# FAR: An End-User Language to Support Cottage E-Services

Margaret Burnett
Oregon State University
Corvallis, Oregon 97331
burnett@cs.orst.edu

Sudheer Kumar Chekka
Oregon State University
Corvallis, Oregon 97331
chekka@cs.orst.edu

Rajeev Pandey
Hewlett-Packard
Corvallis, Oregon 97330
rajeev_pandey@hp.com

## Abstract

*E-commerce has begun to evolve beyond simple web pages to more sophisticated ways of conducting e-business transactions, such as through electronic advertising, negotiation, and delivery. However, to participate in these advances requires the skills of professional programmers, and end-user owners of small businesses often cannot justify this expense. In this paper, we present FAR, an end-user language to offer and deliver e-services. The novel aspects of FAR are its support of small e-services and its multiparadigm approach to combining ideas from spreadsheets and rule-based programming with drag-and-drop web page layout devices.*

## 1. Introduction

In recent years, e-commerce has opened new business opportunities for both large and small businesses. Some of the technology to take advantage of these opportunities is relatively easy to master, even without the help of professional programmers. For example, an end user can, with the help of drag-and-drop tools, create web pages to advertise products and services.

However, devices that support actually selling the products and delivering the services (or a confirmation of the services), such as JavaScript or Java applets for creating dynamic web pages or Perl for dynamically creating new web pages, are programmer oriented. These devices are not accessible to end users, but even if they were, the internet model of doing business is becoming more sophisticated than even these devices can handle. Recently, products such as Hewlett-Packard's e-speak [14] have emerged, which provide software substrates to handle the advertising of availability, negotiation, and communication between businesses and customers.

We have been working to bring these kinds of capabilities to end-user entrepreneurs and small business owners who do not have a staff of professional programmers to handle the existing programming devices. Toward this end, we have created an end-user language named FAR ("Formulas And Rules") for programming just-in-time custom web pages. The prototype of FAR is based upon e-speak middleware. FAR allows end users to offer and deliver e-services without these users having knowledge of the middleware protocols necessary to offer such services. For example, an end user with a database of information about flowers, stored on a PC and maintained using PC software such as Access, could offer "flower advisor" e-services as a cottage, for-profit, business.

FAR combines ideas from three paradigms: web page layout, spreadsheets, and rules. In FAR, users lay out a sample web page with various kinds of spreadsheet-like cells or cell groups. The spreadsheet paradigm has been demonstrated to be usable by end users, yet is a computationally powerful paradigm, and is able to express graphics as easily as textual values [7].

Spreadsheet formulas are "pull"-oriented: a cell expresses its interest in other cells through references in its formula, and updates cause it to "pull" in new intermediate values for a new computation. In our experience with the spreadsheet paradigm (via the visual spreadsheet language Forms/3 [7]), we have noticed that sometimes it seems more convenient to express computations as "push" computations. For example, whenever a button is pushed, we may want 15 cells to change, in which case it may be more convenient to notify the 15 cells all at once than for each of the 15 cells to repeat the same predicate that watches the button's state. From this observation, we decided to also support rule-based programming in FAR.

In this paper, we present FAR. The new contributions of FAR are:

- It is an end-user language that supports small business owners to offer full-featured electronic services.
- It integrates the spreadsheet paradigm with the rule-based paradigm. Both paradigms are supported as alternative views of the same logic, allowing the user to switch between these two paradigms flexibly.

## 2. Background and related work

### 2.1 Multiparadigm languages

A multiparadigm programming language is a language that incorporates two or more of the conventional programming paradigms [12], or a linguistic framework that does not force the programmer into thinking or working in only one model [3]. Two approaches have been pursued in the creation of multiparadigm languages. One is to add additional paradigms to an existing language to permit users to utilize a new programming style without learning a completely new language. For example, C++ extends C with object-oriented programming features. The other is to seek a true blending of paradigms in a new language. Leda [4] exemplifies this approach.

