

The tagged collection: an alternative way of organizing a collection of bookmark-like items and its integration with existing web browsers

Enrico Zini `zini@cs.unibo.it`

July 22, 2002

1 Abstract

This paper shows the problems that arise when hierarchical collections of items like the bookmarks of a web browser evolve over time, and proposes an alternative, yet simple organization to solve those problems. It then proposes and analyzes a viable method to integrate flawlessly this organization into any existing web browser.

2 Introduction

Many applications often have the need to let people build a collection of items like web browser bookmarks. People use these collections to store interesting items they find, then forget them until they need them again, sometimes a long time later.

While applications have no technical problems in keeping this type of data, problems arise when people need to search and retrieve items from the collection.

Usually, the number of these items easily grows rapidly too large to let the user pick them up from a list, and the nature of the items makes basic techniques such as alphabetical sorting unsuitable as the main way for locating the data.

The solution usually adopted is to let people group their items into folders and organize their folders in a hierarchy, so that they can locate what they wish by proceeding “from the general to the particular”.

Other more complicate and interesting solutions like the one presented in [2] try to cluster items automatically or semi-automatically, at the risk of reducing too much the control over the structure of the collection, as when someone else goes on sorting our desks.

3 The problem

Hierarchies usually are built to mimick the importance that a person gives to the various qualities of an item. For example, I could think of storing a bookmark to a E-Zine about computer music equipment under “Computer”, then “Music”, then “Reviews”.

This is great, because it lets people model their collection to their needs, but it can become fragile, since the estimation of the importance of the many qualities of an item usually change over time:

- it can change based on relationships with other items in the collection (I could collect many other bookmarks about various reviews and decide that storing my E-Zine under “Reviews/Music/Computer” suits me better)
- it can change along with the interests of the person (I could get more interested in music and decide that “Music/Computer/Reviews” is better).

While people can perform such rearrangements when they feel that the structure of their collections does not suit their needs anymore, such rearrangements are usually quite boring and time consuming.

Another problem is deciding when to store an item when an item is found. The decision has to be taken quickly, since bookmarks are usually taken when an interest strikes during other operations, but the attention should not have to shift too much from that operation.

With hierarchies, the decisions about where to store an item can be difficult: the web page about a weighted master keyboard should go under “Music/Instruments” or under “Computer/Music/Instruments” ?.

A solution usually offered is to store the item in more than one place in the hierarchy, but then storing the item becomes a difficult task, since it would involve figuring out all correct places in all the hierarchy where the item can belong (if I’m navigating the Internet looking for music information, I might not recall I had a “Music/Instruments” section under “Computer”; if some time later I’ll consider buying some musical appliance for my computer, I might not even think to search outside the “Computer” folder).

Besides storing the data, people want to find the elements they have stored. To do this, they usually navigate their collection until they find the required item. Furnas, in [1], describes four properties that can allow an information structure to be effectively view navigable, that is, like in the case with the current ways to move in a collection, to navigate selecting informations in the current view of the structure.

The four properties are:

EVT1 The number of outgoing links must be “small” compared to the size of the structure

EVT2 The distance between pair of nodes in the structure must be “small” compared to the size of the structure

VN1 Every node must contain some valid information (called residue) on how to reach any given other node

VN2 The information associated to each outgoing link (called outlink-info) must be small.

The usual hierarchy normally fulfills rules EVT1, EVT2 and VN2: EVT1 and EVT2 are usually limited by people when building the tree, since it’s common sense that breaking them would cause a mess. Rule VN2 is respected, since the outlink info is just a name or very short phrase.

Property VN1, however, is usually difficult to maintain, as shown in the examples above. This means that collections manually sorted in hierarchies are not likely to be Effective View Navigable, that is they are difficult to navigate without external aids.

4 An alternative approach

The problem with hierarchies has been identified in that they follow the importance given by a person to the qualities of an item, and this importance is not a static quality.

A better approach would be to structure the collection based just on the qualities of its items: while a E-Zine on computer music could go in “Music/Computer/Reviews” or in “Reviews/Music/Computer”, it always concerns “Music”, “Computers” and “Reviews”.

Instead of being filed in one or more nodes of a hierarchy, nodes could be stored in some kind of “flat” structure, tagged with zero or more categories.

In that way, storing an item in the collection would require adding some attributes picked from a list or occasionally created new, and maintaining the collection would require adding or removing tags when some more insight on the items is acquired or when the number of items stored increases.

An hypothetical search method could consist on a window split in two, with one half allowing the user to require or exclude the various tags and the other half showing the matching items.

5 Deploying the alternative approach

While this could be an interesting idea for a specialized application to store bookmarks or similar informations, it could be interested to use the tagging method with current software.

Existing web browsers all use a hierarchical organization. If this hierarchy could be mapped to a tagged collection, and the tagged collection rearranged into a hierarchy complying with Effective View Navigation rules, we would have found a way to immediately and transparently improve all existing software.

5.1 From the tagged collection to the hierarchy

Let’s analyze a possible algorithm to convert a tagged hierarchy into a hierarchical collection:

- 1 Define the “cardinality” of a tag as the number of items in the collection having it in the tag set
- 2 Add the items with no tags to the tree
- 3 Delete the items with no tags from the collection
- 4 If there are no more items left, goto 9
- 5 Choose the tag T with the highest cardinality and add it to the nodes in the tree, as a new folder called with the name of the tag

- 6** Insert into the new folder all associated items
- 7** Remove from all the items in the collection the tag T
- 8** Goto 3
- 9** For all newly added folders, repeat this algorithm on the collection of their items

The resulting tree is EVT2, since its maximum depth is bounded by the maximum number of tags assigned to a single item, that is usually small.

