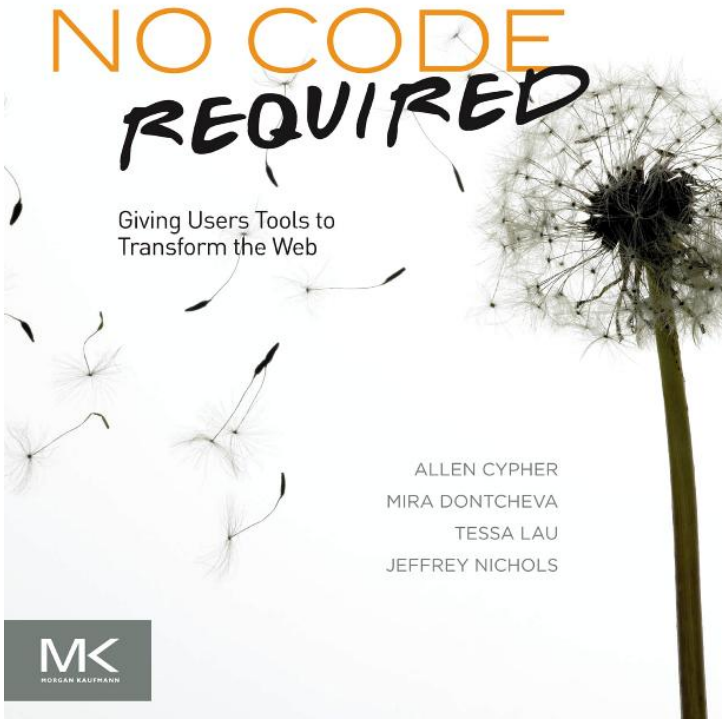

From: Aran Lunzer[1], Kasper Hornbæk[2], Subjunctive interfaces for the Web. In: Allen Cypher, Mira Dontcheva, Tessa Lau and Jeffrey Nichols, editors: *No Code Required: Giving Users Tools to Transform the Web*. Burlington: Morgan Kaufmann, 2010, pp. 267-285.
ISBN: 978-0-12-381541-5

CHAPTER

Subjunctive interfaces for the Web

14

Aran Lunzer,[1] **Kasper Hornbæk**[2]
[1]*Meme Media Laboratory, Hokkaido University*
[2]*Dept. of Computer Science, University of Copenhagen*

ABSTRACT

The data resources and applications accessible through today's Web offer tremendous opportunities for exploration: ask a slightly different question, receive a correspondingly different answer. However, typical browser-based mechanisms for accessing the Web only enable users to pose one such question at a time, placing a heavy operational and cognitive burden on any user who wants to explore and compare alternatives. A subjunctive-interface approach may reduce this burden. Subjunctive interfaces support the setting up, viewing, and adjustment of multiple scenarios in parallel, allowing side-by-side instead of temporally separated viewing, and more efficient iteration through alternatives. We have implemented a spreadsheet-inspired environment where end users can program and use their own Web-access applications that include such multiscenario support. In this chapter we describe three modes of use of this environment – parallel retrieval, coordinated manipulation, and tentative composition – and explain how these may help to alleviate typical challenges in Web-based tasks. At the same time, we acknowledge that the increased scope for exploration made possible through this environment can itself present a form of cognitive burden to users, and we outline our plans for evaluating the impact of this effect.

INTRODUCTION

How inconvenient it is that so many applications for accessing Web resources only deliver results in response to explicit, pinpoint requests. For example, interfaces for flight enquiries typically require the user to specify exactly one destination city, which is fine for users with precisely formulated travel plans but a bore for everyone else. A user who wants to compare the deals and schedules available for a range of destinations must embark on an exploration, submitting a succession of requests and analyzing their respective results.

One problem in such cases is that users have poor support for covering a range of requests, even if the details of those requests follow some regular pattern. A user searching for flights might request information for several routes on a given date, or for a single route over many dates, or combinations of a few routes and dates. But if the interface only supports the handling of a single request at a time, this burdens the user not only with a potentially high number of interface actions to specify and

submit the requests, but also with increased mental effort in planning the requests, remembering which requests have been made so far, and remembering where interesting results were found.

Furthermore, one-at-a-time interfaces provide poor support for comparing results (Terry & Mynatt, 2005), in that making comparisons requires the user to remember – or to have written down, or to request again – the details of those results that are currently out of sight. This again can constitute both a physical and a mental burden. We believe that these burdens can be reduced by enabling the user to carry out a number of requests at the same time. We refer to this as the use of parallel retrievals.

Consider now a second kind of Web interaction. A doctor who has access to her patients' records through a secure Web connection wants to retrieve images from specific stages in the treatment of a single patient, for example to observe progress of a disease within an organ. On obtaining each abdominal image study she goes through the same operations of selecting and scaling the desired sub-portion of the images, in three orthogonal planes, then adjusting the grayscale mapping so as to emphasize the boundary of the diseased region, and finally selecting display of just the overlays containing her own annotations. She accumulates browser windows, one for each imaging study, to be able to switch between them to help grasp the disease's changes over time. If she finds that the diseased region has spread beyond the bounds of the focus area she selected for the earlier studies, she readjusts those earlier views so that she can still compare like with like.

In this situation, which could equally apply to retrieving and manipulating weather maps, or archived pictures from a Webcam, it is frustrating to have to perform the same image manipulations many times over, especially given the risk that information gained from later views will upset earlier decisions. This frustration could be alleviated if there were a way to manipulate the various retrieved resources in concert, continuously confirming that the results will be valid for all of them. This we refer to as coordinated manipulation.

As a third example, consider a holidaymaker who (having successfully selected some flights) is now installed in a foreign city and is planning a day of sightseeing. The city is served by a navigation service Web site that provides estimates of point-to-point journey times on foot or by public transport. Having made a list of addresses he would like to visit, the visitor can use this site to plan an itinerary for the day.

The challenge here is to come up with a sequence of visits, and the journeys between them, that promises to be an interesting overall experience without excessive use of travel time or leg-power. If there were a strict constraint that all the listed sites be visited, a "traveling salesman" algorithm could be put to work in search of a time-efficient total solution – or, perhaps, the conclusion that there is no way to visit them all within a day. However, all but the most ardent tourist would probably take a more relaxed approach, trying a few alternative visit orders and transport options, and being willing to reject sites that turn out to be inconvenient to include. Nonetheless, this would be a frustrating task in the absence of support for what we call tentative composition – meaning, in this case, being able to compose and compare a number of alternative itineraries, involving different sites and/or different visit orders.

We believe that the above three kinds of challenge can all be addressed by offering users access to Web resources through subjunctive interfaces (Lunzer, 1999): interfaces that allow a user to explore alternatives by setting up multiple application scenarios at the same time, viewing those scenarios side by side, and manipulating them in parallel. In this chapter we report our investigations into using subjunctive interfaces for Web access.

The emphasis is on user control. As explained in the description of the third challenge, these cases are not amenable to definition in a way that would allow an automated algorithm to proceed, unsupervised, to an optimal solution. Only the user can specify what cases are of potential interest, and only the user can evaluate and (where necessary) compare the results. This interactivity places practical bounds on the complexity of setup that a user can be expected to handle; subjunctive interfaces are specifically designed to support about 10 parallel scenarios. Applications that would call for dozens or even hundreds of scenarios should be addressed with different techniques.

