RecipeSheet: Creating, Combining and Controlling Information Processors

Aran Lunzer
Meme Media Laboratory
Hokkaido University
North 13 West 8, Sapporo 060-8628, Japan
Tel: +81 (11) 706 7262
aran@bigfoot.com

Kasper Hornbæk
Department of Computer Science
University of Copenhagen
Universitetsparken 1, DK-2100, Denmark
Tel : +45 35321425
kash@diku.dk

ABSTRACT
Many tasks require users to extract information from diverse sources, to edit or process this information locally, and to explore how the end results are affected by changes in the information or in its processing. We present the RecipeSheet, a general-purpose tool for assisting users in such tasks. The RecipeSheet lets users create information processors, called recipes, which may take input in a variety of forms such as text, Web pages, or XML, and produce results in a similar variety of forms. The processing carried out by a recipe may be specified using a macro or query language, of which we currently support Rexx, Smalltalk and XQuery, or by capturing the behaviour of a Web application or Web service. In the RecipeSheet's spreadsheet-inspired user interface, information appears in cells, with inter-cell dependencies defined by recipes rather than formulas. Users can also intervene manually to control which information flows through the dependency connections. Through a series of examples we illustrate how tasks that would be challenging in existing environments are supported by the RecipeSheet.

ACM Classification: H.5.2 [Information Interfaces and Presentation]: User Interfaces – Input devices and strategies, Interaction styles.

General terms: Design, Human Factors

Keywords: Subjunctive interfaces, personal information management, end user programming, information visualization, scientific workflow systems

INTRODUCTION
This paper presents the RecipeSheet, a programming environment that supports end users in creating custom setups for gathering, processing and visualising information from diverse sources.

We first illustrate some of the challenges involved in such information processing, by considering the tasks of three users. Althea is the event organiser for a large information-technology consultancy. One of her responsibilities is to arrange the annual company meeting, including inviting two computing researchers to give keynote talks. When considering candidate speakers she likes to get an overview of each person's interests and publication record by gathering information from the DBLP publication database, the ACM's Digital Library, and Google. Doing this by hand for many candidates involves repeatedly submitting queries at each of the Web sites, and even where the results are presented in standardised format it takes time for her to extract the details that are of interest.

Baya is a biology researcher. Part of his work involves analysing the evolution of genetic sequences by comparing sequences that he believes are likely to share ancestors. A typical task is to use the tool BLAST to carry out a broad search for sequences similar to some chosen protein, select a small subset of those search results for detailed analysis using ClustalW, then examine the outcome in a phylogenetic-tree viewer. Although the first two tools are both available through Web interfaces, his job cannot be done by a simple transfer of results from one tool to the inputs of another. In particular, in hunting for relationships between sequences from different species, he has to perform multiple sequence searches then laboriously use copying and pasting to gather sequence identifiers for submission to the next processing stage.

Our final example is Claude, a public-relations manager for a large retailer, whose job includes making short-notice inspections of regional stores. Budget constraints require him to obtain value for money on the hotels used during these visits. For each city he knows a handful of hotels whose Web sites offer good discounts when the hotel is not busy, but because his dates are also flexible – he just needs to have a suitable gap in his calendar – his search often involves repetitive trial-and-error enquiries at multiple hotels for multiple dates.

These tasks exemplify three information-processing challenges that we are interested in addressing. First, some users access diverse resources to assemble information relating to a given query, and then want to obtain the equivalent information for other queries without repeating the whole assembly process. Second, many information-processing tasks cannot be fully automated from start to finish, since only the user can decide which intermediate

UIST'06, October 15–18, 2006, Montreux, Switzerland

results merit further processing. Third, existing tools do not provide good support for trying alternative scenarios, and in particular for comparing results from alternatives.

There have been various attempts to remedy these problems in isolation. Researchers in personal information management, for example, have built systems that help users manage and access diverse information (e.g., [9]), and systems that provide support of various kinds for repeated capture of information from the Web (e.g., [6,16]). However, little attention has been paid to how users exchange data between applications once the data are in a unified format. The literature on scientific workflow systems (e.g., [10,15]) describes how users may set up specific exchanges of data between systems, but rarely addresses interactive exploration of alternative input or processing settings. We believe that there is a need for a tool that addresses in a coordinated way all of the problems that we have discussed.

We introduce the RecipeSheet, a first step towards such a tool, describing its architecture and user interface and showing how it can address tasks of the kind outlined above. We argue that the RecipeSheet extends existing research on end-user customisation of data access and processing.

THE RECIPESHEET

The RecipeSheet is a programming environment inspired by spreadsheet concepts. Like a spreadsheet it supports cells that contain data values, with some cells' values set by direct user input while others are calculated using inter-cell dependencies set up by the user. The RecipeSheet also includes features designed specifically to address the challenges discussed in the Introduction. We begin by showing those features at work on our three sample tasks, and afterwards provide details of the underlying design.

Example: Drawing together diverse resources

The first example illustrates how the RecipeSheet can be used to bring together diverse resources and to coordinate their presentation. Figure 1 shows a user-built sheet that retrieves information about a researcher. When a name is entered in the *name* cell at top left, three kinds of information are retrieved: the *keywords* cell shows a list of keywords extracted from a publication search performed in the ACM's Digital Library; the *dblpHistory* cell shows a chart with year-by-year entries for this author in the DBLP publication database; and the cell *googleFirst* shows the top result from a Google search on the name.

Figure 2 shows some steps in setting up this sheet. In the RecipeSheet, inter-cell dependencies are not defined in terms of formulas but using an abstraction that we call a recipe. A recipe has named input values (referred to as ingredients) and named results; its role in the sheet is set up by connecting the ingredients and results to cells or to other recipes. The processing carried out by a recipe can be defined in any of several languages.

