting user-defined types by allowing the semi-automatic submission of objects and relationships.

When creating a new object type in an object database, the user must specify the placement in the inheritance hierarchy, the attributes and the respective domains. Then he can instantiate the objects. This process works well in the assumption of a top-down database construction [23] with a schema specification step first and then a database population with data confined to that schema. We assume a hybrid top-down and bottom-up approach, with emergent structure. Therefore, we must provide intuitive means to create structure. Our system offers the semi-automatic object submission interface to allow the user or the presentation layer of the system to submit objects with incomplete information. This relieves the user from having to specify types and still maintains strong typing. The following non-interactive submission does not require the type Person to exist and attribute definition is optional. Defaults are applied for each missing attribute in the submission.

```

attribute age(Person)(number(3)); // optional
Pedro = Person(name:"Pedro",age:40,
Address(street:"VAlegre",nr:2,city:"Coimbra",zip:"30"),
brothers[Jose,Manuel]);
  
```

The system analyses each non-defined attribute and infers a type and length for it using default customizable rules. The corresponding interactive submission is designed at the presentation layer level and offers different degrees of object specification, from free values to detailed object type proposals. This mechanism is used to assist in structure emergence. Presentation user-interface related issues are discussed in a later section.

Relationship submission can be fully automated. Relationships relate any number of objects in the Set[subset[O<sub>i</sub>], subset[O<sub>j</sub>]] with the linking label Rlnkname. By default, the system automatically assigns a name to the relationship that is the concatenation of the two object type names and a relating label. In the example submission, Pedro.relate(image34,"in") if "Pedro" is a "Person" object and "image34" is an "image" object, the relationship will be an instance of "in\_Person\_image". If this type doesn't exist yet, the system creates and then instantiates it automatically. Having this objective way to name relationships helps the user (or presentation layer) querying and displaying the relationships.

The advantages of these schemes are twofold: arbitrary objects and relationships can be submitted by users (subject to workspace privileges) and both OQL (object query language) and metadata manipulation can be used by the presentation layer to offer an intuitive and flexible interface with powerful features.

#### 4 EMERGENCE AND METAMORPHOSIS

Structure emergence is the process of discovering structure in information and metamorphosis refers to the transformation of structure. Raw data can evolve through a process of emergence and metamorphosis. At any given moment different degrees of formality coexist. In this section we explain how our strongly typed object model implements this functionality. To exemplify we build an emergent structure from HTML pages containing bibliographic references.

#### Emergence, Metamorphosis and Type Querying

We mentioned in the introduction that structure emergence is also available in other systems, but they use frame-based representations. Explicit typing is considered harmful: the user is flooded in a non-intuitive intensive and premature typing activity. Our system offers different degrees of user cooperation in typing. Strongly typed objects are beneficial because they more fully express semantics. This way the OODBMS querying and manipulation facilities that were built for strongly typed objects can be directly used by the presentation layer for retrieval, querying and dynamic linking. This is especially important when a large body of automatically, semi-automatically and manual information is gathered by the system. The usual typing, taxonomizing and querying operations will benefit.

Figure 7 shows the paths from unstructured input information to structured data. External information is provided either by the user or loaded from a file or an information gatherer. This information may be directly parsed to objects or directed to the user workspace for interpretation. Information streams, information pieces, weak and strong object types coexist. Object contents can be interpreted for further metamorphosis (objects can also be unparsed back to external entities).

We concentrate now in semi-automatic information transformation. In our prototype system we have built a html page editor to investigate the emergence features. In this case both the displayed spatial layout and the source html page markup are important. Figure 8 shows an html page. Figure 8c shows the desired result of a simplified interpretation example concerning the page in 8a) with information pieces like the one in 8b).