An important difference between these approaches hinges on *access* to the relevant paradigms. The first allows for a "bridge" between constituent paradigms, with usually explicit transitions from one paradigm to another. The second strives for seamless transition from paradigm to paradigm. FAR aims toward the second goal.

In the visual and end-user language communities, in addition to combining various approaches with drag-and-drop GUI layout, there has been some multiparadigm work. Some of these languages are more like high-level component builders than full languages, in that they allow users to specify portions of programs in different languages (e.g., [10, 19]), whereas others allow the user to choose which paradigm to use within the same language (e.g., [16, 20]). These works are about allowing the user to choose a paradigm when writing a program snippet. FAR supports this as well, but also allows the user to switch flexibly among paradigms after the fact (i.e., for later viewing and editing).

### 2.2 Spreadsheet languages

There have been several research spreadsheet languages. The one that has influenced the development of FAR the most is Forms/3 [6, 7, 8]. Like FAR, in Forms/3 spreadsheet-like cells or cell groups can be dragged off a tool palette and placed wherever desired on a window, and this placement determines the ultimate appearance of the final "program" which, in the case of Forms/3, is a spreadsheet. Another influence of Forms/3 upon FAR is the fact that cell values do not have to be textual; cells' formulas can result in graphical images which can in turn be referred to and operated upon, just as can numbers, text, and so on. Unlike FAR, Forms/3 is not an end-user language per se, although some portions of it have been developed with end-user programming in mind. Other fundamental differences are that Forms/3 is not a multiparadigm language and Forms/3 cannot be used to program e-commerce services. The spreadsheet language Formulate [1, 2] is another spreadsheet language that was born from the same roots as Forms/3. An influence of Formulate on Forms/3 and on FAR is the way of allowing multiple cells in a table to share the same formula.

Several other spreadsheet-oriented research projects have aimed at extending spreadsheet language functionality, but through imperative devices or through connections to other programming languages rather than through ordinary formulas (e.g., [9, 15, 21]). FAR does not use these devices.

### 2.3 Rule-based languages

FAR also incorporates the rule-based paradigm. Rule-based programming was pioneered in the end-user programming community by AgentSheets [17] and KidSim/Cocoa/StageCast [13]. In both of these end-user languages, the user specifies the rules by demonstrating a postcondition on a precondition. The intended users are children, and the problem domain is specification of graphical simulations and games. An important difference between these two languages is that in the KidSim family, the only rules present are those that have been explicitly entered by the user, although they can be collected into "Jars" of similar objects that then follow the same rules. On the other hand, in AgentSheets, graphical rule analogies are supported [17] that allow users to generalize a behavior, such as in different directions on the grid or from one object to another (e.g., "Cars move on roads like trains move on tracks"). FAR does not do either of these kinds of generalizations, but does something different regarding which objects follow the same rules. Another difference is that in AgentSheets and KidSim, rules are specified by demonstration, and in FAR they are not.

Altaira [18] is another rule-based language, and was created especially for the domain of robot control. One difference from FAR, AgentSheets, and the KidSim family is that Altaira explicitly brings out the state-machine nature of the rule-based paradigm. Another difference is that, when multiple rules are enabled, all are fired, according to a priority-based scheme. In contrast to this, FAR's definition of rules prevents overlapping rules.

### 2.4 E-commerce through e-speak

The FAR prototype's ability to actually accomplish electronic commerce comes about through its use of e-speak. E-speak is a collection of software developed by Hewlett-Packard that supports electronic negotiation,

including e-commerce. (It is freely available at http://www.e-speak.hp.com.) Although e-speak has helped and influenced the way FAR is implemented, the end user does not know about this association. We briefly summarize the e-speak features that are necessary for understanding of this paper.