We begin by describing the RecipeSheet, a subjunctive-interface-enabled programming environment that we have equipped with mechanisms specifically for accessing Web resources. In the three following sections we then introduce three usage examples that address challenges of the kinds given previously; for each example we discuss briefly its applicability to common modes of use of today's Web. After these examples we address a potential downside to this work: that in seeking to make it easier to pursue an exploration that brings a range of information to a user's screen, we may be counterproductively increasing the burden on the user who then has to evaluate that information. Finally, we report on some initial feedback received from research colleagues for whom we implemented RecipeSheet-based applications for their work.

SUPPORTING MULTISCENARIO WEB ACCESS WITH THE RECIPESHEET

The RecipeSheet (Lunzer & Hornbæk, 2006a; Lunzer & Hornbæk, 2006b) is a spreadsheet-inspired environment that has built-in subjunctive-interface features, and thus supports parallel exploration of alternative calculations and their results. Like a spreadsheet, the RecipeSheet provides support for setting up custom flow-like calculations in terms of dependencies between cells. The subjunctive-interface features mean that the cells providing inputs at the start of a flow (referred to as ingredients) can hold multiple values simultaneously, the user can set up alternative scenarios based on chosen combinations of those values, and the cells holding derived values will then show the results for all scenarios, color-coded and/or spatially arranged to help the user understand which result arose from which scenario.

A RecipeSheet user defines inter-cell dependencies in terms of so-called recipes. There is a set of standard recipes, such as for extracting particular tagged elements from a chunk of XML, but users are also expected to create their own. Recipes can be programmed directly in Smalltalk, Open Object Rexx, or XQuery; recipes capturing behavior from Web applications can be built using the mechanisms of C3W (see Chapter 8) and Web-service recipes can be created with the help of SOAP or REST. In addition, the setup of cells and recipes on a sheet can be saved as a single composite recipe, that can then be used on other sheets.

Figures 14.1 and 14.2 show two of the fundamental operations in setting up a calculation flow on a RecipeSheet: adding a precoded recipe (in this case, written in Smalltalk) by dragging from a Recipe Browser onto the sheet, and then adding wires to connect a recipe to other recipes or cells.

Figure 14.3 illustrates one way for a user to set up additional calculation scenarios on a sheet: here the use of control-click (a mouse click with the Control key held down) on a value creates an additional scenario to hold the clicked value. In ingredient cells, markers with scenario-specific colored regions show which scenarios have been set up with which values. Further mouse operations

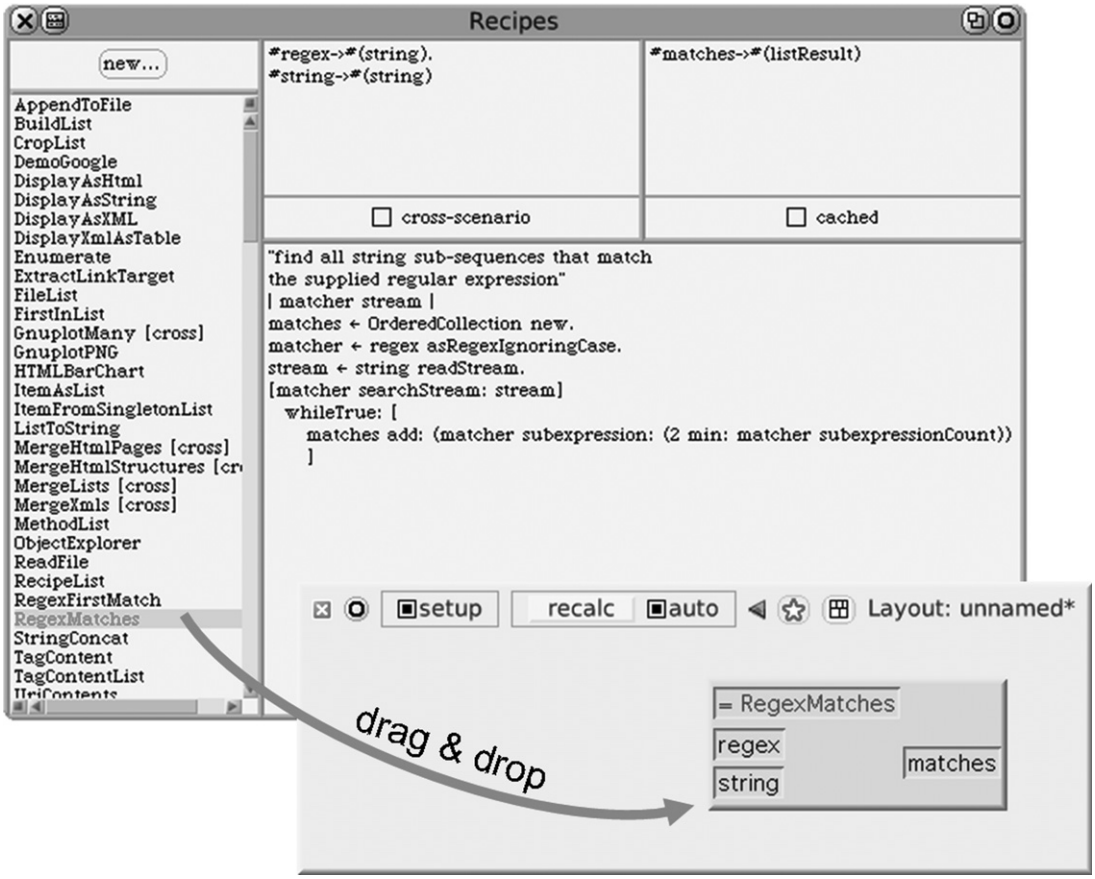

FIGURE 14.1

Adding a prebuilt recipe to a sheet, by dragging from the Recipe Browser. The sheet is in setup mode, in which recipes and their connections are displayed but the contents of any cells are hidden.

allow the user to move these markers to other values, thus adjusting the ingredients supplied to the scenarios' respective calculations. These operations are adapted from our earlier user studies on application-specific interfaces (Lunzer & Hornbæk, 2008).

As noted above, the RecipeSheet incorporates mechanisms originally developed for C3W (Clip, Connect, and Clone for the Web). Compared to the Microsoft Excel-backed substrate used in the original C3W implementation, the RecipeSheet provides more flexible facilities for connecting processing units extracted from Web applications, and for cloning their inputs to see multiple results side by side. The RecipeSheet replicates C3W's viewport facilities in the form of result cells that have the full rendering behavior of Microsoft's Internet Explorer (IE). These IE-based cells themselves offer C3W-style interaction features for selecting or clipping (i.e., extracting) HTML

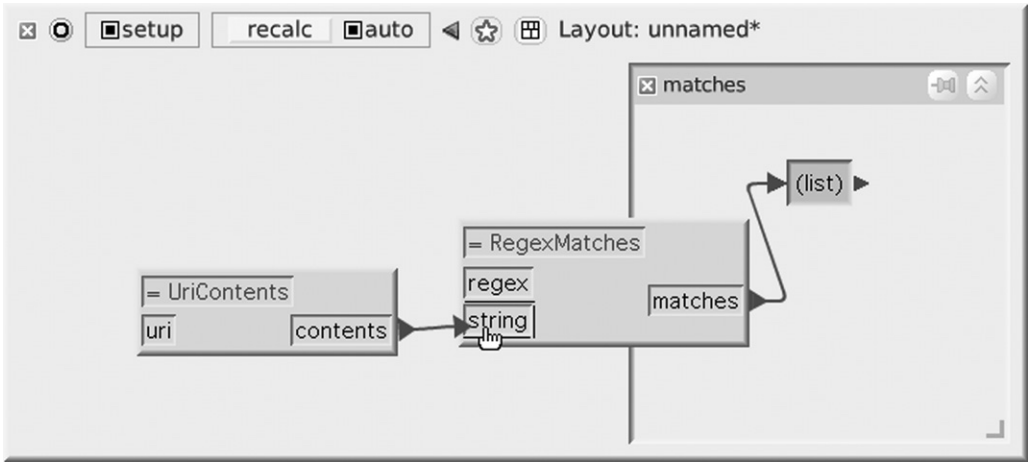

FIGURE 14.2

Dependency connections on a RecipeSheet. Here the user is specifying a connection from the *contents* result of the UriContents recipe to the *string* ingredient of the RegexMatches recipe, whose *matches* result has already been connected to a cell.