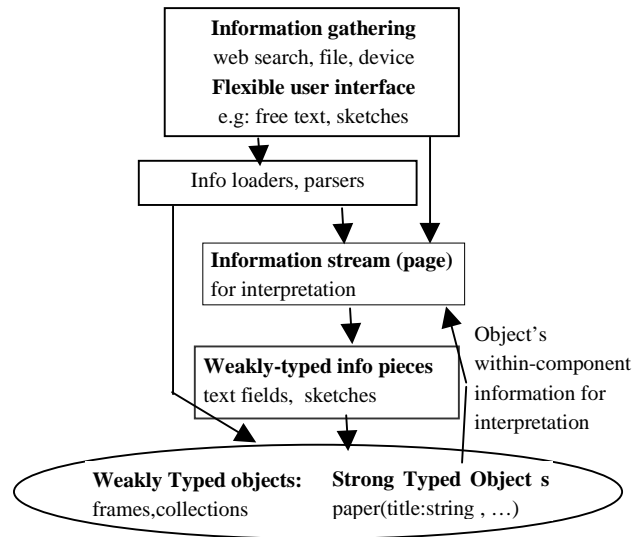


Figure 7–Motivation for Emergence

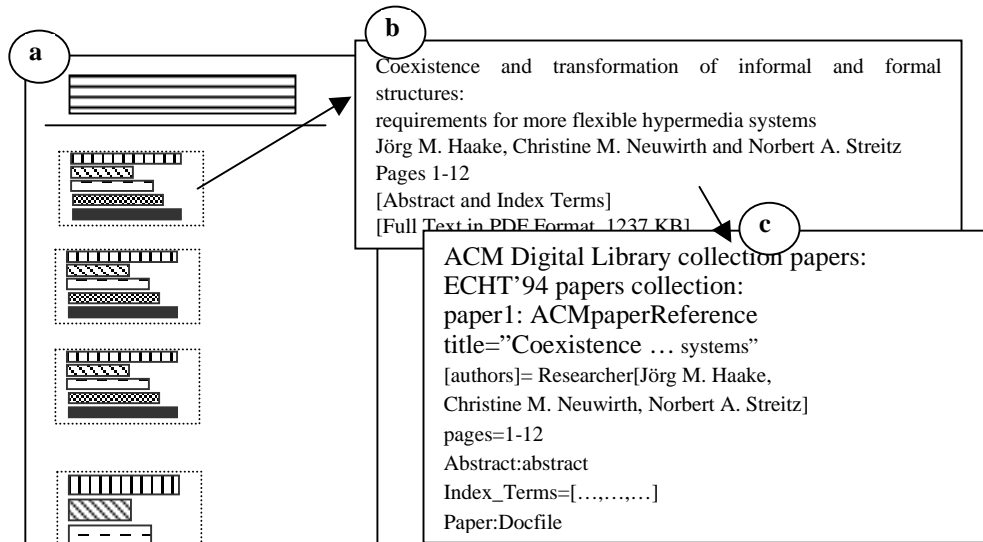
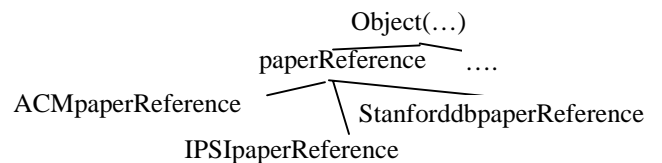


Figure 8 – Information stream for interpretation and a possible result

If the structures in Figure 8c are achieved, it will be quite simple and intuitive to issue queries. For instance, to retrieve all the papers including a given author (Haake), a query like the following one could be used:

```
SELECT paperReference INTO HaakepaperReferences[]
FROM paperReference
WHERE paperReference.authors.contain(Haake);
or (for name containing "Haake")
SELECT paperReference INTO HaakepaperReferences[]
FROM paperReference
WHERE paperReference.authors[].name like '%Haake%';
```

The previous query searches all paperReference objects in the system. This includes all other objects that derive from paperReferences such as in:



This querying capability is very important, because it allows the implementation of a totally dynamic linking scheme. The presentation layer should be able to use this mechanism intelligently to offer visual intuitive dynamic linking to the user. This can be done in a way similar to OVID querying facilities [15] with a (SELECT... FROM... WHERE) mask that is completed by the user specifying the object types and attributes to search from lists of possible ones. But it can also be implemented with a wizard: in our system the user is viewing the content of

an object and selects an author (“Haake”). Then he chooses a menu item to view related objects and the wizard guides him choosing the object type(s) to query from the hierarchy (in this case “paper\_reference”) and the corresponding object attributes to compare with the selected attribute. The results from the queries are automatically placed in a collection in the current workspace.

Table 1 shows the classes of objects that coexist in the metamorphosis system using our own classification (all implemented as object types).

