PatMan: A Visual Database System to Manipulate Path Patterns and Data in Hierarchical Catalogs

Angeliki Koukoutsaki, Theodore Dalamagas, Timos Sellis, Panagiotis Bouros

Knowledge and Database Systems Lab
School of Electrical and Computer Engineering
National Techn. Univ. of Athens, Athens, Greece
{akoukou,dalamag,timos,pbour}@dblab.ece.ntua.gr

Abstract. Hierarchical structures are a way to organize and enrich semantically the available information on the Web. Popular examples of such structures are the product catalogs of e-market stores, which provide data (i.e. products) organized in thematic hierarchies on a category/subcategory basis. Users can navigate these hierarchies to identify data of their preference. Such hierarchies group data under certain properties. Navigational paths are the knowledge artifacts to represent such groups. We consider paths in hierarchies as patterns which provide a conceptual clustering of data in groups sharing common properties. Under this perspective, we have designed and implemented *PatMan*, a visual database system to manage hierarchical catalogs. The system can store navigational paths and data from hierarchical catalogs, and provides a pictorial query-by-example language to manipulate path patterns and data in a uniform way.

1. Introduction

The Internet is today's greatest source of information. Huge volumes of data are posted and retrieved through the Web. Despite this vast exchange of information on the web, there is no consistent and strict organization of data [1]. Hierarchical structures are a way to organize and enrich semantically the available information on the Web. Popular examples of such structures are the product catalogs of e-market stores which provide data (i.e. products) organized in thematic hierarchies on a category/subcategory basis. Users can navigate these hierarchies to identify data of their preference. Such hierarchies group data under certain properties. Navigational *paths* in these structures are the *knowledge artifacts* to represent such groups, and act as semantic guides to reach the data of each group either through browsing or through path expression query languages. For example, the path `/moto/bmw/funduros/` represents several BMW motorcycle models designed for off-road riding and can be used in a path expression to search for motorcycles with price less than 9000: `/moto/bmw/funduros[price<9000]`.

We consider paths in hierarchies as *patterns* which provide a conceptual clustering of raw data in groups sharing common properties. We focus on the extraction and manipulation of these patterns, combined with the manipulation of data. See for example Figure 1, where two portal catalogs, Adorama and B&H[1], are presented. These catalogs provide photo equipment organized in a hierarchy on a category/subcategory basis. Searching for `lenses` in the first catalog needs the path `/cameras&lenses/lenses`, while in the second catalog needs the path `/photo/35mm systems/lenses`. Such paths can be seen as alternative pattern versions for the same group of data.

[1] www.adorama.com, www.bhphotovideo.com

On the other hand, there are cases where the owner of a catalog may need to provide integrated photo systems, with camera bodies from Adorama and the appropriate lenses from B&H. In such case, the two paths `/cameras&lenses/35mm SLR` and `/photo/35mm systems/lenses` form a kind of complex pattern representing different raw data, i.e. bodies and lenses, that should be grouped together.

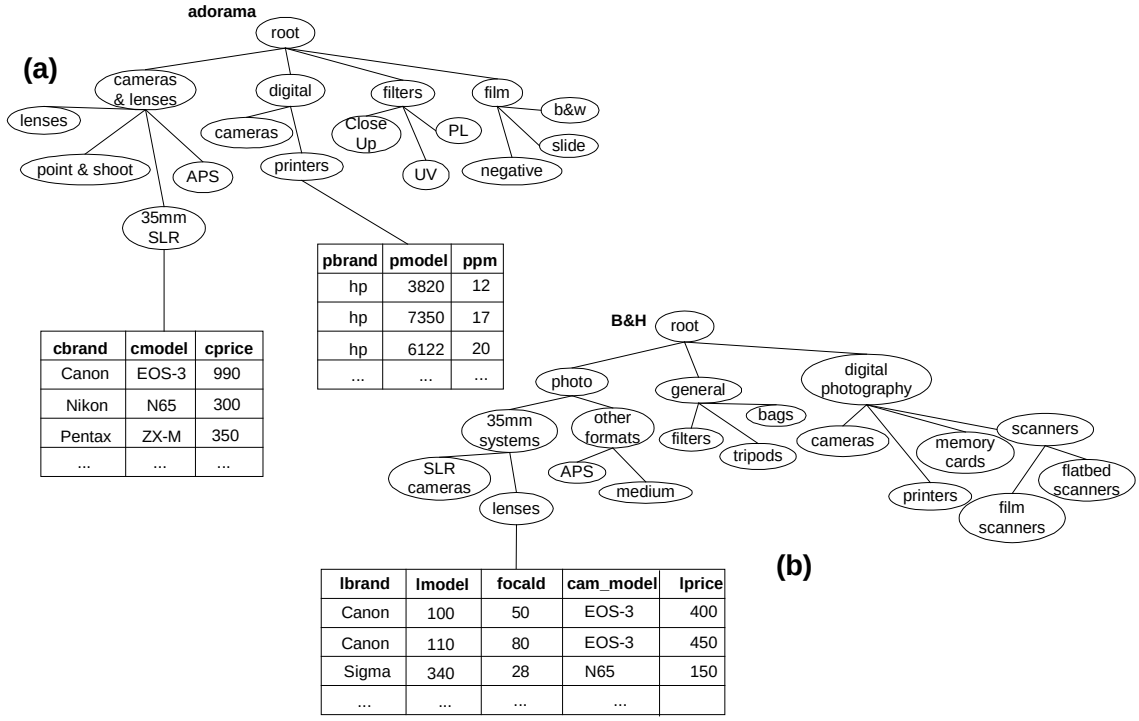


Figure 1: Parts of Adorama and B & H catalogs.

Our work addresses part of the requirements defined for pattern management in [12], [14], emphasizing on the management of path-like patterns and data in hierarchical catalogs. Similar requirements have been also introduced in the inductive database framework [6], where patterns are integrated within the database environment as full-fledged objects. Under this context, a number of specialized inductive query languages have been proposed but most of these concern descriptive rules [5], [10]. Path manipulation techniques for biochemical pathways treated as knowledge artifacts, i.e. biochemical reactions, are discussed in [8] and [13]. In these works, database systems to manipulate and query biochemical pathways are presented. The problem of manipulating paths (and tree-like structures, in general) appears also in XML data management. In [4] the authors present an algebra for XML data, but it is tuple-based and not tree-based. A navigational algebra is presented in [9], but again it treats individual nodes as manipulation units. In [7], TAX algebra is defined, but as a means for selecting and reconstructing bulk XML data.