At the core of e-speak is the notion of an "e-speak engine." To make services available or to request services, service providers and customers make Java calls or send XML documents to the local engine. The local engine then propagates this information to other engines. To bring service providers and customers together, e-speak finds services that fit the criteria of the customer requests. For example, if a customer requests advice about flowering plants below a certain price, e-speak will look for services that offer that particular kind of advice and send the customer a list of all such services, allowing the customer to choose one if desired. There is a very basic, built-in vocabulary that is used in these conversations. In addition, it is possible for a service provider to make new, domain-specific vocabularies available. Alternatively, a service provider might decide upon an existing vocabulary devised especially for their type of service, such as CML, an existing, standardized vocabulary that has been devised for chemists (http://www.xml-cml.org).

We have simplified the explanation of e-speak here, omitting details such as advertising services, finding vocabularies, etc., but these concepts are not necessary to understand FAR, either by the readers of this paper or by FAR's intended users.

## 3. Introduction to FAR

As we have said, FAR is an end-user visual language to allow end users to offer e-services. This means it is aimed at end-user businesses, not at customers.

During its design, we evaluated FAR using the Representation Design Benchmarks [23], a design-time evaluation tool for visual programming languages that is

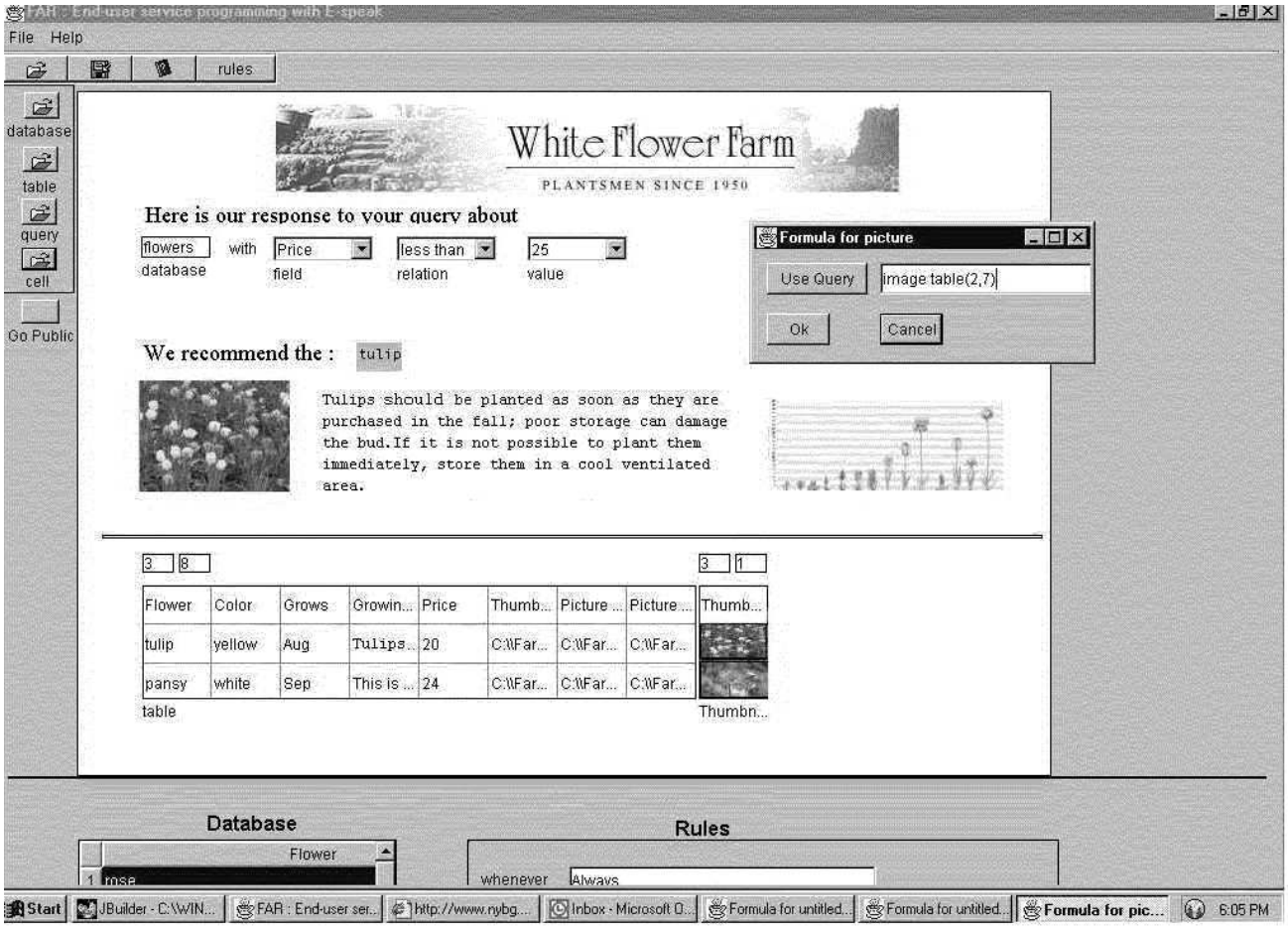


Figure 1: Snapshot of the flower advisor e-service in FAR as it is being created by the gardener. Everything shown in the web page area is a cell or table of cells, including the blocks of text, images, and so on.

based on Cognitive Dimensions [11]. (See [5] for details of this evaluation.) Part of the evaluation process with Representation Design Benchmarks is to explicitly state the prerequisites required of the audience for which the language is intended. This has a bit more accountability than a simple "prediction" of what users will understand, because each assumption the language designer wishes to make about audience capabilities must now be paired with an explicit prerequisite. Thus, the more optimistic the language design team wishes to be about audience capabilities, the more prerequisites they must enumerate. For FAR, the audience prerequisites are some familiarity with browsers, spreadsheet formulas, and database productivity tools such as Access.