Because of the richness of the connections possible on a recipe sheet, setup is carried out in a separate mode in which recalculation is suppressed and all cells' contents are hidden. In Figure 2 a user starts with a blank sheet and adds a first recipe (1), thus automatically entering setup mode. The recipe is represented as a configuration box, which includes labels for ingredients (on the left) and results (on the right). Double-clicking the *name* label adds to the sheet a cell to supply that ingredient, and creates a dependency connection, shown as an arrow, from the cell to the recipe. By stage (2) the user has likewise set up a cell for this recipe's single result, which is the list of keywords extracted from an ACM page; Figure 3 shows an example of how this may be done by a recipe defined using a short Small-

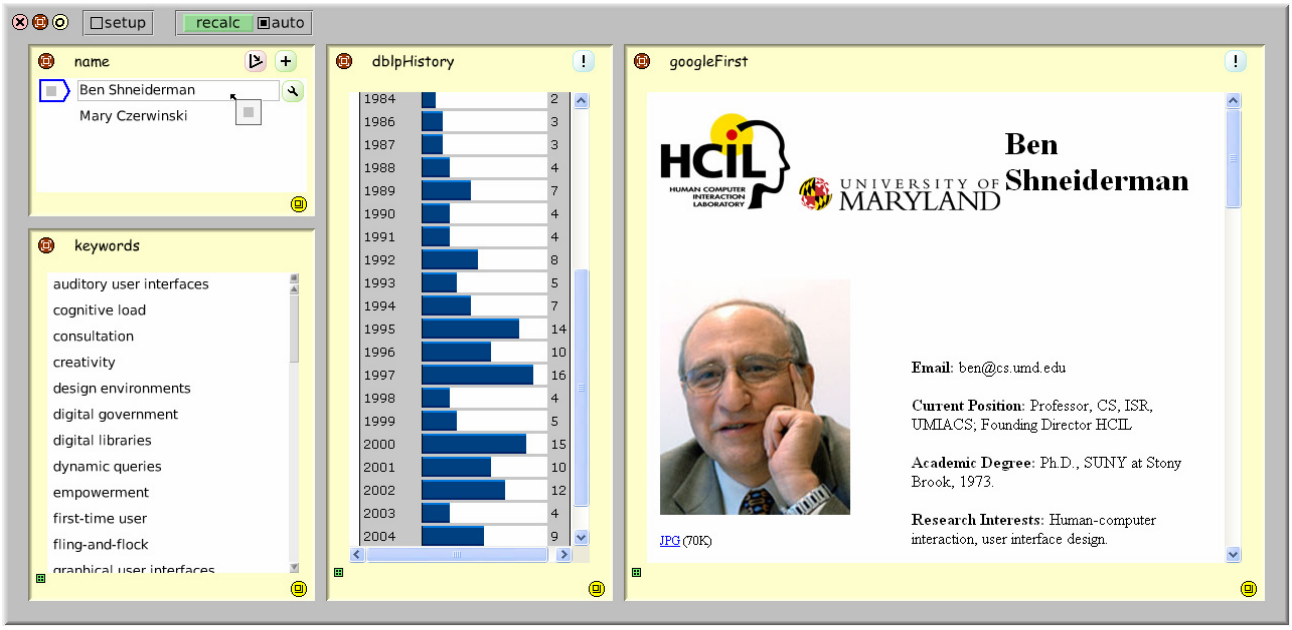

Figure 1. A sheet showing the results of three lookups based on the value in the *name* cell

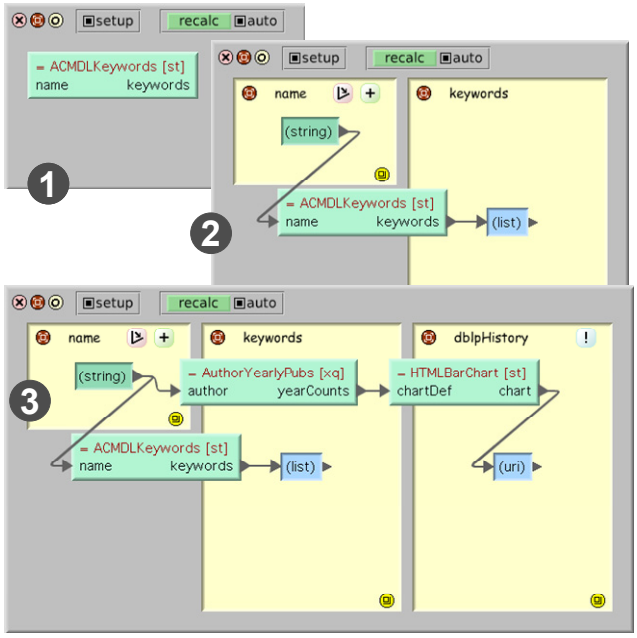

Figure 2 Three stages in setting up the sheet shown in Figure 1: (1) Introducing a first recipe to the sheet. (2) After setting up ingredient and result cells for that recipe. (3) After introducing and connecting two recipes in series giving a second result

talk method. At stage (3) the user has also set up the DBLP history cell, which also depends on *name* but uses two recipes connected in series. The first, written in XQuery, accesses a local version of the DBLP database and creates a result table delivered as an XML value. The arrow connecting this recipe to the *name* cell was created by dragging with the mouse. The second recipe, written in Smalltalk, turns this XML value into a page of HTML defining a chart. The setup of the *googleFirst* cell, not shown here, depends on a recipe that was captured from google.com using the mechanisms demonstrated in C3W [6].

This example illustrates the handling of diverse resources in that it brings together information based on Smalltalk, XQuery and C3W processing, and result displays that include a simple textual list, a segment of rendered HTML, and an entire Web page.

