University of Dayton
eCommons

Computer Science Faculty Publications | Department of Computer Science

6-2006

# Interacting with Web Hierarchies

Saverio Perugini
*University of Dayton*, sperugini1@udayton.edu

Naren Ramakrishnan
*Virginia Polytechnic Institute and State University*

Follow this and additional works at: https://ecommons.udayton.edu/cps_fac_pub


Part of the Databases and Information Systems Commons, Graphics and Human Computer Interfaces Commons, Numerical Analysis and Scientific Computing Commons, OS and Networks Commons, Other Computer Sciences Commons, Systems Architecture Commons, and the Theory and Algorithms Commons

eCommons Citation

Perugini, Saverio and Ramakrishnan, Naren, "Interacting with Web Hierarchies" (2006). *Computer Science Faculty Publications*. 24.
https://ecommons.udayton.edu/cps_fac_pub/24

This Article is brought to you for free and open access by the Department of Computer Science at eCommons. It has been accepted for inclusion in Computer Science Faculty Publications by an authorized administrator of eCommons. For more information, please contact frice1@udayton.edu, mschlangen1@udayton.edu.

# Interacting with Web Hierarchies

Saverio Perugini[†] and Naren Ramakrishnan[‡]

[†]Department of Computer Science, University of Dayton, OH 45469–2160
E-mail: saverio@udayton.edu Tel: (937) 229–4079 Fax: (937) 229–4000

[‡]Department of Computer Science, Virginia Tech, VA 24061–0106
E-mail: naren@cs.vt.edu Tel: (540) 231–8451 Fax: (540) 231–6075

June 23, 2006

**Abstract**

What makes one hierarchical website easier to interact with than another? Why are some interface designs frustrating for certain information-seeking tasks? How can we realize flexible, adaptive, and responsive dialogs in user-website interactions? In this article, we survey some of the emerging designs for hierarchical websites and present vocabulary for thinking about faceted website design.

# Introduction

Hierarchies are ubiquitous on the web, from store catalogs, to news services, to community classifieds. They are one of the most natural ways to organize, present, and navigate online information and, yet, can be a major source for frustration when the hierarchy does not mirror our mental conception of information seeking. As a result, interface designers have developed many variants of the traditional drill-down motif, all of which aim to get the user to their information of interest quicker.

In this article, we survey some desired features of interaction with web hierarchies[1], in particular support for flexible navigation orders, for exposing and exploring dependencies, and for procedural information-seeking tasks. We also present several (adaptive) interface designs which support these features. Rather than advocate the superiority of specific interfaces, our goal here is to present vocabulary for thinking about hierarchical sites. The effective realization of the designs presented here will depend not only on the nature of information-finding tasks to be supported but also on the type of content being delivered through the site.

# Desired Features for Interacting with Hierarchies

## Flexible Navigation Orders

The *flexible navigation orders* feature means that inputs can arrive in any (user-specified) order. To help illustrate this feature, consider a user visiting an online car shopping site, such as autotrader.com:

> *Scenario 1*:
> "I am here to buy a car. The most important criteria for me are price, safety rating, and color, in that order. What's available?"

To conduct such a search online, the user will find helpful a hierarchy organized by price at the top-level, safety rating at the next, and color subsequently. We say that such a hierarchy dictates a *total order* over automobile facets, since the price must be specified before safety rating, which must be done before color. However, the user would be hard pressed to find such a classification; most automobile sites we surveyed organize their inventory alongside more traditional facets such as make, model, and year. In the absence of a suitable totally-ordered hierarchy, this user would benefit from an interface which *enumerates* all possible navigation orders or an ability to *generate* their own (totally-ordered) classification tree for automobiles.

### Enumerated vs. Generative Interface Designs

Enumerative interfaces support flexible navigation orders by exposing *all* of the individual *choices* for unspecified facets at every level [6] and are exemplified in Epicurious.com. For instance, in Fig. 1 (left) all of the individual choices for each of the classification's facets (e.g., main ingredient, cuisine, preparation method, season/occasion, and so on) are enumerated. From Fig. 1 (left) the user selects 'bake' (a preparation method) and thus a choice for preparation method is unavailable in Fig. 1 (right). We can think of enumerated interfaces as multiple totally-ordered hierarchies, one for each for the $n!$ possible navigation orders, where $n$ is the number of facets considered in the classification. So as to not overwhelm the user with all such orders, some sites, such as Kelley Blue Book online at kbb.com, present a choice of only a few facets at each level, yielding a partially enumerated hyperlink structure.

[1] We use the term *hierarchy* in a broad sense here, namely a multi-attributed classification, not necessarily a tree-structured organization.

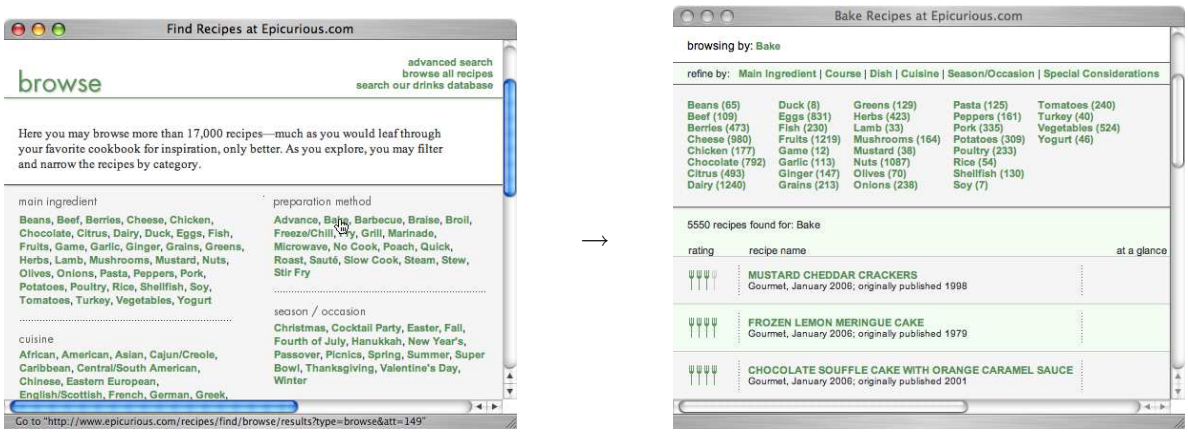



Figure 1: An enumerative interface: the recipe classification at epicurious.com.