To briefly overview FAR, suppose a gardener wants to offer a service that creates a dynamic web page whose contents are to be custom-constructed by retrieving appropriate material from the gardener's PC database. The way the gardener uses FAR to create this service is by laying out a sample web page via direct manipulation, specifying rules and/or spreadsheet-like formulas for dynamically filling in parts of the page based on an incoming query, as in Figure 1. The user specifies as part of these formulas and/or rules the way to retrieve the necessary information about the flower in order to deliver the requested service, such as by looking up the necessary information in an Access database on the user's PC. When the user has completed the creation of the sample web page with its rules and formulas, pushing a button advertises and makes the services available until the user stops making them available.

Some fundamental differences between these capabilities as versus the drag-and-drop authoring tools used by end users to create web pages for search engine access are:

- Although the services created via FAR can be made available for free, part of the querying protocol can include a credit card number, allowing the user to charge for these custom-advice pages.
- FAR is used to specify how to dynamically create web pages on the fly, in some ways similar to the server-side processing available to programmers via cgi scripts, such as for retrieving information from local databases.
- Unlike search engines, the web page produced in response to a customer request always answers precisely the question the customer is asking, and allows a variety of relationships, not just string matching. For example, the figure lists flowering plant sets costing less than $25, and then recommends the lowest-priced one. Asking that query using a standard search engine (Google) yielded "about 98,000" web pages, the first 30 of which did not satisfy the query. (We got tired after looking at 30.) Other search engines we tried (Ask Jeeves and AltaVista) fared similarly or worse.

### 3.1 Programming the flower advisor with drag-and-drop, cells, and formulas

The gardener's process of creating the flower advisor e-service using FAR begins with a blank web page set in a larger workspace. Using drag and drop, the gardener can layout the sample web page by placing objects (cells and tables) in the white web page section of the workspace. For example, the three pictures, textual phrases, and even the line in the middle of the web page section are cells, and at the bottom of the web page section are two tables. Cells `database`, `field`, `relation`, and `value` are instances of a special type of cells called "query cells," and are temporary placeholders for values that will eventually arrive in incoming queries. All cells have attributes such as size, font, color, visibility of names and of borders and of the cell as a whole, which are set by direct manipulation and through pop-up cell attribute menus.