Figure 1 reveals a further distinctive feature of the RecipeSheet: that an ingredient cell can contain more than one value. The user has put two values ('Ben Shneiderman' and 'Mary Czerwinski') into the *name* cell. Currently the first is selected, so it is being used as that cell's value. Simply clicking on the second value would cause the sheet to calculate and display the results for Mary Czerwinski instead. It is also possible to select multiple values in parallel, calling on the RecipeSheet's subjunctive-interface capabilities to set up parallel scenarios. This is one of the features seen at work in the next example.

Example: User-driven filtering of intermediate results

Our second example centres on how the RecipeSheet lets a user intervene to choose which values get passed from one

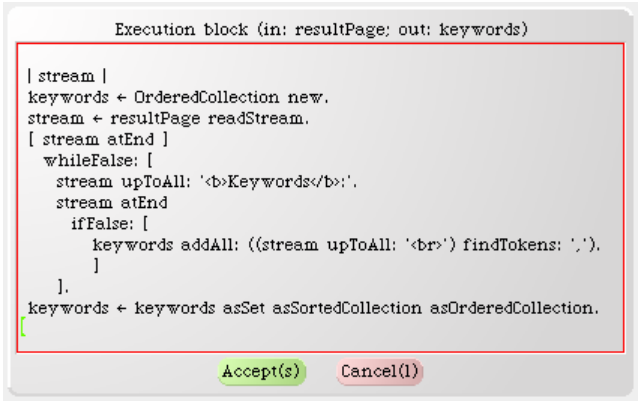

Figure 3. Editing a Smalltalk recipe that extracts keywords from the HTML source of an ACM search-results page.

recipe to the next. Figure 4 shows a sheet built in collaboration with bioinformatics researchers at Hokkaido University. It constitutes a series of processing stages involving Web-based and local bioinformatics tools, including a stage where the user makes selections among the results of one tool to act as the inputs to another.

The first stage, a recipe that supplies the *results* cell, performs a homology search (using BLAST, the Basic Local Alignment Search Tool) on a protein sequence specified by the user in *seq*. The search is performed against the database selected in *database*. This recipe was captured as a C3W recipe from an HTML-form-based interface to BLAST. In the original Web application the user selects the database from an HTML drop-down selection widget. This widget can only take a single value at a time; the user has to choose a single database against which to run a search. However, when the captured application is set up on a recipe sheet, database selection is driven by the list widget in an ingredient cell. As mentioned above, these widgets can support multiple selections in parallel, thus setting up multiple scenarios. In Figure 4 the user has selected the human, mouse and rat databases, thereby creating three parallel searches. Their respective results are shown on colour-coded tabs in the *results* cell, where the tab colours match the markers against the database names.

This example also shows a subjunctive-interface feature that is new to the RecipeSheet: support for processing that merges results from multiple scenarios. In this case, selections made in distinct scenarios within *results* can either be used for separate scenario-specific processing or merged into a single set. This is available because the recipe for the second processing stage has been specified as a cross-scenario recipe. Any cross-scenario recipe is also capable of processing a single scenario at a time; whether it does so or not is determined by a specialised crossScenario ingredient, here connected to the switch in the *xScenario* cell. In Figure 4 the user has chosen to have the values merged, so the display in the *atv* cell (showing a phylogenetic-tree tool, running locally as an applet) includes the selected sequences from all scenarios. The presence of three tabs in

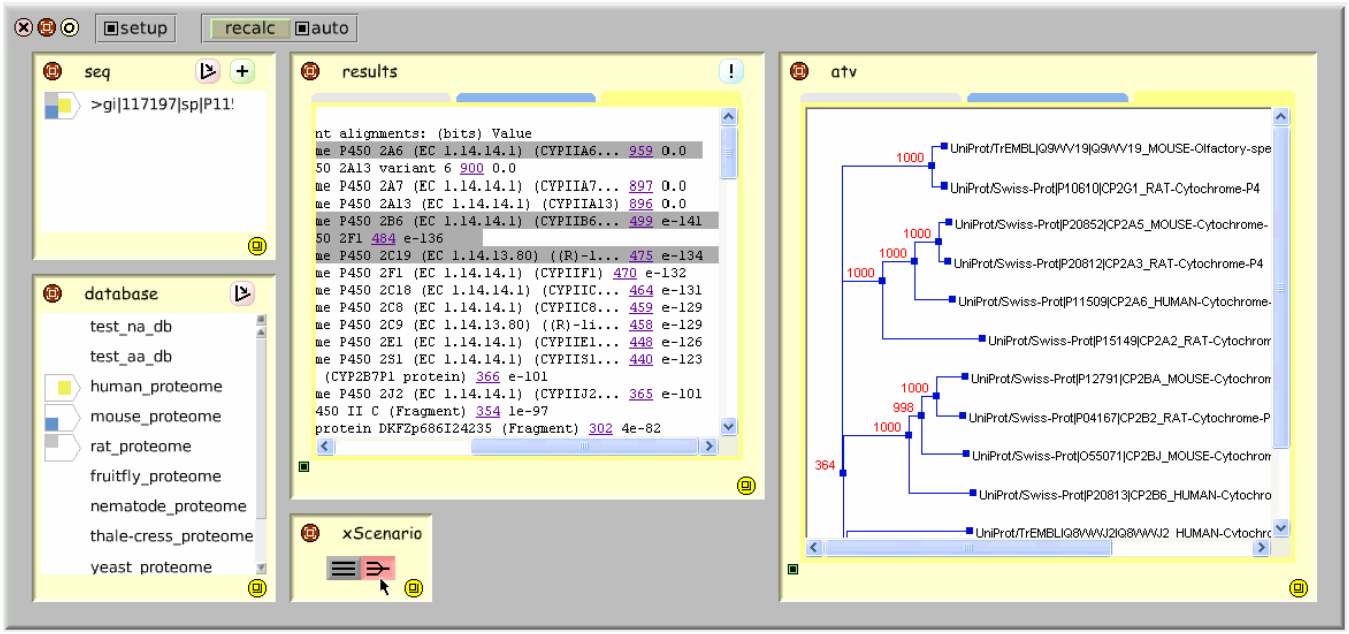

Figure 4. A sheet with multi-stage evaluation based on a combination of Web-based and local bioinformatics tools. The user makes selections among the results of the first stage to drive the subsequent processing. Here results from three scenarios are combined into a single final chart.

the *atv* cell confirms that the cell's contents are drawn from three scenarios. However, downstream of the merge point all the scenarios contain the same values, so the sheet's processing logic automatically executes the *atv* cell's recipe just once.