There are alternate enumerative interfaces designs for hierarchies. For instance, pull-downs menus, such as those used for browsing digital cameras and camcorders at sonystyle.com (see Fig. 4), are enumerative, but hide the enumerated choices behind UI components. Other interfaces, e.g., that for Apple *iTunes*, present all facets and choices on a single page, typically in tabular form, and explicitly relay back how attribute values are pruned during an interaction. In a sense, these interfaces are enumerative but by supporting browsing information *within* a page, the illusion of traveling through hyperspace is lost. As the number of classification facets increases (automobiles can have $>$20 individual attributes) such interfaces can become cluttered, raising issues of screen real estate. We shall have more to say about pull-down menu and table-based interfaces in the following section.

In contrast to enumerative interfaces, generative interfaces support the construction of a totally-ordered hierarchy for subsequent browsing. Fig. 2 illustrates how such an interface might work. Here, the site requests a navigation order first and subsequently generates a totally-ordered hierarchy conforming to it. Such order specification can be done *a priori* (as in Fig. 2) or incrementally, depending on the task to be supported. Consider, for example, the following slightly different information-seeking scenario:

> *Scenario 2*:
> "I am here to buy a car. I'm only interested in fuel-efficient cars — those that yield greater than 40 miles per gallon — and would like to begin my search from there."

An interface to support such a task might permit the user to specify the navigational order of the classification's facets *on-the-fly*. Such an interface meshes browsing the hierarchy with specifying its navigation order and has been described metaphorically as '[laying] down new track to suggest useful directions to go based on where we have been so far and what we are trying to do' [5]. A generative approach is reactive to the user's directives and organizes subsequent pages so as to reconcile what is left of the hierarchy with what the user desires to do next. See [3] for a discussion of other elements of a generative approach.

An alternate interface for supporting scenarios such as those in this section is an out-of-turn interface, introduced and described in [10] (see Fig. 3). An *out-of-turn* (OOT) interface lies somewhere between an enumerative and generative interface. Unlike interfaces based on facets, pull-down menus, or a table, it does not enumerate all of the individual choices at each level. Unlike a generative interface, it does *not* permit the user to *browse* in any order. An OOT interface simply empowers the user to supply a (out-of-turn) value for a facet or facets alternate to that solicited on the current webpage when the user does not like the

