Heuristic data modeling in information systems

© Martin Davtyan

Bauman Moscow State Technical University
martin.davtyan@gmail.com

Abstract

The paper describes research-in-progress work on information systems that store semi-structured data and use heuristics to make hypotheses about possible data structure. Such systems are intended to be more user-friendly as data formalization is performed by answering simple questions.
Academic supervisor: Yuri Gapanyuk, gapyu@yandex.ru.

1 Introduction

As the information technology spreads over the globe, more and more people become involved in tasks of storing and exchanging information. Most of these people have no skills or time for programming or installing database software, neither these people can do data formalization needed. Blogging systems made it convenient for every person to share textual information or multimedia content. In some cases, like notes or shopping lists, the textual data representation is enough. But in other cases, people also need to structure their information somehow, to make it convenient to understand or share, for example, when storing small shop catalogues or club participant lists. There are also search benefits to that point, as having some data structure allows us to make search queries more meaningful than the full-text search queries. There are no tools allowing a non-professional user to set up a system to store and exchange semi-structured data. The requirements for such a system would be:

- Users should be able to store information and share it with other users.
- It should have a user-friendly interface for data structuring.
- It should also have a convenient search tool making use of the structure created.

In this article we propose an approach to constructing such a system.

Such a system would allow users with no preliminary knowledge of data structures or logic take full advantage of structured data, such as convenient navigation and powerful search. It would also free users from data formalization attempts when there is not enough data to formalize, but it still needs to be published.

The main challenge here is setting up the data structure. Making up a ‘schema’ implies proper formalization of the data, which is not always possible due to inability of a user. Formalization is also difficult if it is impossible to predict, what kind of properties or variables are about to be stored, before we get enough data to analyze.

This paper proposes a new approach for building user-friendly information system, letting a user store data in terms of objects of no predefined structure, and to simplify formalization of the stored data. Making propositions about a possible structure of data and asking user simple questions about it, we can use provided answers to construct a semantic-aware search in addition to a full-text search. Our hypothesis about ‘schema’ that fits the stored data can be also used for effective visual representation of this data on the Web, allowing more convenient way to share it. That way we can make use of data structure without forcing user to explicitly set it up for us.

2 Existing systems

There are two ways to build an information system required that people use now. The first way is to store information in tables. There is a variety of convenient spreadsheet software products providing intuitive way of data formalization, and there is no problem with adding new columns if we encounter new fields. The problems here are:

- The formalism user provides when setting up the column names is not used for searching, the search is still full-text. If there are ways to use complex queries in particular spreadsheet software system (i.e. queries that use the ‘schema’ provided by user), they are way too complicated for an average computer user to understand.
- Sharing is complicated, as users often end up emailing spreadsheet files to each other, copying rows from one table to another.

The second way of solving the problem is to invite programmers and data modelers to build a website. The user explains how exactly the data he intends to store looks like, and data modelers make up a database, in

Proceedings of the Spring Young Researcher's Colloquium On Database and Information Systems SYRCoDIS, Moscow, Russia, 2011

most cases a relational one, that is capable of storing that data. Programmers then add a web application that allows user to create, edit and share that data on the Web. The problems of this approach are:

- Programmers and data modelers' work requires money and time.
- Any change in a data scheme will require calling programmers and data modelers again.

3 System overview and motivating example

In this section we propose a model of an information system that meets the requirements stated in Introduction.

The simplest and fastest way to share information with someone is to use the Internet, so we propose a system to have a website interface. User should be able to create objects – named key-value mappings of no predefined structure (as mentioned above) – and store them in a personal set. To illustrate the related workflow, let us consider an example.

Suppose the user intends to store car rental data and share it with current and prospective clients. There is only one basic action a user can perform when starting to work with the system – create an object. The user describes a car in terms of properties: {'transmission' : 'manual', 'mileage' : '30000km', 'horsepower' : '101'}. He also gives the object a name – 'Skoda Octavia I Tour' – and saves it. After saving the object, he is offered to create another one with the same property names, so he now only needs to fill in property values. As the quantity of user's objects increase, user gets exposed to the first question. The system outputs a number of objects (most likely with the same property set) and asks a user if it is possible to give all these objects a common name. The user's answer, if positive (e.g. 'Vehicle'), is used as a name of a new class. This is how classes are created.

Classes emerge in the system as the user inputs information. Suppose we have two classes registered – 'Vehicle' and 'Truck', only differentiating in a 'cargo weight' property the 'Truck' class has. The system outputs a number of objects of both classes and asks whether all 'Truck's are 'Vehicle's. If user confirms, the corresponding subclass relationship is registered.

By the time the user has published enough information, he is also being asked, whether the 'horsepower' property is always a number. If true, this property is registered to be of a certain data type.

All the data structure information the system gets from user's answers is used for user interface optimization. Being given an object classification, the navigation through user's objects is arranged accordingly. User also gets organized shortcuts for creating objects of registered classes. Data type information allows validation of newly created objects if they belong to a registered class.

The example shows how the system takes advantage of formal data structure without forcing user to do the formalization. Asking simple questions and providing related object examples to assist answering, data structure is exposed to the extent that is usable for crucial interface and search improvements.

4 System architecture and workflow

4.1 Input data and storage

As questions of building user interface and developing web applications are not in the scope of this paper, we shall make assumptions about what data we get from user. We assume that user provides objects, i.e. lists of key-value pairs, where key values are called property names, and values are called property values. We can state the data we store is semi-structured [1].