Class	<i>Weak typed info pieces</i>	<i>Weakly typed objects</i>	
<b>Types</b>	Info. pieces	Collection	Frame
<b>Struct</b>	Fields, Set[fields]	collection(obj:set[object])	videoScene(objects:set[Object]) <may assign types to slots>
<b>Expls</b>	uninterpreted text	folders, workspaces	typed collections

Class	<i>Semi-weakly typed objects</i>	<i>Strong Typed objects</i>
<b>Types</b>	Semi-Frame	Object
<b>Structure</b>	SFrX ([atname:value], slots:set[Object])	ObjX ( [atname:value])
<b>Examples</b>	Relating(name:string, objsl:set[object])	paper(abstract:string, sections:set[section])

**Table 1–Metamorphosis classes**

Weakly typed objects are either frames or collections. A collection is simply a named bag of objects of type collection (collection:set[object]). Folder and workspace object types are derived from collection. A frame is a typed bag of objects (slots) (FrameTypeX:set[object]). A semi-weak frame type contains slots and typed attributes. Strong object types contain only typed attributes. The user interface is based on nested collections, so it is important to support some collection-related queries as well. An example is “find all the collections containing paper references with author “Haake”. The presentation layer can answer with a query:

```
SELECT paperReferences.rels INTO Haakecollections[]
FROM paperReferences
WHERE paperReference.authors.contain(Haake)
AND paper.rels.GetType() = collection;
```

This query supposes the paperReference type is hyperenabled (contains relating (and referencing) object references). Alternatively, an extended query language similar to the one in [2] could be used to automatically implement object walking in the collections and frames to search for “Haake”. This is retrieval with incomplete knowledge:

```
SELECT collection INTO result[]
FROM collection
WHERE PATH_p.collection.paper.author[].name
contains('%Haake%');
```

Taxonomizing is easy with structured objects. It creates collections of object references based on a common property. Generic query languages already have a GROUP BY clause to group the results based on an attribute. This could be extended to select the groups of objects into collections. To index paperReferences by category based on an attribute such as the author name:

```
SELECT paperReference INTO ReferencesByAuthor[]
FROM paperReferences
GROUP BY author[].name;
```

The fact that many operations deal with collections is stressed in [17]. Query language support for collection querying as extensively as possible would be important in our system and in hypermedia systems in general. Extending query languages to handle collections is also an important research issue [20].

### The Emergence and Metamorphosis process

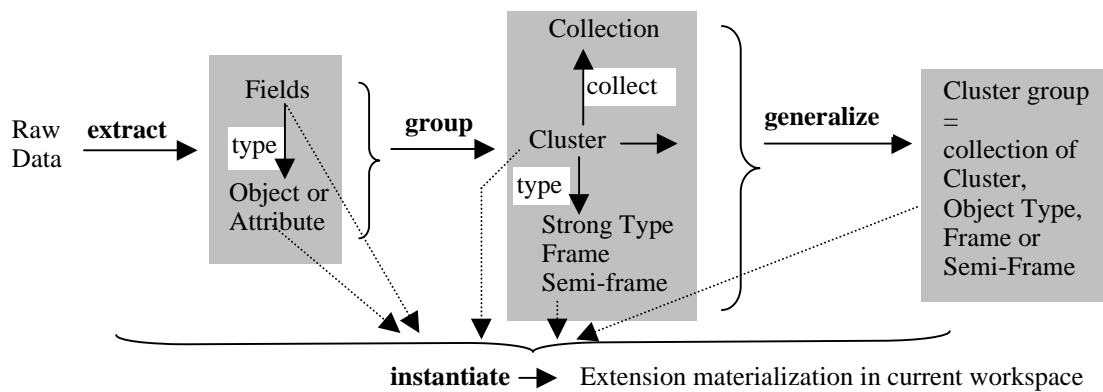
Given an input page, the user and the system engage in discovery to achieve weakly-typed information pieces. The basic structural type in the page is the field (usually text field). A group of fields can form a cluster and a group of clusters can form a cluster group (clusters need not be contiguous). The user indicates the structures by visual and sintatic clues. Visual clues (spatial) correspond to a region selecting tool and sintatic clues are patterns present in the input. A “Generalize” menu item commands the system to automatically “extract” and outline all the clusters in the cluster group based on the clues. The user can repeat this process after evolving the clues.

The cluster in Figure (8b) was extracted with the help of the user and then the system generalized to the other clusters. The most important sintatic pattern in the clusters was the <BR> markup after text pieces in the source HTML page to detect the end of each field. The visual clues were the division in two cluster groups, the title region and the paperReferences region, the individualization of the first paragraph and the indication of the text fields. A twin shadow page keeps track of the types assigned to the spatial patterns in the workspace. The whole operation can be recorded to be applied to similar pages. Patterns such as “http://” can be specified to translate directly to a given type (URL).