In our work, we manipulate the descriptions of data as path-like patterns. We capture the notion of alternative path-like patterns and complex patterns to provide a framework to query raw data from many catalogs together with their patterns. Under this perspective, we have already suggested models to represent hierarchical catalogs, emphasizing on the role of paths as knowledge artifacts in such

structures, and defined the *PatMan*QL language to manipulate path-like patterns together with raw data [2].

This paper extents the work in [2] and presents *PatMan*, a fully-functionable prototype system that implements the aforementioned framework. The system provides modules to import structures from hierarchical catalogs coming in various forms (e.g. RDFs representation), and stores them in a relational database system. It also helps the user posing queries using a visual interface that offers pictorial query-by-example capabilities. Finally, it can visualize the query results, showing navigational paths for hierarchical catalogs as well as the raw data organized in these catalogs.

The rest of the paper is organized as follows. Section 2 discusses representation issues for hierarchical catalogs. Section 3 presents an overview of the *PatMan*QL language. Section 4 gives an overall description of *PatMan* system. Section 5 describes some application scenarios using *PatMan*. Finally, Section 6 concludes this paper and presents some issues for further work.

2. Representing Hierarchical Catalogs

We consider data in hierarchical catalogs to be records in relations, organized in the leaves of the hierarchy. These relations are called *resource items*. Every resource item is characterized by a number of *attributes*. For example, `SLR cameras` is a resource item with attributes brand, model and price, reached through the path `/cameras&lenses/35mmSLR` in Figure 2. Three products (cameras) are records of this resource item. The hierarchy together with the resource items form the *catalog schema*.

Definition 1. A catalog schema (CS) is a tree CS={r, N, R} with a root r, a set N of nodes as non-leaf nodes and a set R of resource items as leaf-nodes.

Figure 2 presents the catalog schema of Adorama's catalog for photo equipment.

We can combine several catalog schemas with common resource items, creating *tree-structured relations (TSRs)*. Common resource items are items with a similar semantic interpretation in the knowledge domain that catalogs refer to. Intuitively, a TSR represents the different ways of accessing a resource item in a set of catalog schemas and can be modelled using an AND/OR-like graph. This graph can be seen as a set of patterns, where each pattern is one of the different ways to access a resource item.

Definition 2. Let S = {{r, N_1, R_1}, {r, N_2, R_2}, …, {r, N_k, R_k}} be a set of k catalog schemas in a knowledge domain and ri a resource item in one or more R_i, $1 \leq i \leq k$. A TSR is a graph G having a root r, a node representing the resource item ri and all paths from r to ri. Paths are grouped in OR components. Every OR component can be either an individual path or an AND group of paths.

Figure 3(a) presents an example of a TSR with two OR components, one of which is an AND group denoted by the curved line crossing the involved paths `/photo/35mmSLR/bodies` and `/photo/lenses`.

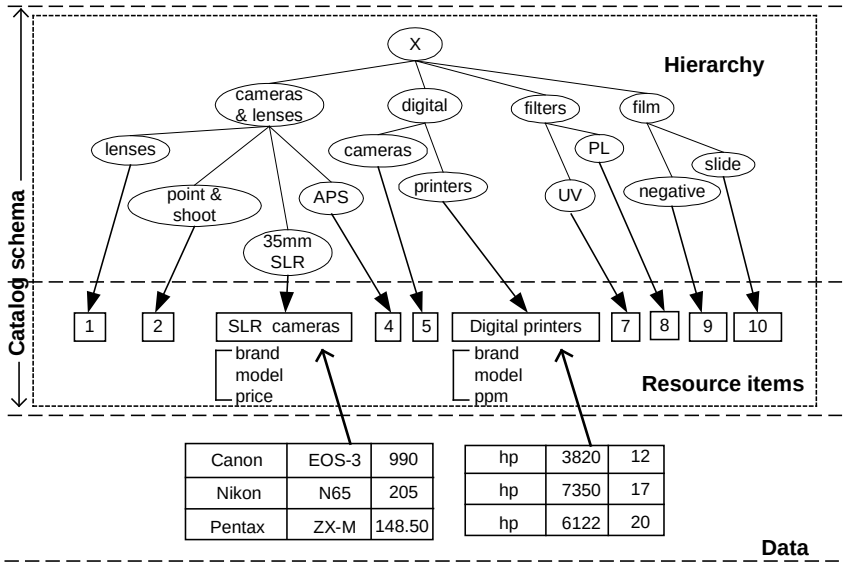


Figure 2: A catalog schema for Adorama's catalog.

Intuitively, an OR component captures a way to access a resource item. For example, `/photo/35mm systems` is one of the two alternative patterns for the resource item `SLR cameras`. An AND group corresponds to complex resource items that can be constructed using items from one or many catalog schemas. To access such items, one should exploit all paths of the group. For example, the AND group in Figure 3(a) corresponds to `SLR systems` built from camera bodies and lenses, two resource items that are in different catalog schemas. The construction of AND groups is closely related to the cartesian product operator that we present in the next section.

Note that in this work we do not consider schema matching issues. We assume that a schema matching pre-processing resolves name mismatches and other inconsistencies [11].

3. The *PatM*anQL Language

In this section, we briefly discuss the *PatM*anQL language (a detailed description of the language is presented in [2]). The *PatM*anQL language manipulates paths together with raw data in hierarchical catalogs. It is based on a set of operators (*select*, *project*, *cartesian product*, *union*, *intersection* and *difference*) to manipulate TSRs from catalog schemas.

1. Select (σ). Select operates on a TSR to produce a new one, selecting instances of resource items and OR components whose paths satisfy some defined predicates. Figure 3(a) shows an example involving a select operator in the query `construct a TSR to include all cameras from TSR 'SLR systems', other than Pentax, with price greater than 200, having '/photo/35mm systems' in their paths`:

σ<brand≠‘Pentax’,price>200><‘/photo/35mm systems’⊆$_>(SLR systems)

Results are presented in Figure 3(b). Notice that /photo/35mm systems⊆ $_ holds if there is a path p in an OR component, such that /photo/35mm systems ⊆ p (i.e. the sequence of nodes in /photo/35mm systems appears identically in p). In this case, the whole OR component is retrieved.