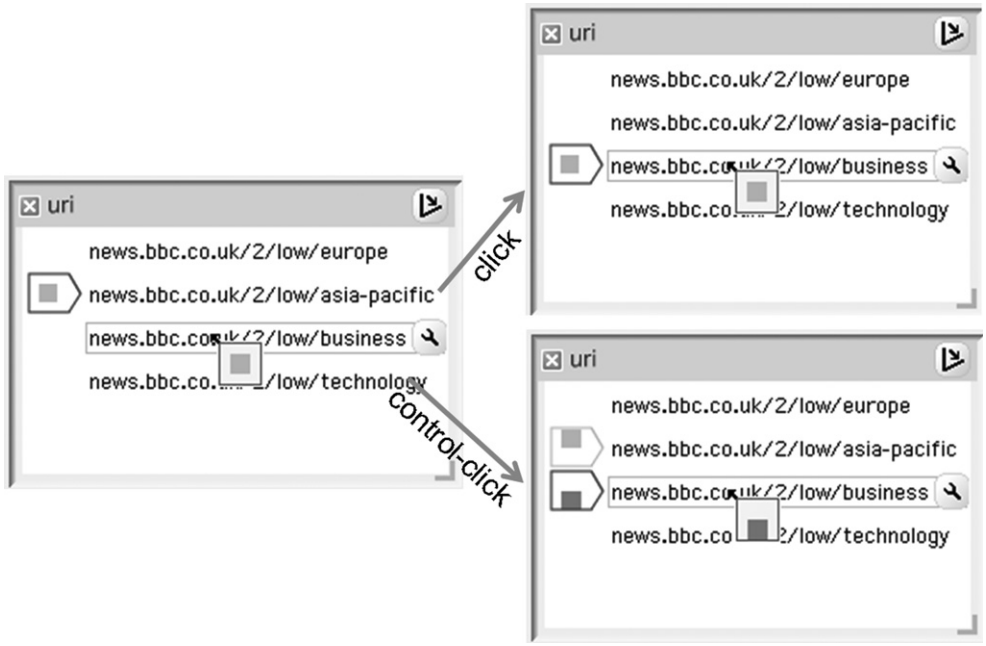

FIGURE 14.3

The basic mechanism for setting up additional scenarios. Whereas a plain click on an ingredient value just switches the input to the clicked value, a control-click sets up a second scenario to hold the value. Scenarios are distinguished by consistent use of color and relative positioning throughout the interface: within the scenario markers, and for side-by-side result displays (Figures 14.6, 14.7, and 14.9).

elements. For example, a user can select arbitrary elements within a page (e.g., in a page that includes a list of links the user might select a subset of the link anchors), and have the contents of those selected elements (in this case, the anchor tags) passed as inputs to another recipe for follow-on processing. Additionally, as shown in Figure 14.4, by interacting with such a cell a user can set up a degenerate form of C3W recipe that will simply extract a selected portion of the cell's contents for use in other cells or recipes.

A further property of the RecipeSheet is that the processing for a recipe is itself an ingredient – in other words, an input – that can be specified in a cell. The RecipeSheet can therefore provide uniform handling of variation in both inputs and processing, which seems a natural requirement in some forms of Web access. For example, whereas one user may want to view the results of sending alternative keyword queries to a single search engine, another might want to send the same query to

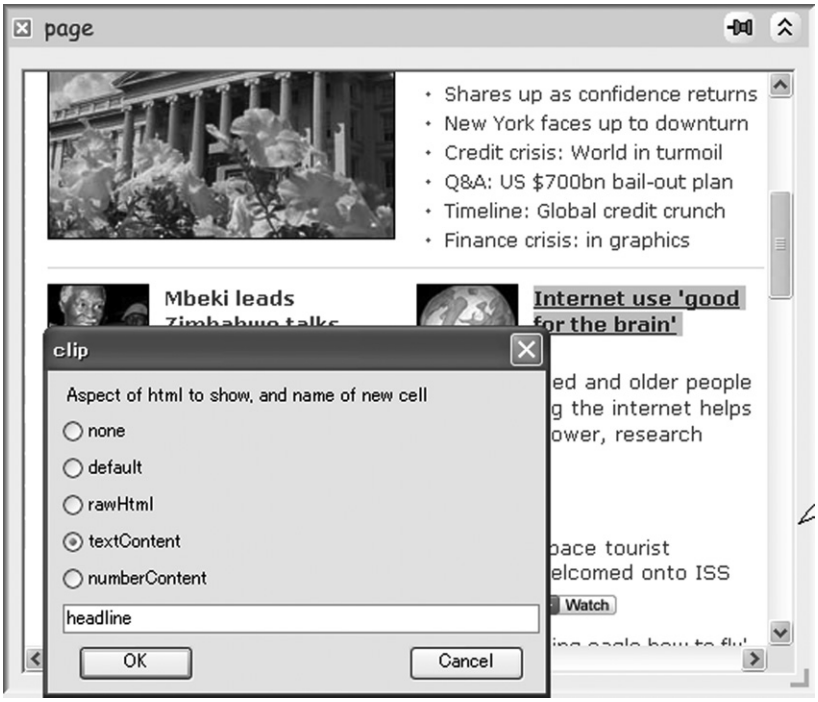

FIGURE 14.4

A visual mechanism for creating simple recipes to extract elements of HTML Web pages, by "clipping" a portion of a page displayed in a result cell. Here the user has highlighted and double-clicked a headline on a news page; in the dialog box she has given the name *headline* to the cell to be created to show this value, and has selected that just the textual content of the selected element (rather than the entire HTML tag) is to be extracted. This creates a custom recipe that will deliver to the *headline* cell the text contents of the corresponding element in whatever HTML is supplied to the cell *page*.

multiple engines. On a RecipeSheet the user can decide freely from one moment to the next whether to view variation in one aspect or the other, or a combination of the two.

This completes our brief outline of how the RecipeSheet supports multiple Web access scenarios. The potential benefits depend on how such scenarios are created and used. Each of the situations in the Introduction can be helped by a subjunctive interface being used in a different way: to support parallel retrieval, coordinated manipulation, and tentative composition, respectively. The following examples illustrate these three modes of use.

PARALLEL RETRIEVAL

Parallel retrieval refers to enabling the use of a retrieval style application, such as a flight enquiry site, to specify not just a single retrieval but several alternatives, differing in arbitrary ways, at the same time. These retrievals are handled in parallel as separate scenarios, and their are results displayed so that the user can see them all simultaneously, and can also see which retrievals delivered which results.