The resulting tree is VN1, since in every node there is at least one of the categories of the item or the “All the rest” category pointing to the parent, and when we have chosen all the categories of the item we can find it.

The resulting tree is VN2, since the outlink-info is the tag name, usually a name or small phrase.

EVT1 remains to check. All child nodes refer to a subset of the items referred by the parent, so the node with the maximum number of outgoing links must be the root. The number of outgoing links in the root is exactly the number of all different tags found in the collection; it remains to see if this number is small.

Supposing to have an uniform distribution of items in all possible combination of tags, and supposing to have no more than 10 items for each possible combination, if n is the number of tags, the maximum number of items that can be stored in the collection is $10 \sum_{i=1}^n \frac{n!}{i!(n-i)!}$. With $n = 20$, a suitable number of tags in our root node, we could store up to 1'048'575 elements!

This number is great, but the scenario would rarely reflect real situations: for example, a programmer might like to tag some links by computer programming language, adding a tag for every computer language he knows; then tag music by genre, adding a tag for every music genre he likes; then proceed with movie types, and so on.

In reality the number of tags cannot grow too big, or it would imply having overtagged the items either by assigning too much tags to every item or doing unnecessary catalogation resulting in very few items per tag; however, our root node is likely to fill up fast enough not to satisfy EVT1.

This problem can be solved by refining the algorithm. Let's introduce the concept of “implication” between tags: one tag implies another if all items tagged with the first are also tagged with the second. Between step 7 and step 8, let's insert step 7a:

- 7a** Remove from all the items in the collection all the items implied by tag T

This should keep the root node small as soon as the collection is tagged in a sane way: all our programming languages can be implied by a tag “Language”, all our music genres can be implied by a tag “Music” and so on.

The modified algorithm is trivially still EVT2, still VN2 and has become EVT1. Is it still VN1? Yes: implication of A by B leaves in B some residue information of A; because a tag is either present in a node or implied by a tag present in a node, it still has a residue in every node.

5.2 From the hierarchy to the tagged collection

Now that we can generate a hierarchy that is effectively view navigable, we need to recreate the tagged collection from it. A suggestion on how to do it comes straight from one of the the examples above.

E-Zines on computer music could go in “Music/Computer/Reviews” or in “Reviews/Music/Computer”, but they always concern “Music”, “Computers” and “Reviews”. This suggests taking the list of parent nodes from an item in the hierarchy and using it as the list of tags for the items.

Would this algorithm introduce some loss of information?

In a generated hierarchy without pruning of implied tags, no information would be lost, since the list of parents of an item would always be a permutation of its tag set.

In a generated hierarchy with pruning of implied tags, an item will appear at least one for every one of its tags, so the information can be rebuilt by merging all the parent sets of every instance of an item.

Since an item would never have a parent corresponding to a tag that is not in its tag set, we have established a biunivocal function between the tagged collection and its tree representation.

6 Conclusion

I have shown how a hierarchy is not an optimal way to organize a collections of items, and I have shown a possible alternative way to do it, the tagged collection. I have then shown an algorithm to store a tagged collection in the bookmark hierarchy of any existing browser, with the generated hierarchy being Effectively View Navigable.

This is enough to produce a tool to manage bookmarks in a smart way and interact transparently with any given browser, with no need to store informations in external databases.

7 Future work

The generated hierarchy is good to find informations, but it is not likely to be suited for maintaining it.

Maintaining a tagged collection would mean to:

- Add an item
- Add tags to the tag set of an item
- Remove tags from the tag set of an item
- Remove an item

Adding a bookmark is straightforward, as it can be appended to the root node; many browsers already have a one-click command to do it.

Tagging newly added bookmarks can be easily performed in two ways:

1. Move or copy the item in a suitable subtree of the hierarchy

2. Create a small temporary tree with branching factor 1 to represent the tag list, put the item at the end of the tree and invoke the sorting algorithm.

However, method 1 makes the user loose the concept of a tagged collection and method 2 lacks elegance and intuitiveness.

Removing a tag from the tag set of an item cannot be easily performed, since it would require to move it outside all the instances of folders named after that tag but keep it inside all instances of folders named after the other tags in its tag set.

Deleting an item from the collection cannot be easily performed, since it would require deleting it in all the places it could appear.

It remains to see if there can be simple tree operations that can intuitively be mapped into tagged collection operations. One possible idea could be using folders with special names like “Trashcan” where to put an instance of items to be deleted and have the sorting algorithm treat them in a special way.

Future work will be to explore the ways of maintaining tagged collections, either using special tree operations or designing ad-hoc interfaces.

Another field of exploration could be optimizing the generated hierarchies: for example, it could be tempting to “flatten” a folder containing one or two items by storing them directly in the parent node and removing the folder. However, this optimization would cause a possible loss of information when converting the hierarchy back to a tagged collection. Research could look for other optimizations, and other ways to store information in the hierarchy to make optimizations possible.

8 Acknowledgements

Thanks go to prof.Fabio Paternò for having introduced me to Information Visualization, to prof.Cesare Maioli and prof.Renzo Davoli for giving me very useful advice and to my father for late-evening calculation assistance.

References

- [1] G. W. Furnas. Effective view navigation. pages 367–374, 1997.
- [2] Israel Z. Ben Shaul Yoelle S. Maarek. Automatically organizing bookmarks per contents.