Table 2 shows typical translations from fields to types.

From	To
Text fields (and other “atomic” input)	Attribute (of any type)
Cluster (text field set)	Object type, objects
Cluster group	Object type extension

**Table 2 -Translation from fields to types**



**Figure 9- Possible paths from raw data to structured types**

Figure 9 depicts possible paths from raw to structured data in our system. The process can be described as having four sub-processes - extraction, grouping, generalization and instantiation.

- Extraction - Fields are created in the raw data by simple selection/outlining. By using a "type" command, fields can optionally be transformed into objects or attributes.
- Grouping - Data resulting from the basic extraction level can be grouped into clusters by selecting the desired extracted data to include in the cluster and then the "group" command. A cluster can optionally be transformed into a type (strong type, semi-frame or frame) or a collection.
- Generalization - Generalization automatically discovers similar clusters in the input data, creating a cluster group.
- Instantiation - Instantiation produces the actual objects in the hyperbase and places them in the current workspace.

After a complete type emergence and metamorphosis, the workspace displays the source page, the final type structure and the resulting object collection of Figure 10. Now we discuss how this is achieved by exposing features of type emergence and metamorphosis in our system:

- Typed, untyped and weakly-typed structures coexist at any time. The user can progressively type parts of the page.
- Typing is intuitive and visually guided. It is aided by the semi-automatic object submission capability discussed in a previous section.
- Typing is initiated by clicking on a field, selecting a cluster or an object and choosing the "Type" command.
- The user must choose the type. This can be an existing, derived or new type. If needed, the object hierarchy is displayed.
- The system can try to match fields with the assigned

type or the user can do it manually.

- The assigned type structure is displayed for editing in a new collection window in the current workspace (as in Figure 10).
- A drag-and-drop functionality allows the user to match fields to type attributes and create a new attribute for a field by dragging it to the type structure window. The system aids in type conversion if necessary.
- An editing functionality allows the user to add, delete or modify attributes and domains.
- Typing can be automatically repeated to all similar patterns discovered in the page.
- The "Instantiate" command automatically creates a collection with the instances extracted from the page. The result collection contains the extension or part of the extension of the specified object type (e.g the ECHT94 papers collection is an extension of type ACMpaperReference in Figure 10).

There are also some difficulties with this process:

- The structure of the clusters may differ slightly (absence or surplus of fields). We solved this problem by attaching optional patterns for field matching in each attribute of the type structure. This was enough in our example, possibly not in other examples.
- The HTML editor is a visual display tool, so it hides the markup. Sometimes non-displayed information from the markup is needed in the emergence process: drag-and-dropping a URL in the html editor to a string attribute assigns only the label that is visible in the editor. But when the URL is dragged to a URL object type both the label and the url part are filled in URL(label,url); The <BR> markup in the html source page was used to help delimit text fields in the first structure emergence phase. This is not suggested in the visual display but a "Show/Hide" button allows the user to see the markup.

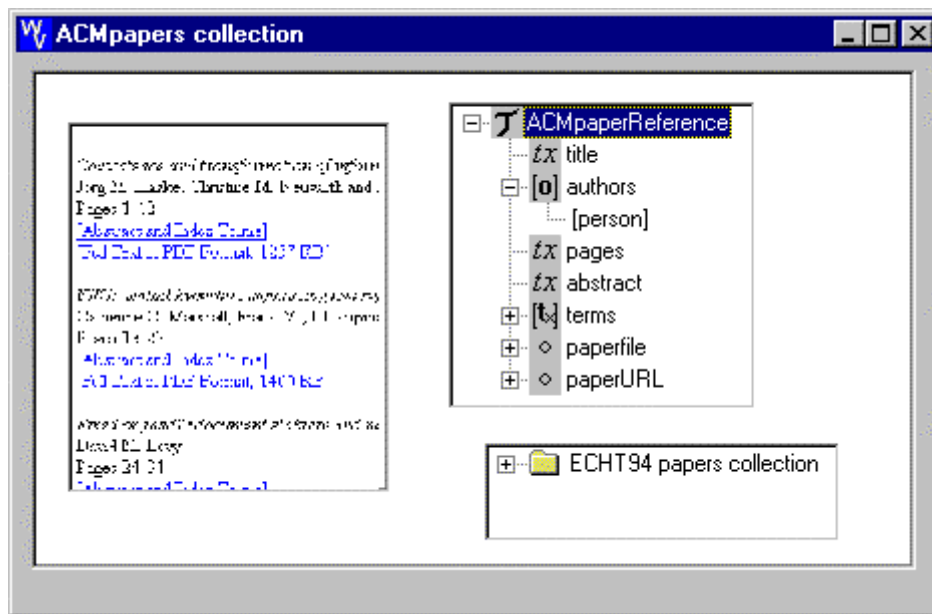
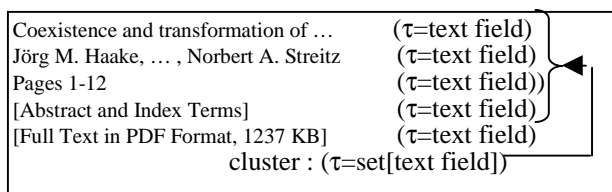