Figure 14.5 shows a sheet that has been set up to find related research article. When multiple searches are run in distinct scenarios, their respective results are merged into a single list for display in the `relatedPapers` cell. The items in this list are marked up, and then sorted, according to which scenarios they appear in. Thus, on the sheet that appears in the back of Figure 14.5 we can see that even though its three queries are all based on papers about the same Elastic Windows project, just three items were found by all three of the queries, and several items were found by one query only. A user who had searched on the basis of any single paper would have missed out on many results.

We suggest that presenting the merged and marked up results from multiple searches can, more generally, help users to work around the bias of any individual search. In (Lunzer, in press) we demonstrate how augmenting a Google search with a set of additional searches narrowed by date (e.g., by adding "1990..1999" to the search phrase) could bring to light items that, though coming at the very top of the results for their particular era, were drowned out of the top entries in a standard (non–date-narrowed) search. In particular, we believe that the markup showing why each item is being offered – for example, that it appeared high in a 1990s search but not in any other – will act as useful guidance for users in judging each item's importance. Muramatsu and Pratt (2001) made a call for this kind of "transparency" in search engine results, to help users of search engines to understand – and to control, if they wish – the transformations, such as stop word removal or suffix expansion, that are applied automatically to their queries. The desirability of such transparency in result presentation has itself gained much attention recently, as evidenced by an extensive survey (Cramer et al., 2008).

However, even having decided to augment a result display to reveal where each result has come from, it is far from clear what form of presentation will work best. Dumais, Cutrell, and Chen (2001), studying the impact of alternative formats for marking up results with automatically derived category information (e.g., distinguishing the various topics of pages retrieved by an ambiguous query such as "Jaguar"), found that users were much quicker at finding relevant items from lists divided according to category than from the complementary form of display in which category information was added

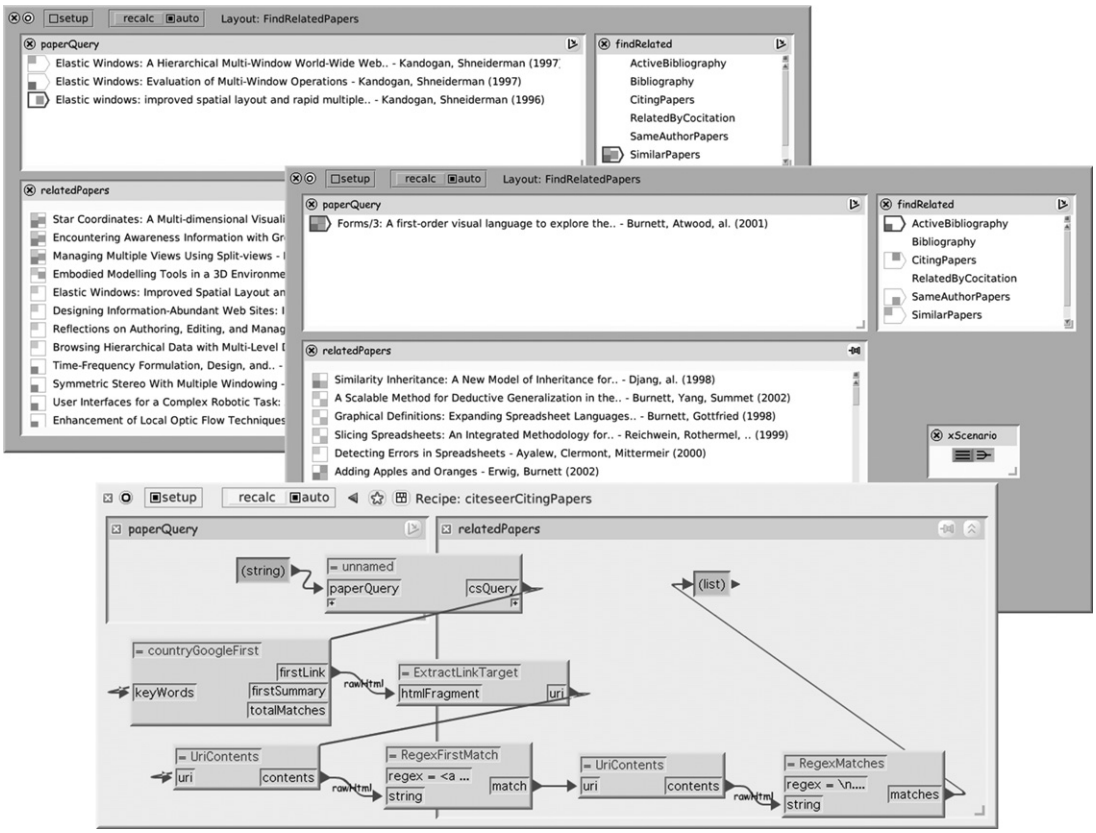

FIGURE 14.5

Parallel retrieval. Searching for academic articles, using mechanisms captured from the CiteSeer and DBLP Web sites. In the two upper sheets, the recipe specified in *findRelated* is used to find articles related to the one specified in *paperQuery*. In the sheet shown at top the user has requested a "similar papers" retrieval for each of three articles from the same project; in the second sheet, four alternative queries based on a single article. Results from all scenarios are merged into a single list, with markup to show which scenarios each item appears in – for example, the "Similarity Inheritance" paper (center of the figure) was found by three out of four searches. The searches were all coded as composite recipes; at the bottom is the setup of the sheet defining the CitingPapers recipe.

to each item in a single list. For an application such as the literature search shown in Figure 14.5, where items typically belong to multiple scenarios (cf. a unique category), and where this multiple membership itself has meaning, the trade-off is likely to be less clear-cut. In general we do not expect that any single presentation approach would be optimal for all parallel-retrieval situations; it depends too much on the nature of the information within each scenario, and the distinctions between scenarios. Our approach, therefore, is to give users the mechanisms they need to build multiscenario interfaces for their own Web searches.

In terms of Kellar, Watters, and Shepherd's (2007) four-category classification of Web-based information-seeking tasks, we regard parallel retrieval as being relevant to at least Fact Finding and certain kinds of Transaction. Fact Finding is used to refer to short-lived tasks for locating specific pieces of information, whereas Transactions covers interaction with Web applications, such as shopping sites, or email or blogging tools. The other two categories of information seeking – Information Gathering and Browsing – are by their nature less structured, and therefore less likely to have the regularity that makes parallel retrieval practical.

The fact that some Transaction-style operations have side effects, such as making purchases, would set a context-specific boundary on the actions that most users would want to perform in parallel. Whereas it would be reasonable to enquire about room prices at several hotels for the same date, for example, it would be highly unusual then to proceed to book them all. On the other hand, if the user's task happens to be to find a single hotel with a room available for each of several separate visits, proceeding to make a simultaneous booking for a set of enquiries (i.e., the various visits) might indeed make sense. Such an operation would fall within what we refer to as coordinated manipulation, as described in the next section.

COORDINATED MANIPULATION

By coordinated manipulation we mean having simultaneous control over several instances of an interactive application; in the introduction we gave the example of using this for browsing images. Within a subjunctive interface these application instances would typically reside within distinct scenarios created by the user.

Figure 14.6 shows a RecipeSheet built for the European Union's project ACGT,[1] which is pursuing (among other things) the development of an OncoSimulator that can reliably simulate cancer growth and treatment. The 3D visualization on the right of the sheet supports a limited form of direct-manipulation interaction: by clicking and dragging with the mouse, a user can rotate a view about horizontal and vertical axes. When there are multiple scenarios, and hence multiple views, their orientations are synchronized by the RecipeSheet such that rotating any one view causes the others to rotate the same amount, as seen in the figure. Such synchronized interaction is a staple of recent developments in coordinated and multiple views (Roberts, 2007), where it is recognized as a powerful technique for helping users to understand related data.