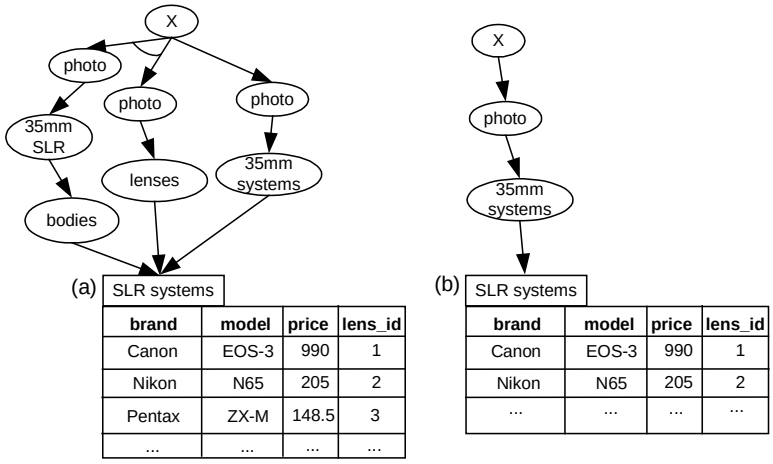


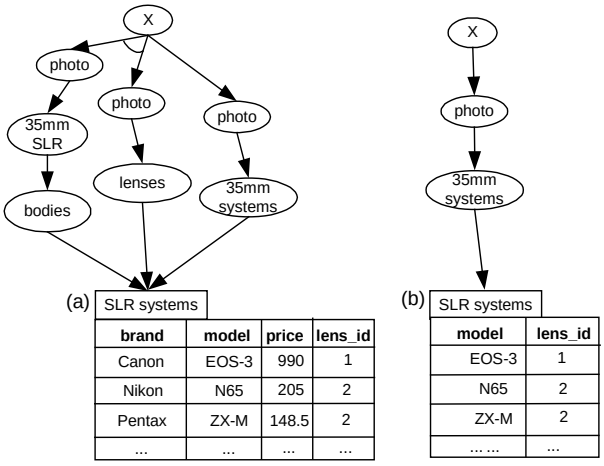
Figure 3: *select* operator.

2. **Project (π).** Project operates on a TSR to produce a new one, keeping only some of the paths of each OR component or OR components on the whole, and some of the attributes of the related resource item. Figure 4(a) presents an example of a project operator in the query construct a new TSR from the TSR ‘SLR systems’ keeping only the rightmost component (i.e. #2) and the attributes ‘model’ and ‘lens_id’ of the resource item:

π<model, lens_id><#2>(SLR systems)

Results are presented in Figure 4(b).

3. **Cartesian product (x).** Cartesian product operates on two TSRs to produce a new TSR, combining (a) every OR component of the first TSR with every OR component of the second TSR, constructing an AND group of paths, and (b) every instance of the resource item in the first TSR with every instance of the resource item in the second TSR. Figure 5(c) shows the cartesian product of the two TSRs of Figure 5(a) and (b). The AND group of paths /photo/35mmSLR/bodies and /photo/lenses (i.e. the leftmost OR component of TSR SLR systems) is combined with the one and only OR component /camera & lenses/lenses of TSR Lenses, to construct a new AND group of three paths. Similarly, the rightmost OR component of TSR SLR systems is combined with /camera & lenses/lenses of TSR Lenses, to construct a new AND group of two paths. The new TSR has a resource item with attributes from both TSRs and combinations of instances.

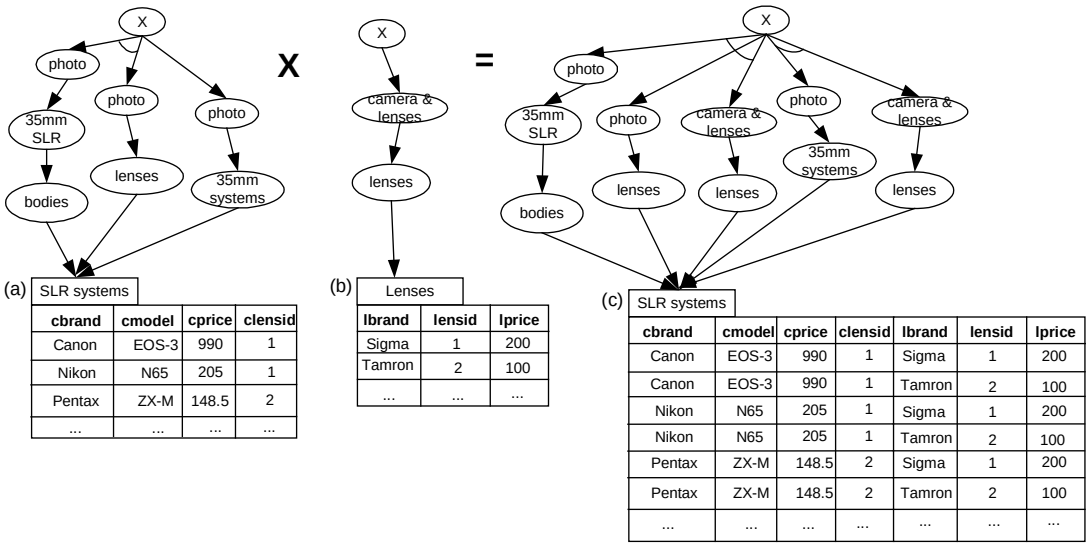


brand	model	price	lens_id
Canon	EOS-3	990	1
Nikon	N65	205	2
Pentax	ZX-M	148.5	2
...	...	...	...

model	lens_id
EOS-3	1
N65	2
ZX-M	2
... ...	...

Figure 4: *project* operator.

4. **Union (U) / Intersection (∩).** Union operates on two TSRs with the same set of resource item attributes. It produces a new TSR with all the OR components of the two TSRs and the union of their instances. Similarly to the union operator, intersection operates on two TSRs with the same set of resource item attributes. It produces a new TSR with all the OR components of the two TSRs and their common instances. Figure 6(c) shows the union of the two TSRs of Figure 6(a) and (b). The TSR constructed has all OR components of both TSRs. Also, the result contains instances either from the first or the second TSR.



cbrand	cmodel	cprice	clensid
Canon	EOS-3	990	1
Nikon	N65	205	1
Pentax	ZX-M	148.5	2
...	...	...	...

lbrand	lensid	lprice
Sigma	1	200
Tamron	2	100
...	...	...

cbrand	cmodel	cprice	clensid	lbrand	lensid	lprice
Canon	EOS-3	990	1	Sigma	1	200
Canon	EOS-3	990	1	Tamron	2	100
Nikon	N65	205	1	Sigma	1	200
Nikon	N65	205	1	Tamron	2	100
Pentax	ZX-M	148.5	2	Sigma	1	200
Pentax	ZX-M	148.5	2	Tamron	2	100
...	...	...	...	...	...	...