For some kinds of repetitive processing, user intervention – such as the filtering of intermediate results seen here – is crucial. This example shows how the RecipeSheet can provide appropriate support.

Example: Exploring alternatives

This final example demonstrates the facilities available in the RecipeSheet for exploring multiple alternatives. Figure 5 represents a user dragging ingredient values to a recipe for checking the availability and price of hotel rooms when planning a trip. In the lower part of the figure is a recipe setup that filters the user's list of favourite hotels, according to the selected city. Having requested the Tokyo hotels, the user picks those whose location would suit his plans for the trip, and drags them to the *hotel* cell on the upper sheet. Each item that gets transferred is a chunk of XML whose elements include details of the hotel's name and address, and a pointer to a recipe for checking room availability and price for that hotel. The user has previously captured these recipes from Web sites, using C3W; for some hotels the recipe comes from the hotel's own site, while for others it comes from a travel site through which the hotel offers discounts. At this level of control over processing, the kind of recipe makes no difference. All room-checking recipes accept as ingredients the check-in date, number of nights and number of people, and return the total price of the cheapest available room. As shown in Figure 6, the recipe setup for this sheet involves one recipe that extracts the roomChecker element from the selected hotel item, and a second recipe that uses this element as the Procedure ingredient for actually checking room availability. Note that it uses the default value of 1 for the number of people.

Figure 7 shows two situations where multiple scenarios are on display at the same time. In the upper view, the user has set up the numbers of guests and nights, and a few alternative dates that are of interest. Clicking what we call the pivot button on a cell that contains multiple values generates a set of scenarios, one for each value; in this case the

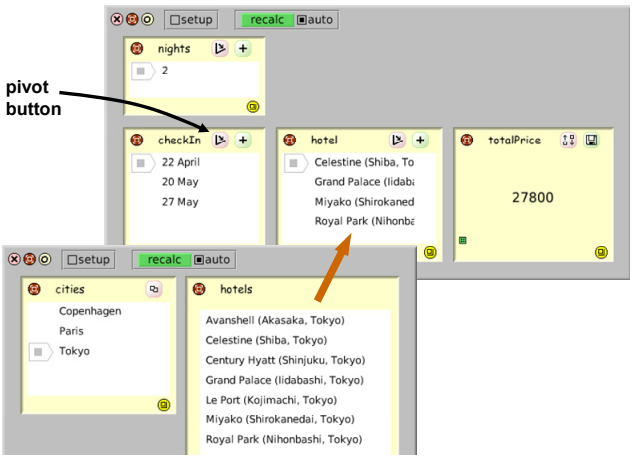

Figure 5. At top, a sheet for exploring room availability at a hotel picked by the user. XML-encoded data items representing hotels are dragged from the lower sheet, which accesses a repository of the user's favourite hotels. Each hotel item includes the details of a recipe for checking room prices at the hotel.

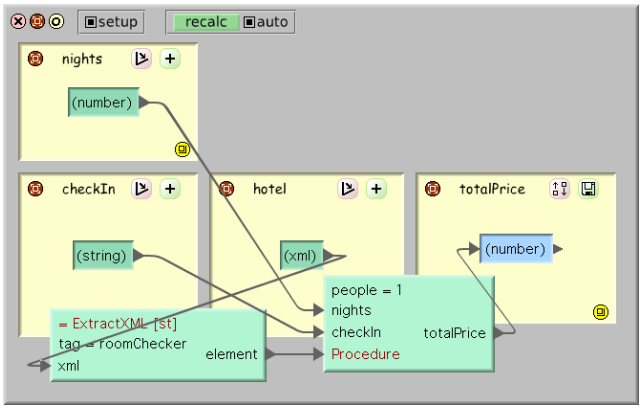

Figure 6. The recipe setup in the room-availability sheet. One recipe extracts the roomChecker element of the currently selected hotel item, then passes this as the Procedure ingredient of a recipe that actually makes the enquiry.

user has clicked the *checkIn* cell's pivot button. The resulting display includes, side by side, scenarios that check room prices at the selected hotel for each of the three specified dates. The values in the *totalPrice* cell are displayed with layout and colouring that are correlated with the markers in the ingredient cells. The side-by-side displays help the user to compare values for a given result (in this case, price) across scenarios, while the correlated arrangement helps the user to see which ingredient values were used in creating each result.

The lower part of Figure 7 shows a more complex exploration, involving two dates and four hotels. In this case the user has also clicked the sorting button on the *totalPrice* cell, to rearrange the scenarios so that the smallest value appears at top left and the largest at bottom right. Thus the positions of the coloured blocks in the markers next to each hotel give an indication of the relative cost of that hotel. Understanding all the correlations available in such a display may take some time for a user who is new to subjunctive interfaces, but in studies [14] we have observed that with practice users can obtain the information for multiple scenarios more quickly and easily than if they had to evaluate the scenarios individually.

One contribution of this example is illustrating that with a RecipeSheet it is as straightforward to create parallel scenarios based on different recipes as on different ingredient values. This arises from our design of the recipe abstraction, in which the processing to be carried out by a recipe is treated as just another of the recipe's ingredients. In the next section we explain this abstraction in more detail.