The above mechanisms are static in the sense that their effects on the web page are the same when the web page is delivered as they are when it is specified. The aspect that happens dynamically (at web page delivery time) is the system's choosing of appropriate content for these objects. The gardener specifies this aspect using formulas and/or rules to govern the behavior of all the objects in the web page section. These formulas and rules are evaluated as soon as the gardener enters them to provide immediate visual feedback, and are again used at web page delivery time to compute the up-to-date values to be delivered to the customer.

We focus first on formulas, deferring our discussion of rules. The reason for the use of formulas is to allow end users with spreadsheet skills to reapply these skills to specify the logic of their just-in-time web pages. (Recall that a prerequisite for using FAR is previous spreadsheet experience.) The FAR prototype supports some of the usual spreadsheet operators, and also `if`, `image`, and `whose` operators. For example, "`if x=3 then 10`" returns 10 if `x` is 3, and otherwise the cell's value is "no value" (blank). As another example, the formula shown for the `picture` cell (the picture of tulips mid-left in Figure 1) returns the image stored at the result of the argument, which is the value of `table(2,7)`.

Tables are groups of cells that are allowed to share common formulas, similar to the grids/matrices of Forms/3 [7, 8], Formulate [1, 22], and the shared formulas of the Lotus spreadsheet system. For example, the `Thumbnails` table has been partitioned into two parts: the top cell and

all the others. The top cell's formula is a string that sets the column heading value to "Thumbs," and the remaining cells' formula (shared) is "`image table(thisrow, 6)`", which produces the thumbnail images stored at the path locations listed in column 6 of `table`.

Alternatively, a table can have a single formula defining it as a whole. For example, the `whose` operator fills in an entire table with the result of a query, such as the formula for the table named `table`. This formula (not shown) is a group of references to the query cells along the top of the page, which currently evaluates to "`flowers whose Price less than 25`". The `whose` formula is automatically generated when the user presses the "Use Query" button in the formula window.

When the gardener is finished setting up formulas dependent on entries in `table`, some of the columns are made invisible, such as those giving filenames of images, so that they will not actually appear on the web pages that are ultimately delivered to customers.

### 3.2 Tying an incoming query to local PC information

The gardener's FAR program needs to retrieve data from a PC database, which the gardener has previously created using some widely used software package such as Access. To set up a relationship with this database, the gardener clicks on the database button in the tool palette in Figure 1, and chooses the appropriate database. Once this is done, any cells and tables on the page can refer in their formulas to elements of the database. For example, as explained above, the table at the bottom of the sample web page being laid out by the gardener is referring to a subset of the database, namely the elements of the database that have the desired price.

Near the top of the sample web page are the query cells, labeled `database`, `field`, `relation`, and `value`. The gardener placed these query cells using the query button on the tool palette. The query consists of several cells to individually hold each element of a future query against the database the gardener selected. As with other cells, the user can give query cells formulas, so as to have a sample query to work with while creating the sample web page. In the figure, the gardener has given the query cells formulas with sample values in them by selecting items from the drop-down menus that reflect the structure of the selected database. Alternatively, the formulas can also be specified by simply typing something in, as the gardener did with the query cell named `value`. As demonstrated above, other cells' formulas can refer to query cells just as they can refer to any other kind of cell.

The reason a "real" query that eventually arrives from a customer will use the same names the gardener used to label the query cells is due to the e-speak abstraction of "vocabularies." For example, the query cells in Figure 1 reflect the vocabulary to be used by customers to request the gardener's e-service. In the e-speak world, an e-service such as the customized flower advisor being created here, is registered and advertised in the e-speak electronic community with an accompanying vocabulary, so that customers interested in the service can make use of it. A vocabulary is a set of terms for defining a service. The FAR runtime system can automatically generate a new vocabulary based on the cell names the user has used to label the elements. The vocabulary is automatically made public as part of the advertisement of the service, and these functions are automatically performed by the FAR system. Thus, users of FAR (cottage business owners such as the gardener) are only naively aware of this vocabulary and generation of this vocabulary, since it is automatically taken care by the system.