What is not readily apparent from the picture is that these views are in fact Web browsers, and the visualizations are AJAX-enabled pages. This provides the scope for implementing coordination at various levels, potentially applicable to a wide range of applications. The simplest form of coordination involves mirroring operations at the level of individual mouse and keyboard events. This allows coordinated control of visualizations that, like the 3D view in the figure, give uniform responses for user actions at equivalent coordinates within the view. If one were to open a set of Google Maps pages on different locations, for example, the operations of panning, zooming, and image selection could be mirrored at this level. Typing in a request for navigating from the map location to some other (common) location should also work, showing the different routes in

[1]EU FP6 Integrated Project "Advancing Clinico-Genomic Trials on Cancer" (2006-2010); http://www.eu-acgt.org.

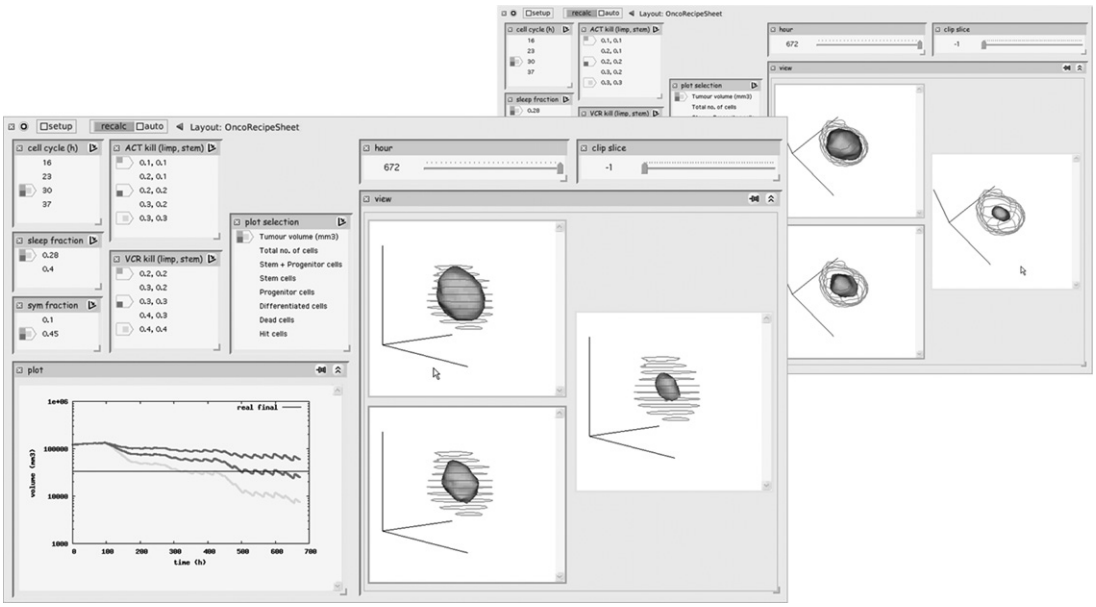

FIGURE 14.6

Coordinated manipulation. Two views of a sheet for exploring results from the ACGT OncoSimulator, a model for predicting the response of an individual patient's tumor to various forms of therapy. The five input cells at top left set the values for various simulation parameters. Here the user has set up three scenarios representing three levels of responsiveness to chemotherapy. In the large cell on the right, which shows an interactive 3D visualization of the simulated tumor, user manipulation is mirrored across all scenarios; in the background we see the outcome of rotating about a horizontal axis.

the individual views. Where this simple approach would break down is if the user switches into a mode such as a city's Street View, where the interaction options available depend on one's precise location in the city.

A next level of coordination would be through identifying and mirroring logical events: abstracting combinations of mouse movements and clicks to make up events such as selecting a menu item, or highlighting the entity at some location within an HTML page's DOM tree. Going a level higher still, one could employ mechanisms such as those of Koala/CoScripter (Little et al., 2007) to record and share operations in a way that would be robust even in the face of (some) differences in page layout.

Mirroring events at an abstract level therefore makes it possible to support not just manipulation of the objects within Web pages, but coordinated clicking of link anchors to navigate from one page to the next through matching regions of a Web site – for example, through standardized sets of pages relating to hotels on a travel site, or proteins in a bioinformatics repository. Hence, as mentioned previously, the possibility of querying a travel service to find a hotel that has a room available for each of several visits, then going through the booking procedure for all those visits together.

The above discussion shows how a given task can straddle the border between parallel retrieval and coordinated manipulation. Work by Teevan et al. (2004) suggests that much directed search on the Web – that is, search for a target that is known in advance to exist – is carried out as a mixture of the basic elements that underlie the two. Teevan et al. distinguish between, on the one hand, teleporting, by which they mean jumping directly to a Web page found as the result of a query, and on the other hand, orienteering, their term for localized, situated navigation that begins at a familiar starting point (such as a portal) and narrows in on the target. As noted before, the best we can do as interface designers is to provide facilities for users to choose for themselves the mix of teleporting and orienteering, and the range of scenarios over which they wish to perform the two. For now we are investigating what facilities make sense for the user group developing and calibrating the ACGT OncoSimulator.

TENTATIVE COMPOSITION

Some Web-based tasks can be characterized as the composition of multiple pieces of retrieved information, where it is the overall composed entity that serves the user's purpose, rather than the elements on their own. The example given in the introduction, of building a sightseeing itinerary, is essentially a composition of the route recommendations returned by the navigation service in response to various point-to-point queries. Being able to experiment with and compare alternative compositions, such as in this case varying the sequence of locations to be visited, is what we refer to as tentative composition.

As with the preceding two modes of use, tentative composition covers a broad range of users' tasks. At the simple end of this range are tasks in which the "elements" being combined are merely the values for placeholders within a structure that has been defined in advance: an everyday example would be the choice of starter, main course, and dessert to make up a meal; using today's Web one might create an office party invitation by composing venue details, transport information, and a map. Cases such as these can be treated simply as parameterized retrievals, and therefore explored using parallel retrieval mechanisms.

At the complex end of tentative composition tasks are cases of general design with arbitrarily many degrees of freedom, such as the planning of a new building or of a multi-continent concert tour. For some of these complex domains there are already specialized applications that include support for exploring design alternatives, and we are not suggesting that generic mechanisms for handling multiple scenarios could provide a similar strength of support. We believe that subjunctive interfaces will make their mark on small-scale, ad hoc composition of Web resources.

Supporting tentative composition requires, first, providing a substrate on which compositions are built. Then there must be a convenient way for the user to specify alternatives, and supportive mechanisms for viewing the corresponding outcomes and understanding how they differ – either in terms of the final results, or the alternative specifications that led to them. The RecipeSheet, having been designed to work as a substrate for flow-style calculations based on values supplied in cells, is inherently suited to the simplest kinds of tentative composition which, as stated above, can be set up like parallel retrievals. Figure 14.7 shows one such example, where the composition being carried out is the application of style settings to a Web page.