Recipes and Ingredients

The RecipeSheet's processing abstraction is based on an image of an amateur cook referring to a detailed recipe, which specifies the ingredients to use and the procedure for cooking them. In such a recipe, making small changes either to the ingredients or to the procedure may lead to results that differ in interesting ways. In this sense the in-

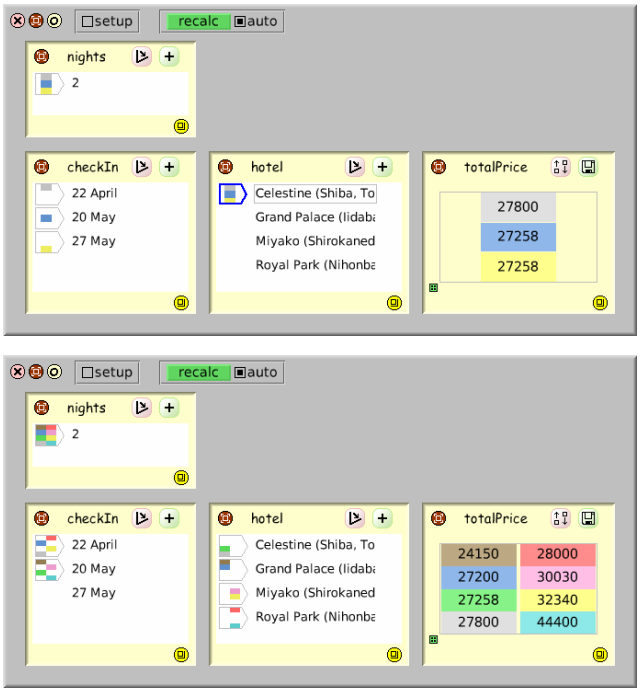

Figure 7. Above: Simultaneous exploration of three dates for a single selected hotel. Below: Eight scenarios, defined as combinations of two check-in dates for each of four hotels. The scenarios have been sorted on the basis of their price results.

gredients and procedure have equal status, and the cook is free to explore by varying either.

By contrast, in end user programming environments the ingredients and processing typically have unequal status. In a spreadsheet, for example, though it is easy to try alternative values in cells it is relatively hard to try alternative formulas. But there are many information-handling situations where it is the processing that the user wants to vary; a simple example is wanting to try a given query against two or more online search engines. The RecipeSheet, by using the recipe abstraction, is an environment where such variation becomes straightforward.

The ingredient types currently supported by the RecipeSheet include string, number, Boolean, URI, HTML, XML and recipe. The fact that recipes themselves can be handled as ingredients is a key feature of the design. Recipes are either primitive or generic. Generic recipes have the structure shown in Figure 8; they include declarations of the name and type of each ingredient and result (the Procedure ingredient always being of type recipe), and optionally a default value for each ingredient. These names and types together make up the recipe's signature.

Every recipe handles the message *evaluate(ingreds)*, where *ingreds* is a list of {*name, value*} pairs for some or all of the recipe's ingredients. A generic recipe handles this by adding to the *ingreds* list default values, where available, for any ingredients not supplied in the original message. It then forwards the message to the recipe held as its Procedure ingredient. In this way, eventually the message will

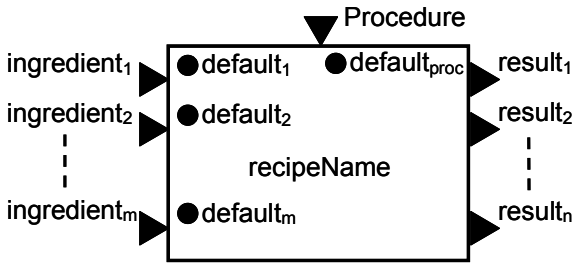

Figure 8. Schematic (from [12]) for a generic recipe, with connections for a list of named ingredients and a list of named results. Optional default values for ingredients are shown as black dots.

arrive at a primitive recipe. The RecipeSheet currently supports primitive recipes written in Smalltalk, ooRexx (Open Object Rexx) and XQuery, Web application recipes captured using the C3W techniques, and Web service recipes accessed through SOAP. In addition there are composite recipes, which encapsulate a combination of recipes such as the setup on a recipe sheet. Each form of primitive recipe offers interfaces for creating and editing the recipe. Smalltalk, ooRexx and XQuery recipes, for example, let the user edit a string defining the code or query to be executed.

It is clear from the above description that the recipe in a recipe's Procedure ingredient must match the signature of the generic recipe itself. For example, each of the hotel room-checking recipes must handle values for *nights* and *people*. That said, the RecipeSheet includes simple heuristics for ad-hoc matching of signatures based on ingredient and result type, even if the names do not agree. Thus a generic recipe that has one string and one numeric ingredient, and one string result, will accept as its Procedure value any recipe that has exactly one string ingredient and one numeric ingredient, and exactly one string result.

One role for such heuristics is in supporting recipe variation that is especially dynamic, such as Procedure-ingredient values (i.e., recipes) that are themselves created on the fly by other recipes. This is useful, for example, in creating multi-stage processing pipelines from recipes chosen dynamically by the user. The fact that at least simple signatures can be matched without name agreement eases the setup and use of this kind of facility.

The RecipeSheet environment

The various specialised features of the RecipeSheet require interface mechanisms beyond those of a standard spreadsheet. Some of the key differences are as follows:

1. Whereas a spreadsheet formula typically calculates the value for just a single cell, a recipe may deliver multiple results.
2. Unlike the workings of a formula, a recipe's processing can itself change dynamically.
3. A recipe can have ingredients supplied directly from the results of other recipes, not just from cells.
4. Instead of each cell containing one value, a cell can contain many values. The user chooses dynamically which value will be used, or can choose multiple values for multiple scenarios.