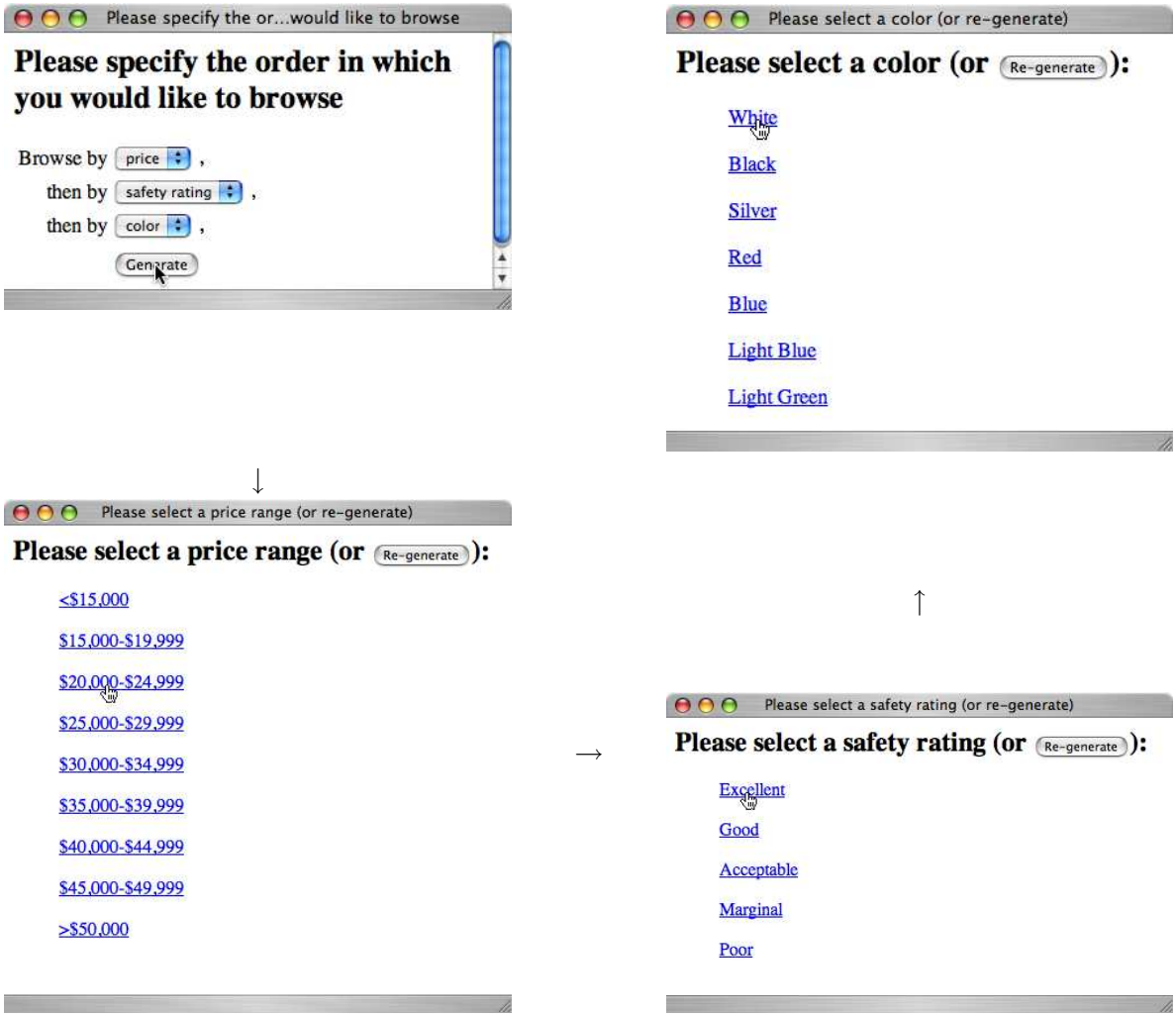



Figure 2: Use of a generative interface to support *Scenario 1*. (top left) First, the user specifies that she desires to browse the automobiles by price, safety rating, and color, in that order. A totally-ordered classification is then generated and the user progressively clicks on hyperlinks labeled '$20,000–$24,999' (bottom left), 'excellent' (bottom right), and 'white' (top right) to retrieve a page listing cars with these features or presenting additional facets of classification, e.g., make and model.

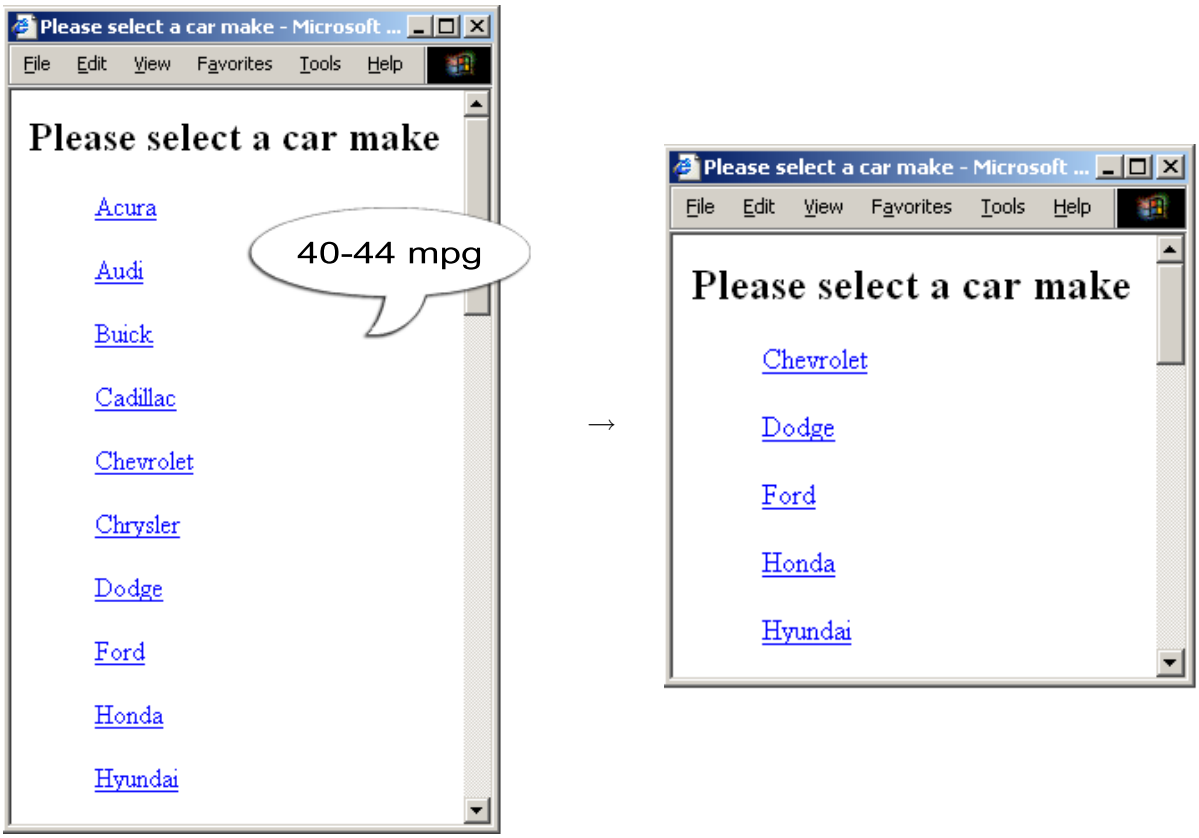



Figure 3: Use of an out-of-turn interface in a voice-enabled browser. (left) The user decides to not pursue any of the presented hyperlinks and instead speaks '40–44 mpg' out-of-turn. (right) Processing the out-of-turn input results in a reduced set of choices, because some makers do not fit the mileage specification.

current organization. We can permit the user to supply input out-of-turn by speaking to the browser through a voice interface (see Fig. 3) [8]. Alternatively, we can support OOT interaction using a toolbar embedded into the browser (see Fig. 5) [10]. Since an OOT interface does not enumerate all facets of classification at any level, it is important to provide users with *meta*-inquiry capabilities to inquire which attributes are left unspecified. However, observe that out-of-turn interaction does not imply free form input, only the ability to communicate input that is normally solicited further into the interaction.