Figure 10 –Paper references html page after collecting the objects

- Dragging a string to a set attribute. The authors field should not be dragged directly to the authors attribute. Instead the user first divides the text field by choosing the “Divide” option and specifying the ‘;’ delimiter. This creates a cluster of text fields that the user drags to the authors attribute.

The following example illustrates some of the aspects of emergence. Given the following stage and supposing it was already generalized to the cluster group of paper references in the page,



Suppose the ACMpaperReference type does not exist yet but the paperReference type does, with the attributes  $\tau$ =paperReference(title:string,authors:[researcher]). The user selects a new ACMpaperReference type as derived from paperReference. The type structure is displayed in a window. In this case it should be something like  $\tau$ =ACMpaperReference:paperReference(title:string,author s:[researcher]). Now the user would drag the first field to the title attribute. Then he would divide the second field and drag the resulting cluster to the name attribute in the researcher type in the authors field. This action would create three researcher objects with only the name filled. Now the user would add new attributes to the ACMpaperReference type structure and continue with the

process. After the user is done with typing, he can select the whole paperReference cluster group and choose the “Instantiate” option. The system automatically creates the corresponding objects for the page in a result collection.

As mentioned previously, metamorphosis is the process of modifying the structure of information. Up to now we were mainly concerned with type emergence but type modification is similar. Instead of an input stream, now the source is a collection of objects belonging to the extension of a type (a simple OQL query can build a collection based on any desired condition). Similarly to emergence, a new type must be chosen (either from existing ones or by specifying a new one). Having the two type structure windows opened, the user specifies a mapping by dragging the attributes. Then the user commands the system to transform. It creates the result collection and transfers or copies the objects from the source collection to the result collection, performing the mapping along the way.

## 5 CONCLUSIONS

In this paper we have answered the questions: how can we enforce strong typing in an emergent structures environment? And why is this so important? But simultaneously we took into account the complaints that over-typing and over-organizing activity preempts usability. This way we presented a flexible system which at the same time offers a great degree of flexibility and helps the user enforcing types for repeating patterns gathered from diverse sources. This results in a system that can use OQL to achieve dynamic linking for navigation, indexing and information organization, not only over base types but also over emerged ones. Most hypermedia systems and specifically those that support emergence and

user-defined structures keep a frame-based representation, which by its own nature doesn't enforce a regular structure and wastes the advantages of having object query capability. We focused on the situation where an external information source produces repetitive patterns to be transformed into objects with the aid of the user. So this work has a clear implication in the research for hypermedia systems and database systems alike.