Figure 5: *cartesian product* operator.

5. **Difference (-).** Difference operates on two TSRs with the same set of resource item attributes. It produces a new TSR with the OR components of the first TSR and the instances of the first which do not exist in the second.

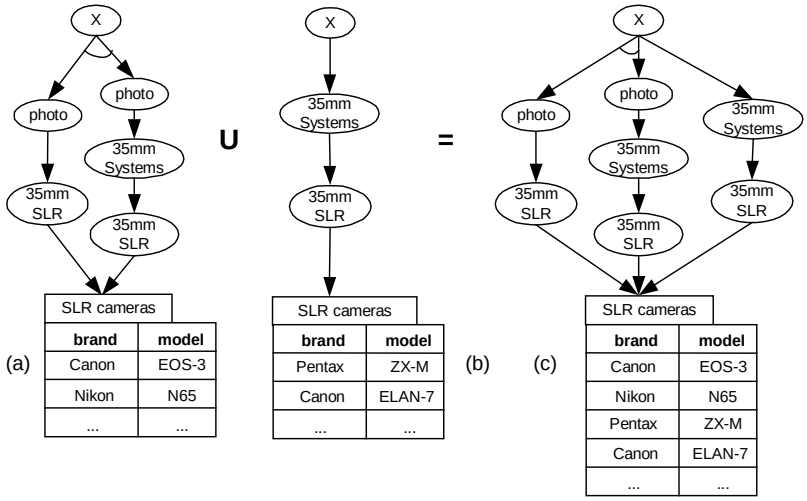


Figure 6: *union* operator.

4. The PatMan System

We have developed PatMan, a prototype system that manipulates paths and raw data in hierarchical catalogs in a uniform way. PatMan provides modules to import TSRs from hierarchical catalogs coming in various forms (e.g. RDFs representation), and store them in a relational database system. It also helps the user posing queries with a visual interface that offers pictorial query-by-example capabilities. Finally, it can visualize the query results, showing the structure of the resulting TSRs as well as the raw data that satisfy the query predicates. Figure 7 shows the architecture of the system. In the next subsections we describe each one of system modules in detail.

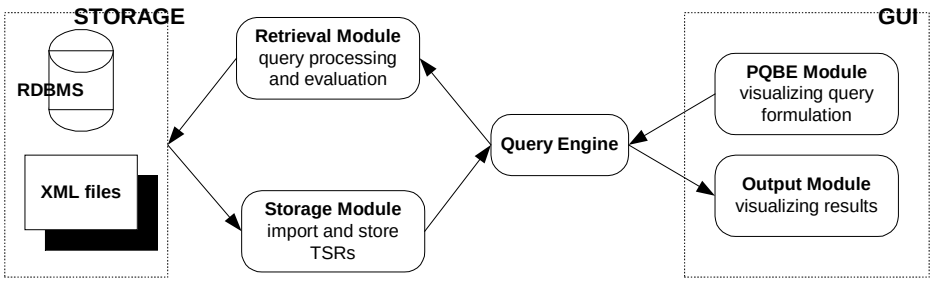


Figure 7: System architecture.

4.1 Storage Module: Import and Store TSRs

Using the storage module, one can import TSRs from hierarchical catalogs coming in various forms, for example in RDFs representation. The module detects IS_A paths in RDFs schemas (i.e. class/subclass relationships) and constructs OR components for TSRs. Resource items are initially empty, but the user

can fill them using data retrieved from other TSRs stored in the system. An example of a RDFs file that encodes a hierarchy of artists is shown in Figure 8.

```
<rdfs:Class rdf:ID="Root"/>
<rdfs:Class rdf:ID="35mm-Systems"/>
<rdfs:subClassOf rdf:resource="#Root"/>
</rdfs:Class>
<rdfs:Class rdf:ID="35mm-SLR">
<rdfs:subClassOf rdf:resource="#35mm Systems"/>
</rdfs:Class>
<rdfs:Class rdf:ID="lenses">
  <rdfs:subClassOf rdf:resource="#35mm Systems"/>
</rdfs:Class>
<rdfs:Class rdf:ID="MF-lenses">
  <rdfs:subClassOf rdf:resource="#lenses"/>
</rdfs:Class>
```

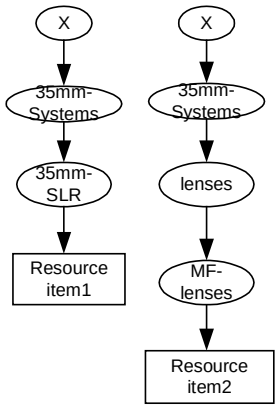
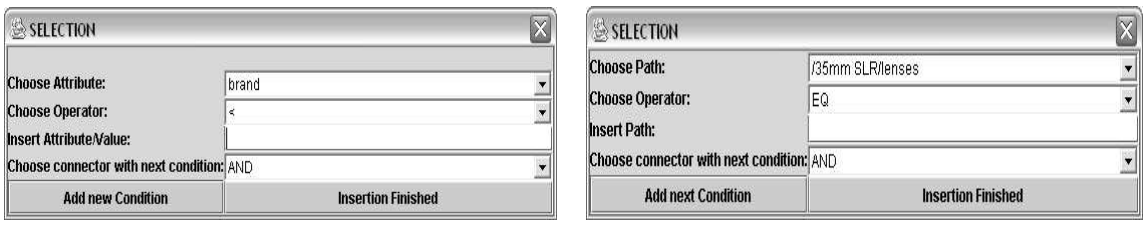


Figure 8: TSRs extracted from a RDFs hierarchy**.**

Figure 8 also presents the TSR extracted from the previous RDFs hierarchy. Extracted TSRs can be stored (a) using plain XML files or (b) using the PostgreSQL RDBMS system, utilizing a relational schema that follows the all-edges-in-one-table example [3].

4.2 Pictorial-Query-by-Example (PQBE) Module: Visualizing Query Formulation

The PQBE module helps the user posing queries in *PatMan*QL. The visual interface provided offers Pictorial-Query-by-Example capabilities, and, thus, minimizes user errors coming from queries that do not conform to the schema of a TSR, or lack certain attributes from the resource items used, etc. The user may apply the unary operators *select* and *project* to manipulate a single TSR. Figure 9 shows the dialog boxes used to apply a selection operator on a TSR. Figure 9(a) shows the dialog box that restricts the attributes that the user can select (e.g. `brand`) , according to the involved resource item. The dialog box in Figure 9(b) restricts the paths that the user can select by retrieving and showing automatically the paths of the involved TSR (e.g `/35mm SLR/lenses`).