Observe that the nature of content presented in a hierarchy affects which interface designs may be used to interact with it. Some content is inherently *faceted* meaning that when presented in a hierarchy, each level corresponds to a facet of information assessment; main ingredient, cuisine, and preparation method in epicurious.com or make, model, and year in kbb.com. On the other hand, large directories of links to websites, such as Yahoo! and the Open Directory Project (ODP) at dmoz.org, present *unfaceted* content, where there is not a one-to-one correspondence between hierarchy levels and facets. For example, the top-level of ODP presents hyperlinks labeled 'Arts,' 'Music,' and 'Sports' which correspond to nothing more specific than a topic. Since there is no concept of a facet, flexible interaction with these structures is typically achieved using purely navigational links. For instance, ODP offers shortcuts (links whose target is a page deeper in the hierarchy), backlinks (to pages at levels higher than the current), and multiclassification links (which bridge otherwise unconnected, but seemingly related, topics) to allow users to circumvent the rigid

nature of the site's static navigation structure. Notice that, with the exception of an OOT interface, none of the interfaces described here can be used to interact with an unfaceted hierarchy. (In order to construct an enumerated interface, the designer must enumerate all possible orderings of facets, again, requiring their existence. Similarly, to use a generative interface, the user must be able to specify an ordering of facets, thus requiring their existence.)

In summary, some interfaces force the user to supply values for facets in an order completely predetermined by the hierarchy. Other interfaces, e.g., an out-of-turn interface or an exhaustively enumerated interface, give the user *carte blanche* to communicate inputs in any order they see fit. Still other sites, such as Project Vote-Smart (vote-smart.org), provide a combination of the two, either enforcing order at the beginning of the interaction and permitting flexibility at the end, or vice versa, or mixing and matching both to create novel interaction experiences. We must also realize that sometimes it is desirable or necessary to design for retaining at least partial control over input order. For example, several sites, such as those for weather reports, require a zip code from the start to even begin interaction with the user.

## Exposing and Exploring Dependencies

Let us now turn to a different consideration when designing interfaces to hierarchies. In a given website, there are likely to be several dependencies between individual selections and it is desirable to expose these relationships to site visitors. At a basic level, progressive drilling down and retracing paths does indeed expose such relationships (e.g., upon clicking a model choice for car, the user might notice that the model does not meet his mileage requirement, so he backtracks and tries other selections). Obviously, we wish to do better, without back-and-forth interactions on the part of the user.

To illustrate what we are alluding to, refer again to Fig. 3. Here when the user says '40-44 mpg,' choices such as Acura are removed, since these cars do not meet the mileage specification. Such hyperlink pruning on the current page provides immediate visual feedback about dependencies. The *exposing and exploring dependencies* feature means that the interface obeys or exploits the dependencies implicit in the hierarchy and provides facilities to help expose them to users. Such web functional dependencies range from the simple 'every Civic is a Honda' tautologies to less salient, but more interesting, relationships such as 'all cars with a safety rating of 3.5 or higher achieve $<$20 mpg on average' or 'the Honda Civic Hybrid doesn't come in the color red.'

We can obtain such web functional dependencies from either domain knowledge or a data-driven analysis of the hierarchy (e.g., we can reason that since *none* of the paths through the classification containing hyperlinks labeled 'Honda' and 'Civic Hybrid' contain a hyperlink labeled 'red', the dependency mentioned above must hold). Techniques from association-rule mining [1] are relevant here. We can summarize functional dependencies by using '$\rightarrow$' to denote 'implies' and a $\neg$ to denote 'negation.' For instance, 'Honda $\rightarrow \neg$ Toyota', 'Civic $\rightarrow$ Honda', '$\{$safety rating $> 3.5\} \rightarrow \{<$20 mpg$\}$', and '$\{$Honda, Civic Hybrid$\} \rightarrow \neg$ red'.

### Reactive Menus and Query Expansion Interfaces

Some commercial websites now explicitly recognize that dependencies are ubiquitous in hierarchies and provide facilities to exploit them. For example, the Kelly Blue Book has a search for make by model name which invokes web functional dependencies of the form *model* $\rightarrow$ *make* which help lead the user to a desired automobile. Another simple method exposing dependencies to users, which is an extension of the exposure implicit in the hyperlink pruning via out-of-turn interaction mentioned above, is through the reduction of