The principal entities for users to work with are cells, recipes, and scenarios.

Cells

Cells are places for viewing and interacting with ingredients and results. Cells are created by the user, typically to be associated with an ingredient or a result of some recipe. Each cell is thus fixed as being either an ingredient or result cell, and has a fixed value type; these characteristics determine the cell's appearance and interaction features. The preceding examples include ingredient cells for supplying numbers, strings and XML entities, and result cells for displaying numbers, lists and HTML pages. Naming and arrangement of cells is up to the user, with the constraint that each cell on a sheet must have a unique name.

The basic form of an ingredient cell is a list of candidate values, within which the user selects one value to be used when the sheet's values are recalculated. If the cell is providing the value for an ingredient with a fixed range (e.g., the BLAST database selection), the list cannot be edited. If the value range is not fixed, the user can edit existing values and can add new ones, either by use of a dialog box or by dragging values of appropriate type from elsewhere. In Figure 7 the *checkIn* cell has values that were entered using a dialog box, while the *hotel* cell is filled with XML values dragged from a cell on another sheet.

Result cells are in principle for displaying values, so they do not allow their contents to be edited. However, for some value types (e.g., HTML) a result cell lets the user augment the content with mark-up, such as by highlighting certain elements. When a cell holding the result of some recipe is connected as an ingredient of another, the second recipe can access the details of this mark-up. An instance of this is the cell *results* in the bioinformatics example (see Figure 4), where the user selects items within the display to specify which results are to be used by the next processing stage.

Recipes

Recipes define the dependency relationships between cells, as well as the processing used in evaluating those dependencies. A recipe can be added to a sheet in various ways: it can be chosen from the RecipeSheet's global repository, which is accessed through a sheet's pop-up menu; it can be dragged from a cell where it appears as a list entry (recipes are values like any other, so they can appear in cells); or the user can request a new custom recipe of type Smalltalk, ooRexx or XQuery. A custom recipe starts with no ingredients or results, and no procedure, but provides edit controls to let the user specify them.

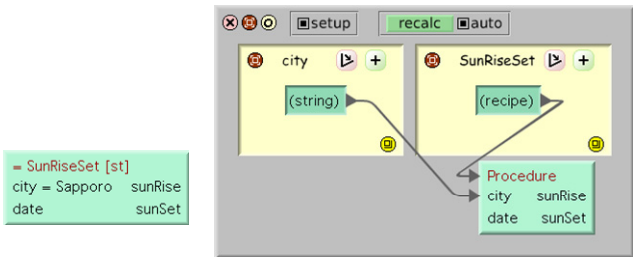

Figure 9. Starting to set up a recipe. Left: the initial configuration box for the SunRiseSet recipe, which has a default value for the 'city' ingredient. Right: the situation once 'city' and the Procedure ingredient have been connected to cells.

As shown on the left within Figure 9, a recipe first appears as just a configuration box showing the ingredients and results. The user interacts with this box to specify the recipe's role on a sheet. Figure 9 shows two stages in setting up a Smalltalk recipe that calculates sunrise and sunset times, given a city and date. This recipe has 'Sapporo' as the default for its city ingredient. If that is acceptable to the user, this ingredient can be left unconnected. Alternatively, if a different fixed value (say, 'Copenhagen') is wanted, the user can edit the default to the new value. In the case shown here the user doesn't want a fixed default, and has therefore double-clicked on 'city' to create a string-type cell to supply this ingredient. This user also wants to be able to specify different values for the processing – for example, perhaps to test the sunrise calculation by comparing its results with those from an online weather service. Notice that when an ingredient is connected, whether from a cell or from the result of another recipe, its default value is no longer displayed.

Figure 10 uses a different recipe (BLAST, as seen in the bioinformatics example above) to illustrate support for so-called aspects. The user has double-clicked 'results' on the BLAST configuration box, to set up a cell to show the search results. For a result of type HTML, as in this case, this brings up a dialog to let the user choose from among various aspects. The 'default' aspect, which was used in the setup in Figure 4, renders the HTML in an Internet Explorer control. Alternative aspects include the raw HTML, and the text content obtained by stripping away all tags; either of these would set up a cell that displays just a plain text string. Aspect selection is also sometimes needed when connecting a recipe's ingredients. For example, the second stage of processing in Figure 4 takes as its ingredient the 'selectedElements' aspect of the *results* cell.

Recipe setups are subject to the constraint that while many connections can have the same source, no two connections can share their destination.

Scenarios

As shown in the bioinformatics and hotel examples, the RecipeSheet supports the basic features of a subjunctive interface [11,13,14], namely that multiple scenarios defined in terms of alternative ingredient values can exist at the same time, can be shown side by side, and can be adjusted in parallel. The interface elements that enable this behaviour are what we call widget multiplexers: the multi-scenario input widgets through which a user makes selections in ingredient cells, and the multi-scenario display widgets for result cells. Further details are given in [12].

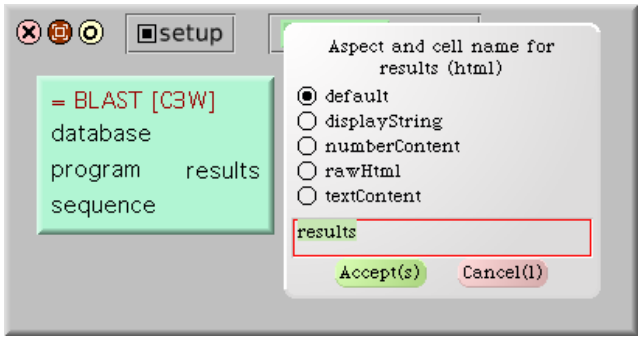

Figure 10. Choosing what aspect of a value to display, after double-clicking on the HTML-valued result of the BLAST recipe.