(a) **(b)**

Figure 9: Dialog boxes for *select* operator.

Figure 10 shows the dialog box used to apply a projection operator on a TSR. This dialog box restricts the user's options by retrieving and showing the available attributes, paths and OR components of the involved TSR.

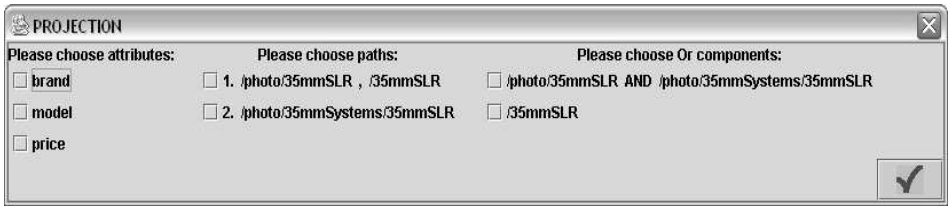


Figure 10: Dialog box for *project* operator.

The user may also apply the binary operators *union*, *intersection*, *difference*, *cartesian product* to manipulate a set of TSRs. In this case, the user should initially select all the TSR schemas that will participate in the query. Then, she can select a pair of TSRs and apply one of the binary operators available. The new TSR constructed as the result of this operation can be further processed as a single entity using a unary operator or can participate in a new query using a binary operator and another TSR. The task continues till the user constructs a TSR of her preference using the binary and unary operators available. As the dialog box in Figure 11 shows, the user can apply a unary operator in each TSR as well as a binary operator to combine the two selected TSR schemas. We note that in any step of the query formulation, the user can call the output module and see the structure of the hierarchy maintained in all TSR participating as well as the data organized in its hierarchy.

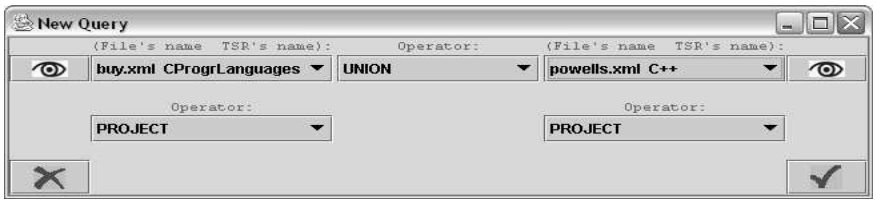


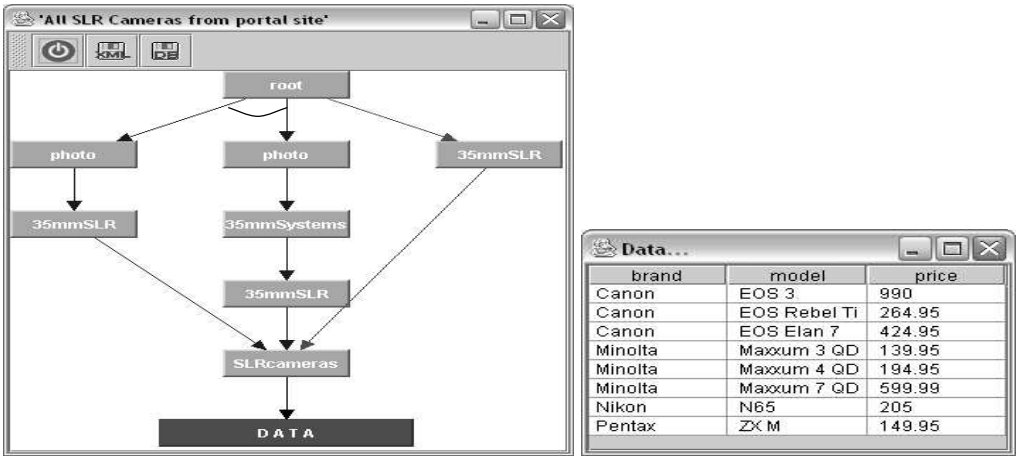
Figure 11: Dialog box for creating a query.

4.3 Query Engine/ Retrieval Module

These are the system modules where the actual evaluation of the queries takes place. Queries are reformulated to PatManQL expressions using the operators *select, project, cartesian product, union, intersection* and *difference* discussed in Section 3. The TSRs involved in the query are retrieved either from the XML files or the tables of the RDBMS creating instances of TSR structures in the main memory. These structures are created dynamically, and capture the form of OR components, the attributes and the records of resource items. Depending on the operator, the OR components are processed separately from data records. The manipulation of OR components is based either on selecting/pruning single paths or groups of them, or by combining paths, or by unifying them under a common root. Data management functionality is provided, implementing SQL-like querying operations.

4.4 Output Module: Visualizing Results

The output module presents the query results, visualizing the structure of the hierarchies maintained in the TSRs and the data records organized in the resource items. It also helps the user to explore TSRs which are already stored either in plain XML files or in the RDBMS. Figure 12(a) shows a TSR schema which is the result of a query. The AND groups are denoted by the curved line crossing the involved paths (in the original implementation paths in a certain AND group have identical colors). For example, `/photo/35mmSLR` and `/photo/35mm Systems/35mmSLR/SLR Cameras` belong to an AND group. Resource items are represented by the DATA nodes. Selecting such a node, a new window opens, showing the attributes and the data records of the resource item (see Figure 12(b)).



(a) (b)

Figure 12: Graphic Result Interface.