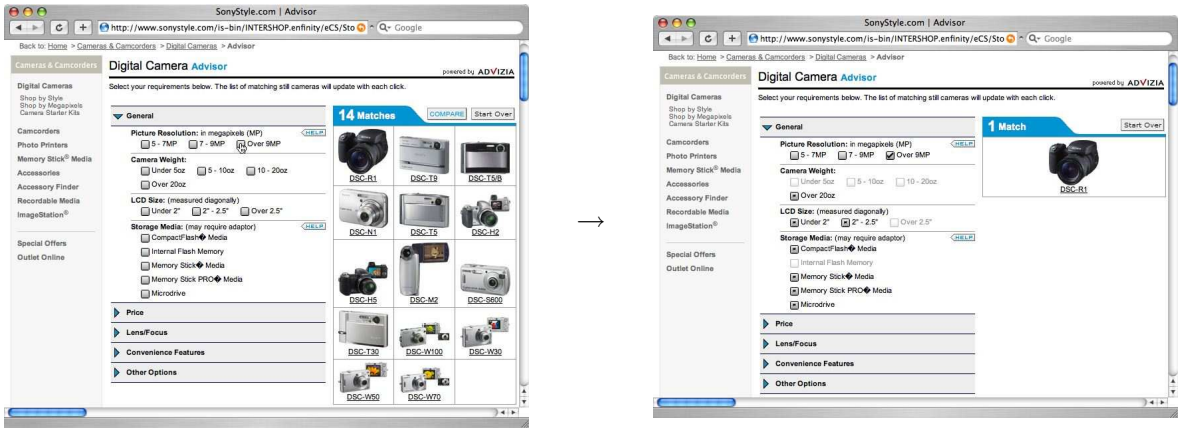



Figure 4: The *Sony Advisor* reactive menu for exposing and exploring dependencies. (left) From the start there are 14 digital cameras available and the user checks the box specifying cameras with a resolution over 9 megapixels (MP). (right) Only one camera meets the criterion – the DSC–R1 model. Through this interaction the site reveals the '9MP → DSC–R1' dependency to the user, as well as several others such as '9MP → ¬ DSC–T9.'

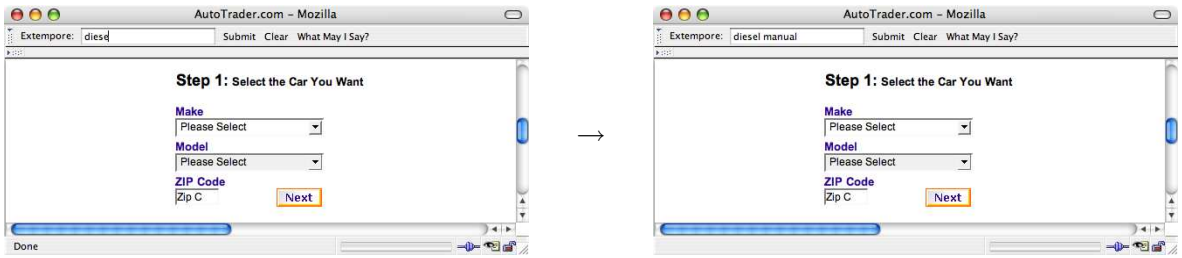



Figure 5: Use of a real-time query expansion interface (see toolbar at top of browser) to pursue the information-seeking goal of *Scenario 3*. (left) User types 'diese' out-of-turn (when site is soliciting for car model/make). (right) As soon as the user enters the 'l', thereby completing the specification of the term 'diesel,' the query is automatically expanded (in real-time) to 'diesel manual.' This indicates that a query for 'diesel' and a query for 'diesel manual' have the same results or, in other words, 'cars with diesel engines only come with a manual transmission' and hence there are no automatic diesel vehicles in this site.

options available in pull-down menu designs. For instance, the *Sony Advisor* reactive menu (see Fig. 4) available through sonystyle.com (mentioned earlier in the section on flexible navigation orders) updates a set of products, such as digital cameras and camcorders, as users add and remove options, such as picture resolution, from those products. The automatic addition and deletion of products from the current set (right side of either window in Fig. 4) based on the status of the the pull-down checkboxes for each facet (left side of either window in Fig. 4) exposes the dependencies underlying the products Sony offers and allows shoppers to naturally explore the interplay of features in the available models. Wine.com has a similar interface for exploring dependencies while browsing wines.

We can also expose these dependencies through query expansion. For example, when the user says 'safety rating $>$ 3.5' (when, e.g., the site solicits for car make), we can expand this query to 'safety rating $>$ 3.5, $<$20 mpg' because *all* cars with a safety rating greater than 3.5 also yield $<$20 mpg. Such automatic query expansion, when conducted in real-time, provides immediate feedback to the user.

Consider the following information-seeking scenario to help illustrate the use of real-time query expansion:

> *Scenario 3*:
> "I am interested in buying a car that runs on diesel fuel, but I've heard that diesel engines are usually only available for cars with a manual transmission. However, I want an automatic transmission. Are any diesel cars equipped with an automatic transmission?"

Such a scenario can be easily supported if we know whether the 'diesel $\rightarrow$ manual' dependency holds. Fig. 5 illustrates how a user might employ a real-time query expansion interface to an automobile hierarchy to approach this scenario.

Notice that real-time query expansion is *user-independent*; the expansion is the same for *all* users and only depends on the co-occurrence of terms on the paths through the hierarchy. Prior queries have no bearing on the expansion. Additional (expansion) terms are added to the query *only* when the composite query yields the *same* result as the unexpanded query. Interfaces using a similar style of real-time query expansion for information exploration and discovery are emerging on the web. See *Google Suggests* (http://www.google.com/webhp?complete=1&hl=en) or Stanford's auto-complete *Search on TAP* (http://sp06.stanford.edu) for examples.

It is important to note that the complete set of dependencies satisfied by a classification change as the user interacts with it. For instance, consider the user who clicks on 'Lexus' (car make) at the top-level of an automobile classification. This interaction eliminates the 'Lexus $\rightarrow \neg$ BMW' dependency and may cause the 'coupe $\rightarrow$ sunroof' dependency to emerge. This latter dependency indicates that all Lexus coupes have a sunroof and would likely *not* exist in the site *prior* to the selection of 'Lexus,' unless *all* coupes from *all* car makes have a sunroof (unlikely). By revealing the (all or nothing) dependencies underlying the (automobile) facets of a hierarchical classification, real-time expansion can be used to help a user understand the constraints underlying a domain.