Beyond these simple cases, the RecipeSheet's supportiveness depends on how the composition is defined as a calculation flow. The building of a sightseeing itinerary could be tackled in various ways:

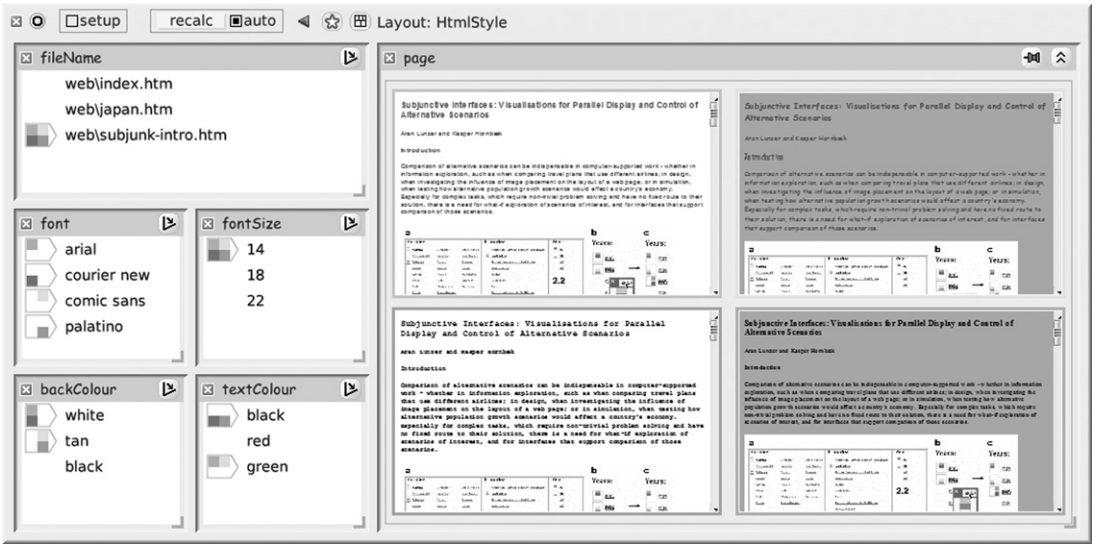

FIGURE 14.7

Tentative composition. In this case what is being composed is a rendered Web page, based on values supplied for the page content and for various style-defining parameters. The user has set up four alternative "compositions," and can see at a glance differences between them, such as how the font style affects the amount of space needed to render a given paragraph.

one possibility is to have a cell defining each sequential step in the itinerary (for example, one cell specifying the first visit address, a second cell specifying the second visit, and so on); another is to have a single cell in which the whole itinerary is specified as a list of addresses, and that lets the user specify different lists for different scenarios. The fact that the RecipeSheet makes specifying alternative processing as easy as specifying alternative parameter values would be useful in experimenting with alternative navigation services within these itineraries. However, we readily admit that both of the above approaches have potentially troublesome limitations: for example, the first would be highly inefficient for a user who wanted to try adding or removing one visit from an existing itinerary, whereas the second would provide poor support for grasping rapidly how two or more itineraries differ.

Although we are sure that the current design of the RecipeSheet is not the final answer in terms of supporting tentative composition in general, we believe its current level of support is sufficient to begin evaluation on exploratory tasks of this kind.

RISKS OF COGNITIVE OVERLOAD: THE PARADOX OF CHOICE

The Paradox of Choice is the title of a popular book by Barry Schwartz (2004), in which he points out that although having some freedom to make choices in your life feels much better than having no choice at all, too much choice is a problem in its own right. People get stressed by the amount of

mental effort involved in weighing up alternatives, by the worry that other, better alternatives are somewhere out there to be found, and, after making a choice, by the fear that on balance one of the rejected options might have been better.

Given that subjunctive interfaces are intended to improve the quality of information users receive by encouraging them to request and view more alternatives, Schwartz's studies suggest that we might be doing our users more harm than good. Especially given the vast amount of information available over the Web, it can be argued that what users desperately need is more filtering, not more retrievals.

However, we feel that the current popular approach to helping users make sense of the Web – namely, using some hidden ranking or other heuristics to deliver a small, possibly high-quality but necessarily biased selection of results – is asking users to put too much trust in online systems. There is some evidence that users are alert to this: for example, Lin et al. (2003) found, in a study of users' attitudes to question-answering systems, a tendency to feel uncomfortable accepting answers from systems that provided only the bare answer. The users wanted to see some context surrounding the answer, to help them confirm its legitimacy.

Nonetheless, there are also plenty of studies showing that giving users too much to do is counter-productive. Beaulieu and Jones (1998) discuss the relationships between the visibility of system functions, the balance of control between user and system, and the user's cognitive load. They experimented with a relevance-feedback retrieval interface that was purposely designed to keep the users in control of their queries, by revealing the details of the feedback-derived query terms and requiring the users to review and adjust those terms. The users' response to this interface was to become less active, rather than more, presumably because they felt that making all those adjustments would be too much work. Muramatsu and Pratt (2001), who concluded from their study of transparent queries (mentioned earlier) that perhaps the best style of interface would be a "penetrable" interface – one that lets the user know what has been done, and also provides a chance to change it – made a point of adding the caveat that providing too much control could inadvertently overload the user.

Part of the issue, as Beaulieu and Jones note, is that users need to feel that the decisions available to them are relevant to their personal information needs, rather than being just artifacts of the interface. For our own goals of deploying end user programming and customization techniques that help users to express a range of directions to investigate, and then to make sense of the corresponding range of results, we must therefore strive to ensure that users will perceive this effort as part of what they wanted to do anyway. If we can achieve that, there is hope that users will regard the ability to set up and work with multiple scenarios as a welcome level of choice, rather than an unwelcome source of stress.

RESULTS OF INITIAL EVALUATIONS

Our initial evaluations have been based on two applications that we built using the RecipeSheet to meet specific needs of some of our research colleagues. Rather than pursue laboratory evaluations of the RecipeSheet, we decided to test it on real and complex examples of custom built Web processing. We set out from Shneiderman and Plaisant's (2006) notion of multidimensional, in-depth, long-term case studies, in which the idea is to iteratively develop and refine information visualizations in collaboration with professionals. Thus, we have worked in depth with two research projects

and the applications they develop. Below we discuss these cases and present some initial results, relating in both cases to the pursuit of parallel retrievals.

The first application is an interface for running queries against DBpedia, an RDF representation of 2.6 million data items and their relationships as extracted from Wikipedia. Figure 14.8 shows an example of use of a sheet that has been set up for submitting SPARQL queries against the DBpedia data set and viewing their results. This example shows an exploration of the question, "With what country or countries is each Nobel Prize winner associated?" The data held in Wikipedia, and hence DBpedia, allows many ways of identifying such associations: Figure 14.8 shows the simultaneous posing of queries based on (a) the country of an educational institution attended by the person; (b) the country of an institution where the person worked; (c) the country where the person was born; and (d) the country where he or she died. Being able to pose these queries independently but simultaneously, and to see, from the scenario markup included in the results, which answer arose from