4.5 Other Functionalities

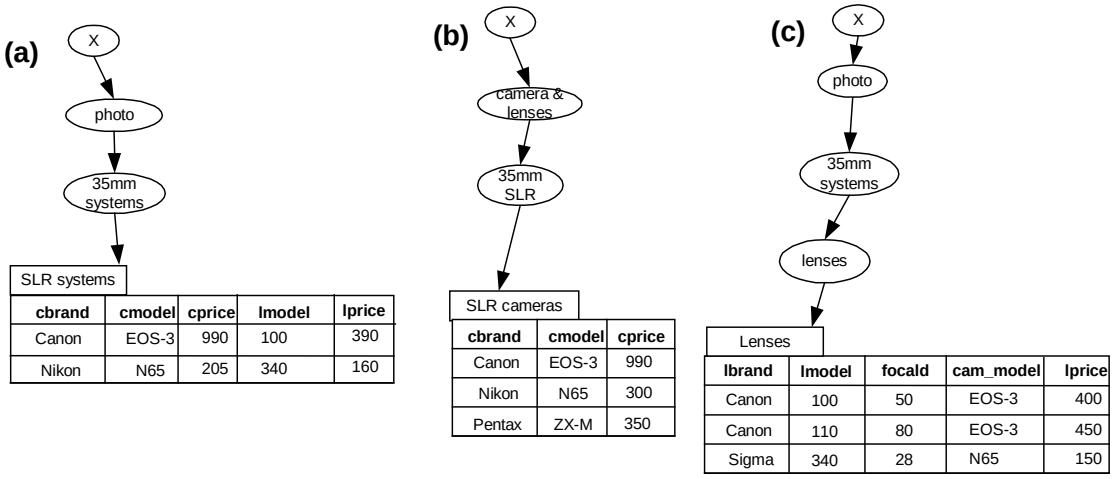
The *PatMan* system, beside its core functions for querying and storing TSRs from hierarchical catalogs, offers a set of tools supporting the user to perform other administration and manipulation tasks:

1. *Query Storage.* Queries can be still written (following original *PatMan*QL's syntax) in text files and loaded in the system to be executed on the TSRs stored.
2. *Data Import.* Attributes and data records can be added to empty resource items of stored TSRs. New data records can be also added to existing resource items of stored TSRs. Generally, resource items can be filled or constructed from scratch, given (a) relational tables filled with records or (b) plain record-oriented delimited text files.
3. *XML-RDBMS Import.* TRSs which are stored as plain XML files can be imported in the RDBMS.
4. *RDBMS-XML Export.* TRSs which are stored in the RDBMS can be exported as plain XML files.

5. *TSR re-shape.* The user can change the name of the TSR, the name of the XML file used to encode it, its description and the name of resource items.

5. Application Scenario

This section presents an application scenario to show how our system can be used to manipulate navigational path patterns and data from hierarchical catalogs. Figure 13 presents three TSRs from catalog schemas related to photo equipment. Specifically, (b) and (c) are TSRs from the catalog schemas of Adorama and B&H (presented in Figure 1), while (a) is a TSR of a new, imaginary catalog owned by X to serve the needs of our example. The first TSR is named `SLR Systems` (stored in the file X.xml), the second `SLR cameras` (stored in the file Adoramas.xml) and the third `lenses` (stored in the file 'B&H.xml).



cbrand	cmodel	cprice	lmodel	lprice
Canon	EOS-3	990	100	390
Nikon	N65	205	340	160

cbrand	cmodel	cprice
Canon	EOS-3	990
Nikon	N65	300
Pentax	ZX-M	350

lbrand	lmodel	focald	cam_model	lprice
Canon	100	50	EOS-3	400
Canon	110	80	EOS-3	450
Sigma	340	28	N65	150

Figure 13: TSR examples.

Case 1. X sells integrated photo equipment, that is camera bodies and lenses as one package. Since new lenses are out in the market, X needs to find among the lenses provided by B&H, those that fit in 'Canon' bodies provided by Adorama, and are not in her stock as integrated systems. To create complex resource items (i.e. camera bodies and lenses as an integrated package), the user applies the *cartesian product* operator on `SLR cameras` and `lenses` (see Figure 14) to construct `TSR1` (see Figure 15). `TSR1` includes paths from both TSRs in an AND group and data for integrated photo equipment (i.e. camera bodies and lenses).

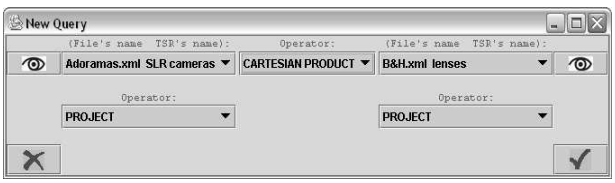


Figure 14: Applying *cartesian product* operator.

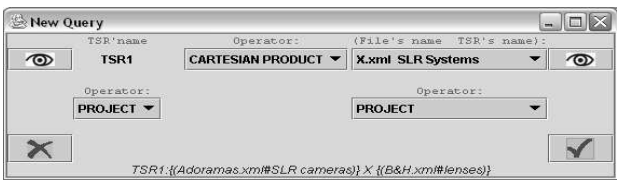


Figure 15: Result of *cartesian product* operator.

From the records of the resource item node of `TSR1`, the user keeps only those (a) referring to 'Canon' camera bodies and (b) having lenses that fit these bodies (see Figures 16 and 17). From all the attributes, the user keeps `cbrand`, `cmodel`, `lmodel` (see Figure 18).

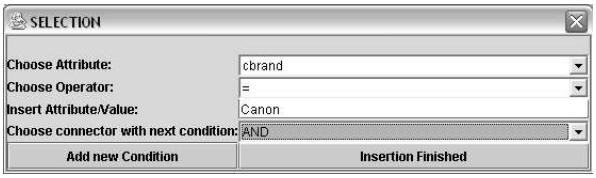


Figure 16: applying *select* operator: first condition.

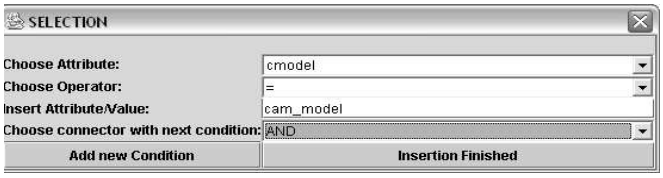


Figure 17: applying *select* operator: second condition.

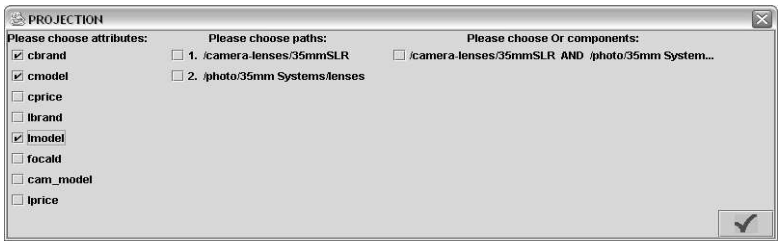


Figure 18: applying *project* operator.

Figure 19 illustrates `TSR1`. This TSR refers to integrated systems having 'Canon' bodies from Adorama and fitted lenses from B&H.