The motivation behind the subjunctive-interface approach is our belief that letting a user work with multiple scenarios side by side can improve support for exploration, for example in decision-making tasks.

Saving and restoring

A structure of cells and recipes built by a user can be saved in two forms for later restoration. First, the entire structure can be considered as a single composite recipe. The ingredients of this recipe are defined by the names and types of the ingredient cells, and results likewise by the result cells. This recipe can be added to the global repository under a name provided by the user, allowing it to be retrieved for use on sheets created in the future. The second form of saving includes the further information needed to capture a snapshot of an entire sheet. This includes the layout of cells on the sheet, the values within them, and whatever scenarios have been established by ingredient-value selections. Restoring the latter saved form cannot be done onto an existing sheet, but creates an entire new sheet.

RELATED WORK

A comprehensive review of technologies relevant to gathering, manipulating and exploring data is outside the scope of this paper. Here we highlight categories of existing research that relate to the RecipeSheet.

Support for unified access to differing kinds of information is a key goal in the area of personal information management [9,17]. Tools such as Haystack [1], Stuff I've Seen [5] and Hunter Gatherer [16] allow users to create and maintain custom collections of information. Some systems go further, enabling users to create custom collections of tools, or what we refer to as information processors. C3W [6] and Chickenfoot [2] are specifically related to Web-based content. A1 [8] offers access to both Web-based and local tools, of a range of types.

Another class of systems supporting construction of custom combinations of processing tools is environments for scientific workflow. Indeed, the RecipeSheet's connection mechanisms borrow ideas from the interactive wiring interfaces of systems such as Taverna [15] and Kepler [10]. Our implementation of data aspects is an alternative approach to the use of 'shim services' in Taverna for making frequently needed standard data conversions. Our support for users to create custom recipes using a programming language of their choice is similar to the scripting mechanisms available in these systems.

Scientific workflow systems in general emphasise the construction of processing pipelines that, once started, run without user intervention. In contrast, spreadsheet-style interfaces tend to emphasise the display of values rather than processing, and are thus better for supporting user intervention in the course of processing. The spreadsheet approach has already been applied to various domains beyond its original role in personal finance. Some researchers have exploited the standard spreadsheet's tabular structure and built-in operations for duplicating calculations, in order to let users build and manipulate processing flows side by side [4,7]. By contrast, projects such as Forms/3 [3] have shown a variety of powerful ways in which the spreadsheet model can be extended.

However, a limitation of the above systems compared to the RecipeSheet is that although the values flowing through the processing setup are expected to change, either with the passing of time or in response to new user inputs, the processing flow itself is expected to remain static. In a traditional spreadsheet, for example, a user who wishes to switch from examining scenarios that differ with respect to some parameter *A* to scenarios that differ with respect to parameter *B* must typically go through laborious modification of the formulas on the sheet. The scenario-management facilities of Microsoft Excel can be seen as an attempt to work around this limitation, by letting the user define a batch of scenarios whose results are calculated and presented in a dynamically reconfigurable 'pivot table'. Yet such batch-style processing is the antithesis to interactive definition and use of multiple scenarios.

The RecipeSheet sits in the progression of work on subjunctive interfaces [11,13,14], which is driven explicitly by the goal of helping users explore alternative scenarios. Early work on subjunctive interfaces addressed specific application examples, whereas with the RecipeSheet it becomes possible to support applications that users build for themselves: by embedding subjunctive-interface features in an end user programming environment, the setups built by users of that environment become implicitly capable of supporting multiple scenarios. Related work by Terry et al. [18], who describe the lack of support for exploring alternatives in most computer applications as arising from a blinkered 'single-state document model', led to the development and demonstration of Parallel Paths. One difference between subjunctive interface mechanisms and the Parallel Pies interface developed by Terry et al. is that the latter shows multiple results by pie-chart-style tiling of just a slice of each result, rather than the whole display. This avoids having to make room to display multiple entire results, but provides a limited form of side-by-side comparison; unlike in the widget multiplexer of a subjunctive interface, the user can never see and compare see a number of fully rendered alternatives.

In some respects the RecipeSheet is only an evolutionary step beyond other systems mentioned here. For example, it would probably not break the system model of existing scientific workflow systems to introduce subjunctive-interface-style facilities. The main challenge would be in creating appropriate scenario-aware interaction and display components to perform the role of the widget multiplexers used in the RecipeSheet's cells. On the other hand, we believe that the recipe abstraction and its interface mechanisms offer a new degree of flexibility for creating, manipulating and using information processors.

DISCUSSION

The current status of the RecipeSheet is that of a working system that has yet to pass through systematic evaluation. While our experience with the RecipeSheet confirms that at a functional level it enables the construction of useful assemblies, we have yet to test whether its mechanisms are generally understandable and usable. We recognise that the system's core mechanisms – for creation, setup, saving and restoring of recipes – may yet require some cycles of iterative design.

In addition we intend to use the RecipeSheet as the vehicle for our larger evaluation of subjunctive interfaces and their potential to influence users' behaviour in pursuing explorations. Many questions remain unanswered regarding the willingness with which people will set up alternative scenarios themselves, and conversely how they might react to a proactive system that proposes alternatives alongside what the users are doing. The RecipeSheet has the richness needed to start exploring such questions.

We are considering a number of basic system extensions, such as an upgrade to the heuristics used in matching generic recipes against supplied recipe arguments. By incorporating awareness of subtype/supertype relationships we could increase the range of matching that can be performed, without compromising its safety. We may also add support for further primitive-recipe languages, such as Perl or Ruby.