In summary, there are multiple levels of dependency exploration. Drilling-down a hierarchy via out-of-turn interaction (typically) causes the available choices for all facets to be pruned. However, since this design only presents one facet per page, the user is only able to observe how her input affected the set of choices for the current facet. Manipulating reactive menus also causes the choices for all facets to be pruned. However, unlike an out-of-turn interface, here the user is able to observe the pruning in all facets due to the enumerative style of the interface. We can reveal more dependencies by adding real-time query expansion to an out-of-turn interface.

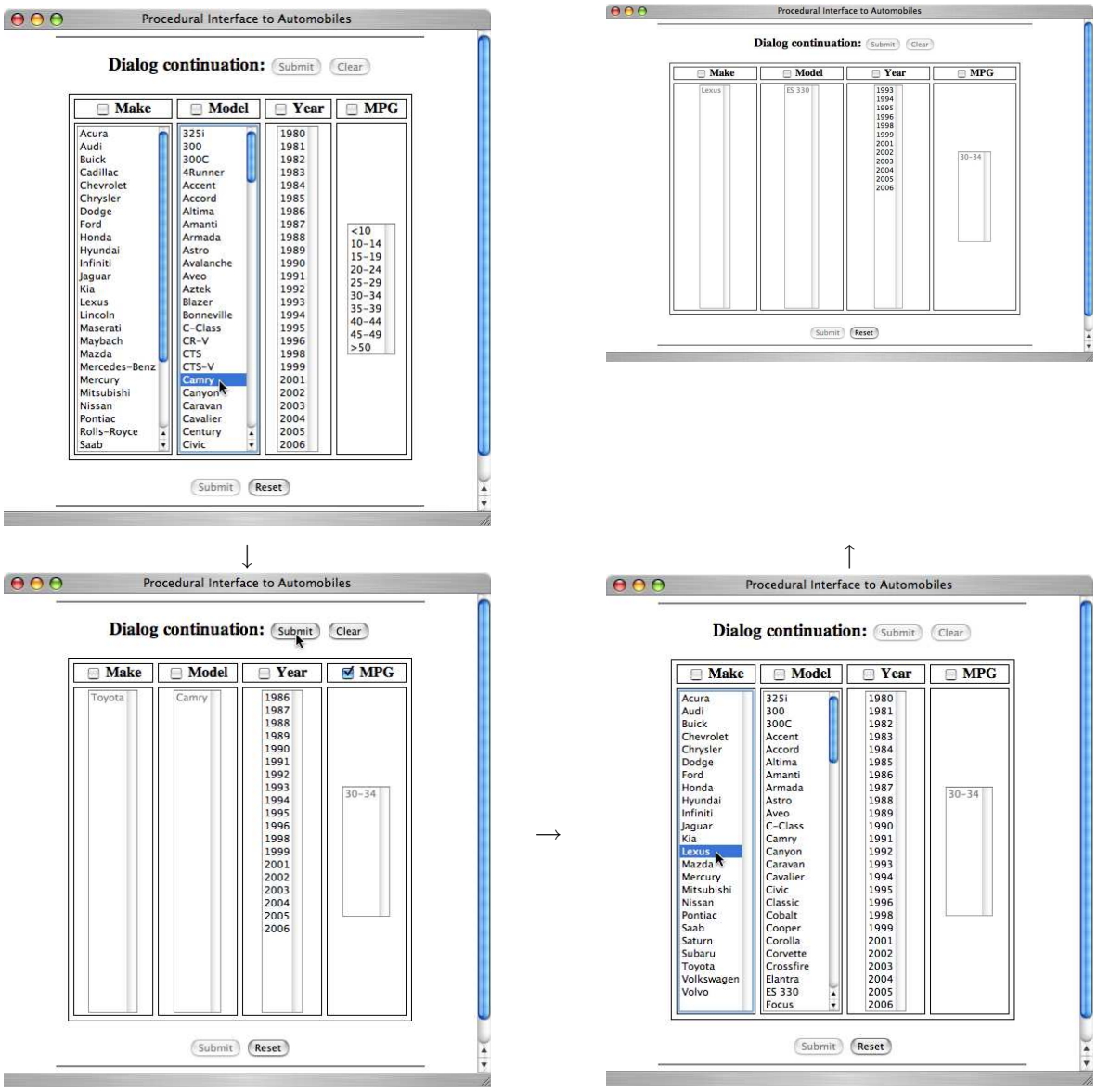



Figure 6: Table-based continuation interface. The two screens on the left illustrate one line of inquiry, yielding a result which is then cascaded into another line of inquiry (right two screens).

## Support for Procedural Tasks

The interactions we have described until now entail drilling down a classification and thereby reducing the size of the hierarchy and the number of items (e.g., cars) remaining. We say that such interactions are *destructive* and involve only one line of inquiry (or control flow). However, strategies for tackling some information-finding tasks are constructive or *procedural* [2] in nature, and require cascading information across multiple sub-goals. For instance, consider the following scenario:

> *Scenario 4*:
> "I'm interested in a Lexus. I want one whose fuel efficiency is comparable to the Toyota Camry."

This scenario involves two sub-goals. First, the user must find the fuel efficiency of the Toyota Camry and, then, use that information to find a particular Lexus. With a faceted or table-based interface, the user would need to manually remember the information retrieved in the first sub-goal, start over with the fully populated instance of the table, and supply the retrieved information to satisfy the second sub-goal. This process is only exacerbated as the number of sub-goals increase. Interfaces with the *support for procedural tasks* feature provide a facility for the user to naturally aggregate retrieved information into a new line of inquiry without having to remember any intermediate result(s) or start over.