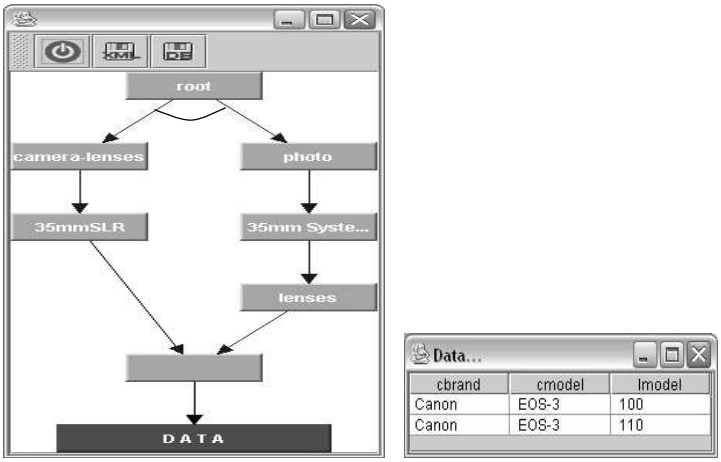


Figure 19: TSR `TSR1` for case 1.

To find integrated systems which are not in her stock, the user applies the `difference` operator between `TSR1` and `SLR systems`, resulting in a new TSR having systems with 'Canon' bodies from Adorama and lenses from B&H which are not in X's catalog. Since difference operates on two TSRs with the same set of resource item attributes, a project operator is necessary on `SLR systems` to keep only attributes `cbrand`, `cmodel` and `lmodel` before applying the difference operator. The resulting TSR (see Figure 20) has one OR component, showing that there is an integrated photo system, including a body 'EOS-3' and lens '110' not offered by X.

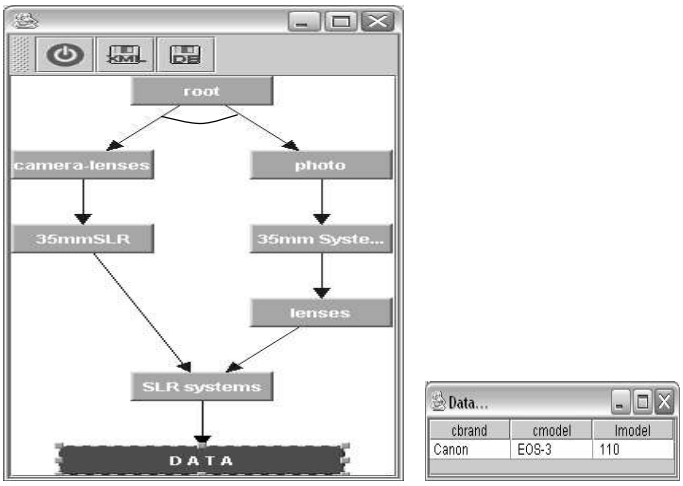


Figure 20: Resulting TST after the difference operation in case 1.

Having as a result only the lenses without the appropriate camera bodies, requires a projection operation which keeps only the attribute `lmodel` and the path `/photo/35mm systems/lenses` (see Figure 21).

Case 2. X now needs to build a new catalog with her own, old one, plus all integrated photo systems, having 'Canon' camera bodies from Adorama's catalog and lenses from B&H catalog, to compare their prices. To create complex resource items (i.e. camera bodies and lenses as an integrated package), the user applies again the *cartesian product* operator on `SLR cameras` and `lenses` to construct a new `TSR1`.

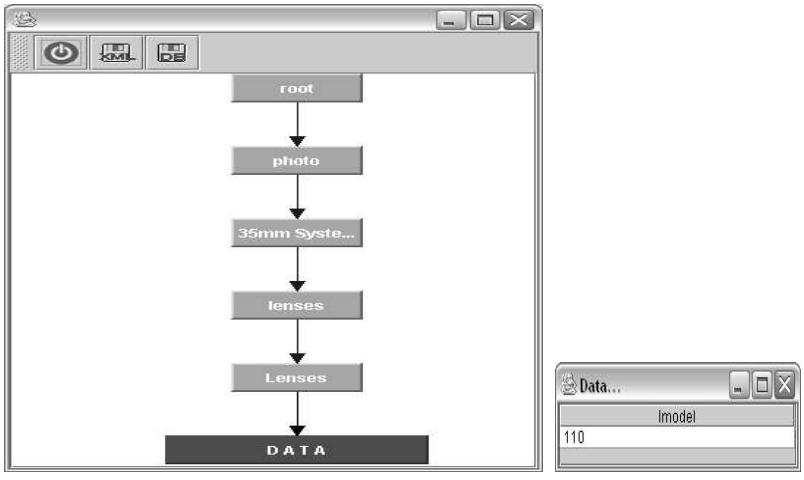


Figure 21: Final TSR for case 1.

From all records of the resource item node of `TSR1`, the user keeps only those records (a) referring to 'Canon' camera bodies and (b) having lenses that fit these bodies. From all the attributes, the user keeps `cbrand`, `cmodel`, `cprice`, `lmodel`, `lprice`. TSR `TSR1` is identical to the one in Figure 19, except that there are more attributes in its resource item.

The `union` operator on `TSR1` and `SLR Systems` results in the TSR `TSR2` (see Figure 22). `TSR2` has two OR components, showing all integrated photo systems from all three catalogs. It includes systems with 'Canon' bodies from Adorama and lenses from B&H, as well as systems from X.

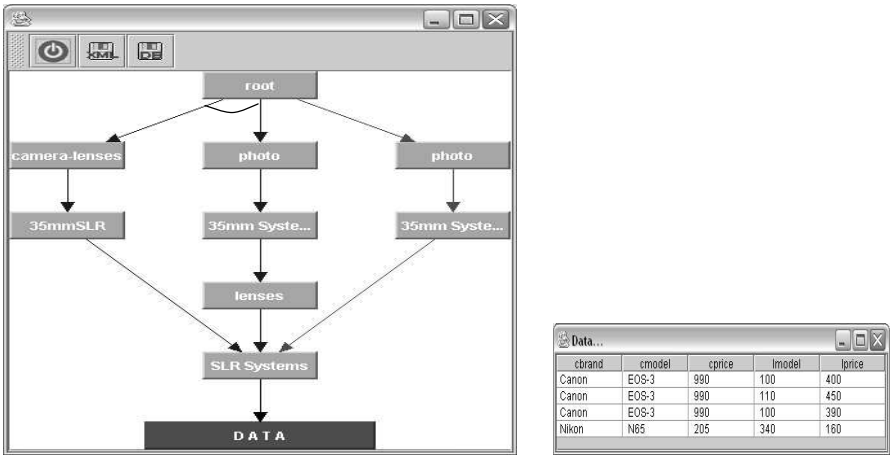


Figure 22: TSR `TSR2` in case 2.