Overshadowing such considerations, we acknowledge that the RecipeSheet's heavy dependence on the Squeak Smalltalk environment renders it relatively separate from the mainstream applications on most users' desktops. This might make such users hesitant to tap into its capabilities. We are investigating how this situation can be alleviated, for example by following the approach taken for Chickenfoot [2] of operating within a Web browser plug-in.

CONCLUSION

Many tasks in modern computing environments require users to access diverse resources, edit or process them locally, and explore the outcomes of alternative processing. We have presented the RecipeSheet, an end user programming environment that provides a unified interface for creating, combining and controlling information processors to accomplish such tasks. The key innovation underlying the RecipeSheet's flexibility is the recipe, a lightweight mechanism for encapsulating information processors and thus making dynamic experimentation with the details of the processing as easy as experimentation with input values. The RecipeSheet has been described through a series of examples, each motivated by challenges that we see in typical user tasks. We are currently refining the RecipeSheet through experience in using it on our own tasks, and plan a series of user studies to evaluate its general usability.

ACKNOWLEDGEMENTS

We gratefully acknowledge the efforts of the developers of the tools on which the current RecipeSheet implementation depends: Squeak, and its recently added bridge to .NET; the eXist implementation of XQuery; and the many bioinformatics tools, and other Web applications and Web services, that we have used as examples. We also thank Jun Fujima for his unstinting .NET programming support.

The development of the RecipeSheet is partly supported by JSPS grant-in-aid number 175000533602.

REFERENCES

1. Adar, E., Kargar, D. & Stein, L.A. Haystack: Per-User Information Environments. *Proc. 8th Intl. Conf. Information and Knowledge Management,* ACM Press, 1999, 413-422.
2. Bolin, M., Webber, M., Rha, P., Wilson, T. & Miller, R. Automation and Customization of Rendered Web Pages. *Proc. UIST 2005*, ACM Press, 2005, 163-172.
3. Burnett, M., Atwood, J., Djang, R., Gottfried, H., Reichwein, J. & Yang, S. Forms/3: A First-Order Visual Language to Explore the Boundaries of the Spreadsheet Paradigm. *Journal of functional programming*, 11, 2 (2001), 155-206.
4. Chi, E.H., Riedl, J., Barry, P. & Konstan, J. Principles for Information Visualization Spreadsheets. *IEEE Computer Graphics and Applications 18*, 4 (Jul/Aug 1998), 30-38.
5. Dumais, S.T., Cutrell, E., Cadiz, J.J., Jancke, G., Sarin, R. & Robbins, D.C. Stuff I've Seen: A System for Personal Information Retrieval and Re-Use. *Proc. SIGIR 2003*, ACM Press, 2003, 72-79.
6. Fujima, J., Lunzer, A., Hornbæk, K. & Tanaka, Y. Clip, Connect, Clone: Combining Application Elements to Build Custom Interfaces for Information Access. *Proc. UIST 2004*, ACM Press, 2004, 175-184.
7. Jankun-Kelly, T.J. & Ma, K.-L. Visualization Exploration and Encapsulation Via a Spreadsheet-Like Interface. *IEEE Transactions on Visualization and Computer Graphics*, *7*, 3 (2001), 275-287.
8. Kandogan, E., Haber, E., Barrett, R., Cypher, A. & Maglio, P. A1: End-User Programming for Web-Based System Administration. *Proc. UIST 2005*, ACM Press, 2005, 211-220.
9. Karger, D.R. & Jones, W. Data Unification in Personal Information Management. *Communications of the ACM*, *49*, 1 (2006), 77-82.
10. Ludäscher, B., Altintas, I., Berkley, C., Higgins, E., Jaeger-Frank, E., Jones, M., Lee, E., Tao, J. & Zhao, Y. Scientific Workflow Management and the Kepler System. *Concurrency and Computation: Practice & Experience.* To appear (2006).
11. Lunzer, A. Choice And Comparison Where The User Wants Them: Subjunctive Interfaces For Computer-Supported Exploration. *Proc. INTERACT '99*, IOS Press, 1999, 474-482.
12. Lunzer, A. & Hornbæk, K. An Enhanced Spreadsheet Supporting Calculation-Structure Variants, and Its Application to Web-Based Processing. *Proc. Dagstuhl Workshop on Federation over the Web. Lecture Notes in Artificial Intelligence, Vol. 3847* (2006), 143-158.
13. Lunzer, A. & Hornbæk, K. Side-By-Side Display and Control of Multiple Scenarios: Subjunctive Interfaces for Exploring Multi-Attribute Data. *Proc. OZCHI 2003*, Brisbane, Australia, 2003, 202-210.
14. Lunzer, A. & Hornbæk, K. Usability Studies on a Visualisation for Parallel Display and Control of Alternative Scenarios. *Proc. AVI 2004*, ACM Press, 2004, 125-132.
15. Oinn, T., Greenwood, M., Addis, M., Alpdemir, N., Ferris, J., Glover, J., Goble, C., Goderis, A., Hull, D., Marvin, D., Li, P., Lord, P., Pocock, M., Senger, M., Stevens, R., Wipat, A. & Wroe, C. Taverna: Lessons in Creating a Workflow Environment for the Life Sciences. *Concurrency and Computation: Practice and Experience.* To appear (2006).
16. schraefel, m.c., Zhu, Y., Modjeska, D., Wigdor, D. & Zhao, S. Hunter Gatherer: Interaction Support for the Creation and Management of Within-Web-Page Collections. *Proc. WWW 2002*, ACM Press, 2002, 178-181.
17. Teevan, J., Jones, W. & Bederson, B. Special Issue: Personal Information Management, *Communications of the ACM*, *49*, 1 (2006).
18. Terry, M., Mynatt, E.D., Nakakoji, K. & Yamamoto, Y. Variation in Element and Action: Supporting Simultaneous Development of Alternative Solutions. *Proc. CHI 2004*, ACM Press, 2004, 711-718.