### Continuation-based Interface Designs

One way for the user to cascade information from one sub-goal onto another is through what we call a *user-initiated continuation*, borrowing its name and motif from the concept of a *continuation* [11] in programming languages. A continuation indicates a 'promise to do something' and summarizes the amount of work remaining at a point of execution in a program. To cascade the output of one information-finding goal to the input of another, we essentially replace the current pruned hierarchy (i.e., the current continuation) with a fresh copy of the original hierarchy that has been pruned *a priori* with respect to the information retrieved in the previous sub-goal. This process provides a smooth transition between sub-goals for the user.

Fig. 6 illustrates how a user might approach scenario 4 using a table-based continuation interface design. The user first supplies the information she *does* have, namely (model), by clicking on 'Camry' in the list labeled 'Model' (Fig. 6, top left). This cause several choices to be pruned out of each of the lists and reveals an MPG of 30–34 (Fig. 6, bottom left). In order to supply this retrieved information in a new line of inquiry to a fully-populated instance of the table, the user checks the box labeled 'MPG' (now that only one item remains) and clicks the 'Dialog continuation' Submit button (Fig. 6, bottom left). This results in an instance of the table containing only the makes, models, and years of automobiles which get 30–34 MPG (Fig. 6, bottom right). Now the user clicks on 'Lexus' (Fig. 6, bottom right) to find the information (i.e., model of Lexus) in which she is ultimately interested. This reveals that the Lexus with fuel efficiency comparable to the Toyota Camry is the ES 330 (Fig. 6, top right). Notice that the continuation facility allows the user to both abandon a given line of conversation (since the requisite information has been obtained) and find herself in the middle of another line of inquiry. Procedural information-seeking and information aggregation is relevant in a variety of tasks, especially 'information *re*-finding,' the process of pursuing some information which you have already found [7] in the past and would like to find again.

Notice that the continuation feature is not an isolated option. It can be added to a variety of interface designs. However, the continuation feature is not relevant in a generative design. Recall that a generative interface is used to create a hierarchy tailored to the needs of a user and not to directly solve an information-seeking task, procedural or otherwise. We can, however, add the continuation feature to the hierarchy resulting from interaction with a generative interface.

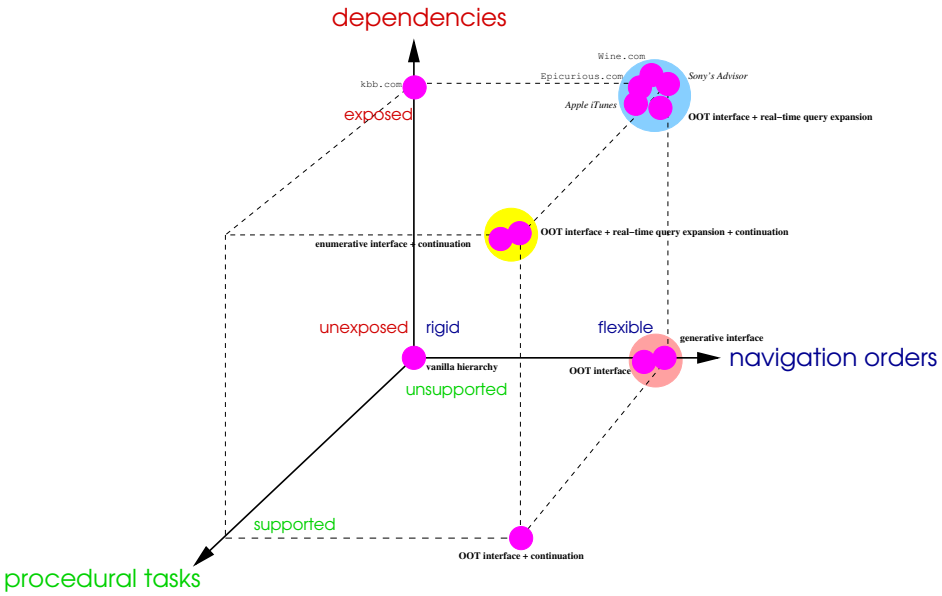



Figure 7: The three dimensional space of interfaces to hierarchical websites induced by the features discussed in this article.

Procedural tasks are common in solving complex constraint satisfaction problems. For example, consider a tourist planning a trip to Europe where the tourist must not only develop a carefully staged schedule of events (e.g., the train arrives in London at 3:00p, rental car will be available at 3:30p, and hotel check-in is at 4:00p), but also satisfy constraints in the process (e.g., *Le Louvre* is closed on Sundays). The ability to automatically cascade output from one information-seeking process into another as illustrated above is an important ingredient for supporting such activities.

## Discussion

The above features offer a sneak peek into the variety of offerings available to improvise on the vanilla hierarchy. We saw that we can achieve *flexible navigation orders* through i) an enumerative (faceted, e.g., epicurious.com; table-based, e.g., *iTunes*; or pull-down menus, e.g., sonystyle.com) interface, ii) an (*a priori* or on-the-fly) generative interface, or iii) an out-of-turn interface. Note that the use of enumeration and the capability to supply input out-of-turn are mutually exclusive in an interface. If all possible choices, for all possible unspecified facets, are enumerated on the current webpage, then there is no need to interact out-of-turn. Similarly, the use of enumeration and generation are mutually exclusive. We demonstrated that *exposing and exploring dependencies* comes for free through the pruning of choices for facets in enumerative and OOT designs. However, an enumerative design reveals more dependencies at once since it presents all choices for all (unspecified) facets on one page. Thus, we can augment an OOT interface with real-time query expansion to make more dependencies salient. We also illustrated that we can achieve *support for procedural tasks* by adding a continuation facility to any interface presented here, save for a generative design.