Searching for paths that lead to lenses including nodes `photo` and `lenses` requires a selection operator (see Figure 23). A final projection will produce a TSR schema only for lenses (see Figures 24, 25).

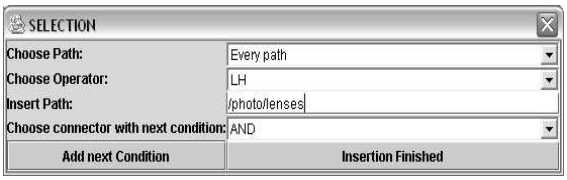


Figure 23: Applying *select* operator.

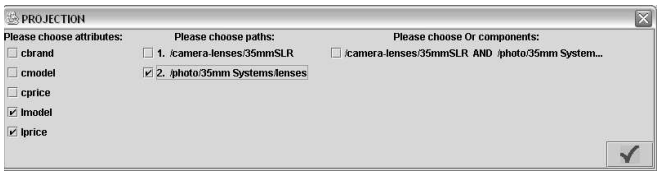


Figure 24: Applying *project* operator.

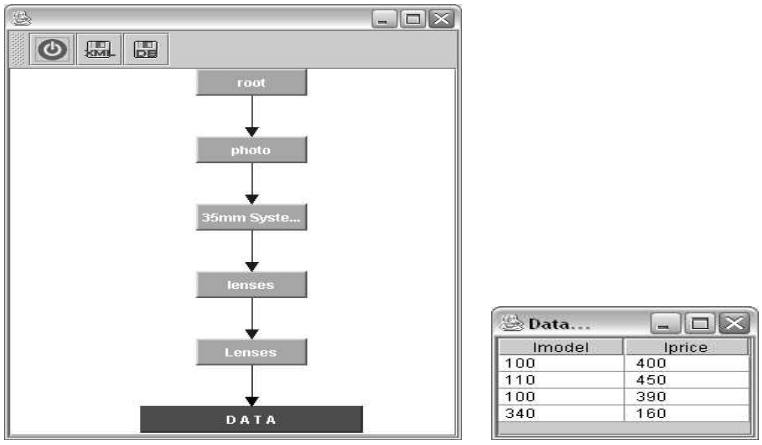


Figure 25: Final TSR for case 2.

6. Conclusions & Further Work

In this paper we described *PatMan*, a prototype visual database system to manage hierarchical catalogs. Such catalogs are represented with catalog schemas. Several catalog schemas can be combined, creating

tree-structured relations (TSRs). TSRs are modeling structures that emphasize the role of paths as knowledge artifacts. They capture the different ways of accessing data in a set of catalog schemas, and maintain alternative path-like pattern versions and complex patterns.

The *PatMan* system manipulates navigational path patterns and data from hierarchical catalogs in a uniform way, providing:

1. mechanisms to import structures from hierarchical catalogs coming in various forms (e.g. RDFs representation),
2. pictorial query-by-example capabilities based on the *PatMan*QL, a query language with the operators *select*, *project*, *cartesian product*, *union*, *intersection* and *difference* to manipulate TSRs from hierarchical catalogs.
3. visualization capabilities to explore navigational paths for hierarchical catalogs as well as the raw data organized in these catalogs.

We plan to extend the work presented in this paper along several directions. First, we will further explore the role of paths as knowledge artifacts in hierarchical catalogs, searching for useful comparison operators for paths. Also, we will introduce additional operators for path management. For example, one can identify the need for a join operator based on the select and cartesian product operators.

7. References

1. S. Abiteboul, P. Buneman, D. Suciu: Data on the Web: From Relations to Semistructured Data and XML, Morgan Kaufmann Publishers, 2000.
2. P. Bouros, T. Dalamagas, T. Sellis, M. Terrovitis: PatManQL, A language to Manipulate Patterns and Data in Hierarchical Catalogs, Proc. of EDBT PaRMa'04 Workshop, Heraklion, Greece, 2004.
3. A. Chaudhri, A. Rashid, R. Zicari: XML Data Management: Native XML and XML-Enabled Database Systems, Addison Wesley, 2003.
4. V. Christophides, S. Cluet, J. Simeon: On wrapping query languages and efficient XML integration, Proc. of the ACM SIGMOD Conf., p141-152, 2000.
5. J. Han, Y. Fu, W. Wang, K. Koperski, O. Zaiane: DMQL: A Data Mining Query Language for Relational Databases, Proc. of the SIGMOD'96 DKMD Workshop, Mondreal, Canada, 1996.
6. T. Imielinski, H. Mannila: A Database Perspective on Knowledge Discovery, Commun. ACM, 39(11), 1996.
7. H. V. Jagadish, L. V. S. Lakshmanan, D. Srivastava, K. Thompson: TAX: A Tree Algebra for XML, Proc. of DBPL Conference, p149-164, 2001.
8. L. Krishnamurthy, J. Nadeau, Z. M. Ozsoyoglu, G. Ozsoyoglu, G. Schaeffer, M. Tasan, W. M. Xu, Pathways Database System: an Integrated System for Biological Pathways, Journal of Bioinformatics, 19, 2003.
9. B. Ludascher, Y. Papakonstantinou, P. Velikhov: Navigation-Driven Evaluation of Virtual Mediated Views, Lecture Notes in Computer Science, 1777, 2000.
10. R.Meo, G.Psaila, S.Ceri: An Extension to SQL for Mining Association Rules in SQL, Data Mining and Knowledge Discovery, 2(2), p195-224, 1998.
11. E. Rahm, P. A. Bernstein: A survey of approaches to automatic schema matching, VLDB Journal, 10(4), 2001.
12. S. Rizzi, E. Bertino, B. Catania, M. Golfarelli, M. Halkidi, M. Terrovitis, P. Vassiliadis, M. Vazirgiannis, E. Vrachnos: Towards a logical model for patterns, Proc. of ER'03 Conference, 2003.
13. I. Rojas, L. Bernardi, E. Ratsch, R. Kania, U. Wittig, J. Saric: A database system for analysis of biochemical pathways, Silico Biology, 2(2), p75-86, 2002.
14. M. Terrovitis, P. Vassiliadis, S. Skiadopoulos, E. Bertino, B. Catania, A. Maddalena, Modeling and Language Support for the Management of Pattern-bases, Proc. of SSDBM'04 conference, 2004.