## 6 REFERENCES

- [1] Ajit Bapat, Jürgen Wäsch, Karl Aberer and Jörg M. Haake: HyperStorM: an extensible object-oriented hypermedia engine Proceedings of the seventh ACM conference on Hypertext '96, March 16-20, 1996, Washington, D.C. Pages 203-214.
- [2] Christophides, V. Rizk, A. Querying Structured Documents with Hypertext Links using OODBMS, Pages 186-197, ECHT '94. Proceedings of the 1994 ACM European conference on Hypermedia technology, Sept. 18-23, 1994, Edinburgh, Scotland, UK.
- [3] Daniel Cunliffe, Carl Taylor & Douglas Tudhope: Query-based Navigation in semantically indexed hypermedia. Proceedings of The Eighth ACM Conference on Hypertext 1997.
- [4] Haake, J., Wilson, B. (1992). Supporting collaborative writing of hyperdocuments in SEPIA. In Proceedings of the ACM Conference on Computer-Supported Cooperative Work (CSCW'92) p 138-146. Toronto, Canada, Oct 31- Nov 4.
- [5] J. Haake, C. Neuwirth, and N. Streitz. Coexistence and transformation of informal and formal structures: Requirements for more flexible hypermedia systems. In *Proceedings ACM European Conference on Hypertext Technology*, pages 1–12, Edinburgh, 1994.
- [6] F. Halasz and M. Schwartz. The dexter hypertext reference model. *Communications of the ACM*, 37, 2, pages 30–39, February 1994.
- [7] Marshall, C.C., Halasz, F.G., Rogers, R, A., Janssen Jr., W.C. Aquanet a hypertext tool to hold your knowledge in place ACM HYPERTEXT'91 Proceedings, pp. 261-275, December 1991.
- [8] Marshall, C.C. Two Years before the Mist Experiences with Aquanet ACM ECHT '92 Conference Proceedings, pp. 53-62, December 1992.
- [9] Marshall, Shipman III, F.M. Searching for the Missing Link: Discovering Implicit Structure in Spatial Hypertext ACM HYPERTEXT '93 Conference Proceedings, pp. 217-230, November 1993.
- [10] C. C. Marshall, F. Shipman, and J. Coombs. VIKI: Spatial hypertext supporting emergent structure. In *Proceedings of the ECHT'94 European Conference on Hypermedia Technologies*, Papers, pages 13–23, 1994.
- [11] Catherine C. Marshall & Frank M. Shipman III, "Spatial Hypertext and the Practice of Information Triage", Proceedings of The Eighth ACM Conference on Hypertext, 1997
- [12] Yoshihiro Masuda, Yasuhiro Ishitobi and Manabu Ueda, Frame-axis model for automatic information organizing and spatial navigation. Pages 146-157 ECHT '94. Proceedings of the 1994 ACM European conference on Hypermedia technology, Sept. 18-23, 1994, Edinburgh, Scotland, UK
- [13] J. Nanard and M. Nanard. Using structured types to incorporate knowledge in hypertext. In *Proceedings of ACM Hypertext'91*, Hypertext – Integrative Issues, pages 329–343, 1991.
- [14] Nanard, J., Nanard. M. Should Anchors Be Typed Too? An Experiment with MacWeb ACM HYPERTEXT '93 Conf. Proc., pp. 51-62, Nov 1993.
- [15] Oomoto, E. and Tanaka, K., "OVID: Design and implementation of a video-object database system, "IEEE Trans. On Knowledge and Data Engineering, Vol. 5, No. 4, Aug 1993, pp 629-643.
- [16] Oomoto, E. and Tanaka, K. "Video Database Systems-Recent Trends in Research and Development Activities" In Chapter 13, "The HandBook of Multimedia Information Management" Eds. Grosky W., Jain, R., Mehrotra, R. , ISBN 0-13-207325-0.
- [17] Parunak, H.V.D. Don't Link Me In: Set Based Hypermedia for Taxonomic Reasoning ACM HYPERTEXT '91 Procs, pp. 233-242, December 91.
- [18] Norbert Streitz, Jörg Haake, Jörg Hannemann, Andreas Lemke, Wolfgang Schuler, Helge Schütt and Manfred Thüring. SEPIA: a cooperative hypermedia authoring environment ECHT '92. Proceedings of the ACM conference on Hypertext, November 30-December 4, 1992, Milan, Italy Pages 11-22
- [19] N. Streitz, J. Geißler, J. Haake, and J. Hol. DOLPHIN: Integrated meeting support across local and remote desktop environments and liveboards. In *Proceedings of ACM CSCW'94*, pages 345–358, Chapel Hill, N.C., U.S.A, October 22-26 1994.
- [20] Tannen, V. Tutorial: Languages for Collection Types SIGMOD/PODS '94 Conference.
- [21] Wang W, Haake J. Supporting User-defined Activity Spaces. In Proceedings of The Eighth ACM Conference on Hypertext, 1997.
- [22] Wash, J. and Aberer, K. Flexible Design and Efficient Implementation of a Hypermedia Document Database System by Tailoring Semantic Relationships. Proceedings of the Sixth IFIP Conference on Data Semantics (DS-6), Stone Mountain, Georgia, May 30-June 2, 1995.
- [23] Zdonik, S.B. Incremental Database Systems: Databases from the Ground Up ACM SIGMOD '93 Conference Proceedings, pp. 408-412, June 1993.