These three features induce a three-dimensional space of possible interfaces supporting (combinations of) them. Fig. 7 situates the interface designs presented here in this space. Notice that while there are 8 (=

$2^3$) combinations of these three features, corresponding to the 8 corners of the cube, two of them are not meaningful and, therefore, two corners of the cube in Fig. 7 contain no examples. Specifically, there are no points at (rigid navigation orders, unexposed dependencies, supported procedural tasks) or (rigid navigation orders, exposed dependencies, supported procedural tasks). The reader may also note that if you want to expose dependencies in an interface, you must also have flexible navigation orders, but the reverse is not so.

Choice of implementation technologies is another important dimension which must be considered in developing interfaces to information hierarchies. Here, we can distinguish between the amount of sophistication built into the interface versus the back-end (databases, dynamic webpage generation capabilities). If we view the user-site dialog as one of communicating inputs and receiving webpages in return, then we must assess how input is captured and represented, where pages are transformed, who is responsible for maintaining the state of the interaction.

If no out-of-turn input is to be supported, designs using hyperlinks or pull-down menus can be realized at the browser level (with static pages possibly augmented with runnable JavaScript code) without requiring costly lookup, mapping, or transformation operations. Out-of-turn input, on the other hand, requires some facility to map the user's unresponsive inputs to hyperlink labels (tags) and, more importantly, to transform the site to accommodate the out-of-turn input. The former is typically supported using XUL (XML User-interface Language; xulplanet.com) for a toolbar or using SALT (Speech Application Language Tags) and X+V (XHTML+Voice) for voice interaction. The latter is typically done via XSLT running on the server.

New web technologies such as AJAX (Asynchronous JavaScript and XML) [9] provide support for more effective communication between front- and back-end components. AJAX is particularly helpful for handling any *asynchronous* communication with the user required for rich and responsive web interaction. For this reason AJAX is ideal for implementing real-time query expansion and other within-page designs like those used in table-based interfaces. For instance, AJAX facilities are crucial to the draggable maps in *Google Maps* at maps.google.com. Moreover, we can use AJAX to develop the mapping module described above while minimizing browser–server communication. Also, toolkits such as the *PLT Scheme servlet API* (see www.plt-scheme.org), which is based on first-class closures and continuations [4], provide a sound approach for realizing *stateful* web interactions, especially those required for the continuation facility described here.

New designs will undoubtedly emerge as we have greater understanding of users' goals and objectives. For instance, in order to support 'what if' analysis we need to provide additional operations such as union and intersection over intermediate results (that might be stored in a scratch-pad akin to the online 'shopping cart'). Another practical interaction feature permits the user to preview a page prior to clicking on the link/button which retrieves it. Such a 'I just want to see what is behind the current page' capability is important in scenarios where the user is unsure, e.g., if clicking 'Submit' is going to charge their credit card! Such novel features will provide the next generation of interface options to support compelling functionalities in web hierarchies.

## References


[1] R. Agrawal, T. Imielinski, and A. N. Swami. Mining Association Rules between Sets of Items in Large Databases. In *Proceedings of the ACM International Conference on Management of Data (SIGMOD)*, pages 207–216, Washington, DC, May 1993. ACM Press.

[2] S. K. Bhavnani, C. K. Bichakjian, T. M. Johnson, R. J. Little, F. A. Peck, J. L. Schwartz, and V. J. Strecher. Strategy Hubs: Next-Generation Domain Portals with Search Procedures. In *Proceedings*

*of the ACM Conference on Human Factors in Computing Systems (CHI'03)*, pages 393–400, Fort Lauderdale, FL, April 2003. ACM Press.

[3] B. De Carolis. Generating Mixed-Initiative Hypertexts: a Reactive Approach. In *Proceedings of the Fourth International Conference on Intelligent User Interfaces (IUI)*, pages 71–78, Los Angeles, CA, 1999. ACM Press.

[4] D. P. Friedman, M. Wand, and C. T. Haynes. *Essentials of Programming Languages*. MIT Press, Second edition, 2001.

[5] M. A. Hearst. Next Generation Web Search: Setting Our Sites. *IEEE Data Engineering Bulletin*, Vol. 23(3):pp. 38–48, September 2000.

[6] M. A. Hearst, A. Elliott, J. English, R. Sinha, K. Swearingen, and K.-P. Yee. Finding the Flow in Web Site Search. *Communications of the ACM*, Vol. 45(9):pp. 42–49, September 2002.

[7] W. Jones, H. Bruce, and S. Dumais. Keeping Found Things Found on the Web. In *Proceedings of the Tenth ACM International Conference on Information and Knowledge Management (CIKM)*, pages 119–126, Atlanta, GA, November 2001. ACM Press.

[8] M. Narayan, C. Williams, S. Perugini, and N. Ramakrishnan. Staging Transformations for Multimodal Web Interaction Management. In *Proceedings of the Thirteenth ACM International World Wide Web Conference (WWW)*, pages 212–223, New York, NY, May 2004. ACM Press.

[9] L. D. Paulson. Building Rich Web Applications with Ajax. *IEEE Computer*, Vol. 38(10):pp. 14–17, 2005.

[10] S. Perugini and N. Ramakrishnan. Personalizing Web Sites with Mixed-Initiative Interaction. *IEEE IT Professional*, Vol. 5(2):pp. 9–15, March–April 2003.

[11] D. Quan, D. Huynh, D. R. Karger, and R. Miller. User Interface Continuations. In *Proceedings of the Sixteenth Annual ACM Symposium on User Interface Software and Technology (UIST)*, pages 145–148, Vancouver, Canada, November 2003. ACM Press.