An intuitive solution for storing objects discussed above is a document-oriented database, such as CouchDB or MongoDB, as the structure of 'object' described above strictly corresponds to the structure of 'document'. In addition, using document-oriented database will make it easy to store data in a cloud storage. Another solution could be an XML database.

4.2 Workflow

We have no data model predefined when user starts to push data into the system, and, accordingly, our database has no schema. But as the user inputs some data, we make hypotheses about its structure, which we then check by asking the user simple yes/no questions.

One can think of information about data structure we get from user as a schema [2]. As schema in relational database systems is defined before storing actual data, and should be supported by DBMS, we propose a term 'heuristic schema' for data model information that is built according to data. Heuristic schema can emerge only after the data is stored, and it is only supported on application level.

4.3 Making hypotheses

Suppose we have a number of objects of in our database, and consider different hypotheses can be made when analyzing their structure. The names examples of such hypotheses are given below:

- Heuristic class. If a number of objects have exactly the same property names, they are probably in the same class. The problem of finding classes is discussed below.
- Heuristic subclass. If heuristic class A has property names P_1, P_2, P_3 … P_n, and class B has property names P_1, P_2, P_3, … P_n, … P_m, heuristic class B is a subclass of heuristic class A.
- Heuristic data type. If property values stored under property name N are always numbers, property N is probably numerical. The same idea can be used with any popular data types, as strings or URLs. In general case one can think of any constraint that applies to values of N property as a data type [3].

Each of these hypotheses triggers asking user a corresponding question, as his answer causes refutation or confirmation. If a hypothesis is confirmed, user is

also asked to give a new heuristic data structure a name, to make it usable in further data input or search queries.

4.4 Using string distance for class hypotheses

Making heuristic class hypotheses only in case of exact match of all the object property names is useless in case of user's misspellings. This problem can be solved using string distances for property names.

As two objects belong to the same class only having the equal number of properties and property values have no impact on making class hypothesis, let us consider the sets of N property name strings. Dealing with misspelling problem, we use Damerau–Levenshtein distance for measuring difference between the strings. Damerau–Levenshtein distance is a string metric given by counting minimal amount of insertions, deletions or transpositions to transform one string to another. These operations correspond to more than 80 percent of human misspellings [4].

With distance between individual property name strings defined, we now need to define a similarity measure between string sets, which represent objects. Let us consider two sets A = $\{a_1, a_2, \ldots, a_N\}$ and B = $\{b_1, b_2, \ldots, b_N\}$ and Damerau–Levenshtein distance function $d(a_i, b_j)$. Given a particular mapping F: A → B, similarity between two sets can be calculated as follows:

$$S = \sum_{i=1}^{N} d(a_i, F(a_i)) \qquad (1)$$

As objects having the same property names belong to the same class regardless of the order in which properties are listed, we need to find the least possible distance between two sets. The problem of finding a mapping function F so that S is minimized is known as an assignment problem [5], which can be effectively solved by Munkres algorithm [6]. After using the algorithm to determine F, we use the corresponding S as a similarity measure. Setting up a threshold value for this measure will determine whether the class hypothesis should be made.

4.5 Using heuristic schema

These are the most important benefits of having schema in semi-structured data storage:

- Better user interface. After making user passively define a class and give it a name, we are able to create a template for this class. User now just has to choose a class for an object, instead of manually inputting property names.
- Classes can be also used for the Web representation of the stored data, with class hierarchy acting like a catalogue tree for navigation through site.
- Heuristic classes and data types can be used for data search using semantics, not only full-text search. For example, search query can set constraints on a certain heuristic field (e.g. number field), or certain class.

5 Related work

Ideas discussed in this paper are connected with Semantic Web technologies in a way they also intend to add some additional structure to the weakly structured data sets [7]. Semantic Web has a great stack of technologies for storing and exchanging semantic data, like OWL, RDF and others, that can be used as a formal representation of heuristic schema.

6 Future work

We are currently working on a prototype of the described system. It is planned to:

- Use semantic web standards for schema representation.
- Use current research results in automated ontology construction for heuristics.

However, the main direction of our research is associated with cluster analysis. It is planned to define similarity on the whole object set, regardless of property amount. This will enable using clustering algorithms, which are useful for hypothesis construction:

- Hierarchical clustering results in a dendrogram, which can be used to build a taxonomy hypothesis.
- Validation within unclassified object set – error hypothesis proposed for objects not belonging to any existing cluster.

7 Conclusion

The described system acts like a compromise between easy and convenient text storage and powerful and searchable fixed schema databases. It allows user to publish data as soon as possible, and it also has all benefits of a structured data storage.

References

[1] Peter Buneman. *Semi-structured data. Tutorial, PODS '97* http://www.cis.upenn.edu/~db/abstracts/semistructured.html

[2] http://en.wikipedia.org/wiki/Database_schema *Database schema - Wikipedia, the free encyclopedia*, February 2011

[3] Luca Cardelli, Peter Wegner. *On Understanding Types, Data Abstraction, and Polymorphism*, 1985

[4] Fred J. Damerau, *A technique for computer detection and correction of spelling errors, Communications of the ACM*, 1964

[5] http://en.wikipedia.org/wiki/Assignment_problem *Assignment problem - Wikipedia, the free encyclopedia*, April 2011

[6] Geoffrey C. Fox, Roy D. Williams, Guiseppe C. *Messina, Parallel Computing Works*, 1994

[7] Pascal Hitzler, Markus Krötzsch, Sebastian Rudolph. *Foundations of Semantic Web Technologies*, 2009