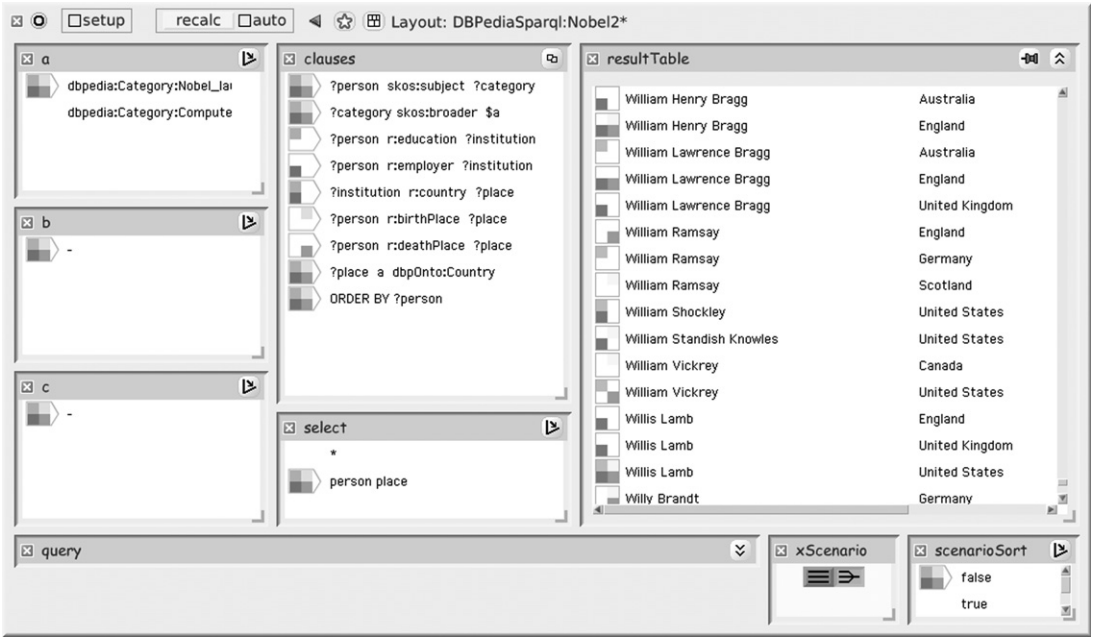

FIGURE 14.8

Requesting and viewing multiple parallel DBpedia queries. On this sheet, queries are defined as compositions of `clauses` entered by the user into the clauses cell. The clauses can include tokens such as $a, $b that refer to values in the corresponding named cells on the sheet (in this example only $a is being used). The user is looking for Nobel Prize winners and the countries with which they are associated, and has set up four scenarios querying, respectively, countries related to the person's education (scenario 1, shown with a mark in the top-left quadrant), employment (bottom left), birth (top right), and death (bottom right). The results show, for example, that – at least according to DBpedia's data – William Henry Bragg was born, worked, and died in England, and also worked in Australia.

which scenario, offers insights that would not be available from a simple list of people against countries. Among the available insights are an appreciation of DBpedia's incompleteness (for example, that country of birth is sometimes not clearly identified), and some common inconsistencies (such as variation between England and United Kingdom).

In evaluating this application we worked with two colleagues who are carrying out research on interfaces for accessing Semantic Web resources. They had both chosen DBpedia as a target platform for their work, and were at the stage of exploring its structure and content to understand what degree of regularity they would be able to exploit. Both were already well versed in the SPARQL query language. We made our DBpedia-specific sheet available for them to install, and supervised their first sessions of use to provide guidance on the basic facilities available. Their explorations of DBpedia continued for around a month, during which they occasionally consulted us with questions or requests for additional features. Our follow-up interviews were semistructured and focused on pertinent work tasks and the match of the RecipeSheet to those tasks. We also specifically canvassed their views on any difficulties they had encountered in setting up and manipulating scenarios.

Both of the interviewees felt that the RecipeSheet interface provided better support for their explorations than, for example, the various Web-browser-based SPARQL endpoints available at the time. One aspect of this perceived improvement arose from some simplifying assumptions we had made in our implementation: supporting a useful but incomplete subset of SPARQL (notably, only SELECT queries), and providing an extensive set of simple pre-canned entity-prefix abbreviations (such as **p:** for all DBpedia predicates). Further benefits arose from basic housekeeping characteristics of RecipeSheet layouts: for example, entity descriptors found within query results could be transferred easily (by drag and drop) to an ingredient cell, where they would remain through saves and reloads of the sheet, serving a reminding purpose even if not used often in further queries. This was especially useful for hard-to-remember entities, such as YAGO subject descriptors.

Both colleagues appreciated and benefited from the ability to run multiple queries side by side. One showed how she had used this in learning about the consistency of use of alternative predicates that indicate the same real-world relationship (e.g., **p:birthDate**, **p:dateOfBirth**); the other gave as an example his exploration of how consistently particular predicates were used across a range of related entity classes (e.g., various subclasses of **yago:Movie**). More generally, they saw the ability to see merged results from many queries as offering an augmented form of UNION: not only do you see the combined list of results as would appear in a standard UNION query, but each result is marked up with the particular query (which would normally be buried as a subpart of the UNION) that gave rise to it.

Various pieces of feedback from these colleagues led us to iterate the design of some basic RecipeSheet features. We learned that our mouse-click-based interface for switching items into and out of particular scenarios in multiselection lists (such as the *clauses* cell) was sometimes confusing. We developed an alternative selection mechanism based on use of the keyboard's number keys, correlated with a simple top-left to bottom-right ordering of scenarios. To select the clause **?institution r:country ?place** in the top-left and bottom-left scenarios as seen in Fig. 14.8, for example, the user presses the 1 and 2 keys (separately) while the mouse is over the marker for that clause. The users found this easier to understand than the mouse-based interface, and also faster.

We also noticed that the users frequently made slips when performing certain operations, such as deleting ingredient values when they had intended to reduce the number of scenarios, or vice versa. In discussions we realized that the relationship between scenarios and ingredient values was a source

of confusion: in particular, the fact that a scenario can exist even if there are no ingredient values selected for it, and hence no colored marks on view corresponding to that scenario. We are working with the users to come up with a less surprising set of interface rules.

In response to specific requests we added a facility for stepping back through the recent history of scenario operations and selections, and a flashing telltale during long calculations (in this case, queries requiring lengthy processing by the DBpedia server) to alert the user that calculation is still in progress.

In our continuing evaluations in this area we are addressing other Web-centered retrieval domains (e.g., recommendations provided through the Amazon.com API), and shall be investigating users' understanding of and preferences regarding the merged presentation of alternative retrievals carried out in parallel. In this we hope to build on the context-presentation findings of Dumais et al. (2001) and follow-on work.

The second case study, which is still in progress, centers on our delivery of the front-end interface for the ACGT OncoSimulator, which has undergone many design iterations since the prototype seen in Figure 14.6. This includes recent changes made in response to feedback from the DBpedia study.

The first rounds of design iteration were carried out on the basis of our own experiences in using the sheet to exercise the simulation code across a range of cases, and in ironing out glitches in communication with the remote Grid services (provided by a project partner in Poland) that invoke the simulator, and with the Web server (provided by another partner in the Netherlands) that delivers the visualizations. The next stage in the project is an extensive validation and calibration of the OncoSimulator code, involving yet more partner groups. This is still under way at the time of writing.

Part of the job of simulator validation involves running large numbers of cases across sweeps of parameter values to confirm that the outcomes evolve consistently. Figure 14.9 shows one such parameter sweep, based on just 2 of the 35 parameters supported by the simulator. By setting up and recording many of these sweeps, the simulator developers will build up a picture of the zones of the parameter space within which the simulator functions predictably, and the confidence intervals that should be applied to its results based on sensitivity to critical parameters and their combinations.

For our evaluation we first wanted to confirm that our colleagues would be comfortable working with the RecipeSheet to set up, view, and where necessary keep records of such sweeps across critical regions of the parameter space. We gave the system to a member of the partner group in the Netherlands, and interviewed him after 3 weeks of using it. Again, the interview was semistructured; we focused on scenario setup and manipulation, with the overall goal of carrying out tasks directly relevant to the real-world validation work that the sheet is being used to support.

We found that the interviewee had become proficient in using the interface for setting up and viewing scenario combinations, and that he was generally satisfied with its features. We had made available to him the new keyboard-based scenario control mechanism that was suggested through our work with the DBpedia users, and he confirmed that for tasks involving complex setup of scenarios he too would opt to use this mechanism over the others available. However, he also gave comments similar to the DBpedia users regarding sources of confusion in the facilities for scenario deletion.

One set of features demanded by and newly introduced for the OncoSimulator application, though potentially of value in other applications involving access to a large parameter space, relates to helping users to understand where in the parameter space there are results that can be retrieved.

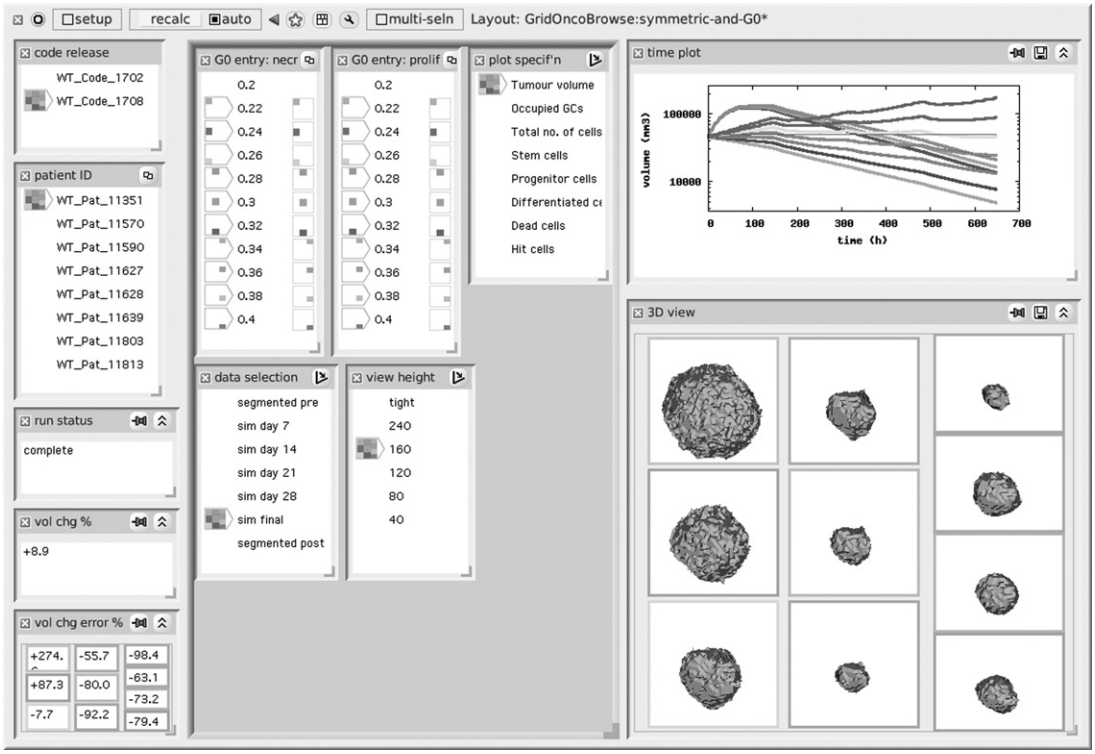

FIGURE 14.9

A parameter sweep carried out in a recent version of the OncoRecipeSheet. Of the 35 parameters used in driving the simulation, two related to *G0 entry* have been chosen, and set up with 10 values paired between the two. The results show, in the plot of tumor volume against time, an apparently dramatic discontinuity in tumor response for values beyond 0.34. Although these values (in combination with the default values used for the undisplayed parameters) are known to represent a biologically impossible situation, it is valuable to observe how the simulator behaves when driven to these extremes.

Given the OncoSimulator's total of 35 parameters, each typically having at least 10 candidate values, the population of the result space is inevitably sparse. We address this by a scheme of adding marks to ingredient values to show where results are available. What we have discovered from our initial evaluations is that users (including the previous interviewee) readily understand the principles of the markup scheme, but nonetheless have some difficulties in working with it in practice. One source of difficulties is the fact that this application has several parameters that work in pairs, and at the current stage of simulator development these paired parameters are typically given the same value, as is seen in Figure 14.9. Such cases seem to cause confusion because of the way that setting a value on one parameter immediately narrows the markup on its paired parameter to a single value. We are currently considering how to evolve this feature to reduce the scope for such confusion.

CONCLUSIONS

In this chapter we have outlined the role that subjunctive-interface mechanisms can play in supporting users' access to and comparison of Web resources. In particular we have shown how Web-access interfaces constructed using the RecipeSheet, a spreadsheet-inspired environment that has subjunctive-interface mechanisms built in, give end users the power to produce, on the fly, their own customized views of multiple resources. Users who also become proficient with the RecipeSheet's construction facilities, starting with its wiring-based interface for creating calculation flows, can go further to build their own interfaces for accessing whichever Web resources happen to interest them.

We are running studies to obtain evidence about the usability and effectiveness of the techniques described here, starting at the end user level. Our first goal is to establish firmly that users understand and can obtain benefits from facilities for using multiple scenarios in their Web interactions. We then plan to investigate how the physical and cognitive effort involved in reaping these benefits, including potential Paradox of Choice effects, influences users' willingness to pursue multiple alternatives rather than taking the easy option of accepting the first results on offer. We hope they can be persuaded that, at least some of the time, putting in a little extra effort does make sense.

Acknowledgments

We gratefully acknowledge the efforts of the developers of the systems on which the RecipeSheet depends, especially the members of the Squeak Smalltalk community, and of the colleagues who have assisted us in building and evaluating the RecipeSheet up to this point: the staff and students of the Meme Media Laboratory, Hokkaido University, and our many collaborating partners in the ACGT project, in particular at the University of Amsterdam and the National Technical University of Athens.

RECIPESHEET

Intended users:	All users; programmers
Domain:	Retrieval, design, and simulation
Description:	The RecipeSheet is not a dedicated Web programming system, but a spreadsheet-inspired environment in which users can build calculation flows and run multiscenario explorations to see how different inputs to the flows produce different results. The RecipeSheet has various mechanisms for accessing and displaying Web data.
Example:	The top results from a Google search can sometimes be affected by adding search operators such as "inurl:" or "allintitle:". A RecipeSheet could be set up to present the top 10 results from a Google search, and a user could choose to run several alternative searches in parallel. The sheet would merge the results, helping the user to check that she was not missing interesting results simply because of the presence or absence of an operator.
Automation:	Yes, tasks with unidirectional flows of data can be automated.
Mashups:	Yes, a sheet's flow can include the supply of results from one processing step as inputs to another, and each step can be delivered by a different Web site or service.
Scripting:	No, though programmers may create processing components.
Natural language:	No.
Recordability:	Yes, user actions can be recorded and result elements from form-style Web sites may be extracted, similar to C3W.
Inferencing:	No.
Sharing:	No, but recipes and layouts may be saved as files that can be distributed to other users.
Comparison to other systems:	The application of the RecipeSheet to Web-based resources is an extension of the original vision for C3W. Unlike other systems, building a sheet may require skills beyond those of a typical end user.
Platform:	Implemented as a plugin for Internet Explorer.
Availability:	A release is planned in 2010 to the Squeak community.