Conversations about e-speak vocabularies are conducted by transmitting and receiving XML documents, and hence FAR makes use of XML for this purpose. As described in the preceding paragraph, a service can provide its own (new) vocabulary, or it can make use of standardized vocabularies that have been created by standards organizations for particular types of businesses. Given an existing vocabulary of interest, the FAR system automatically generates a query template to match that vocabulary, from which the user can then delete elements that are not to be part of the service offered, and can then proceed with providing sample formulas for the remaining elements.

### 3.3 Rules

Rules can be viewed as a network of constraints, but the expressive power tends to favor the predicate: a single rule will often include one predicate and all the desired effects (a "push" expression). Spreadsheet formulas also can be viewed as a network of constraints, but with the expressive power favoring the consequent: a cell's value is expressed in terms of a combination of all the different predicates that affect it (a "pull" expression). FAR leverages the common denominator by allowing the end user to opportunistically switch between these two programming paradigms at any point. The way this is done is that every cell with a matching predicate is automatically defined to be a participant in the same rule.

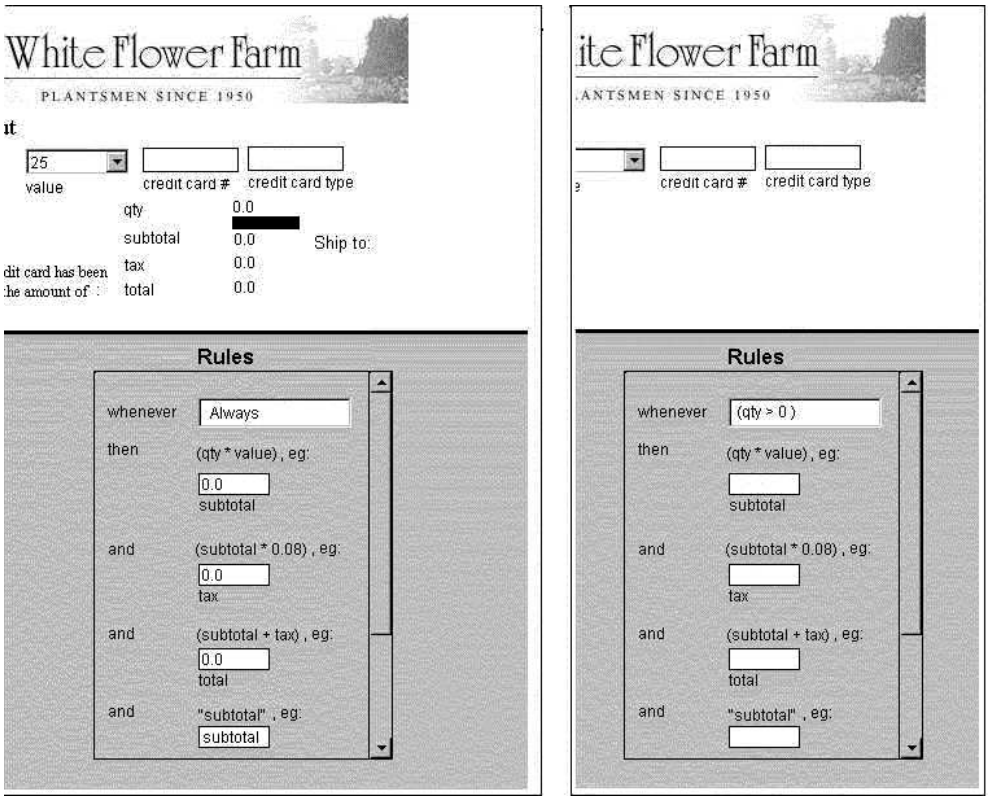
**3.3.1 What is a matching predicate?** A cell *C* with an `if`-expression "`if` *predicate* `then` *consequent-expression*" can be described by the tuple (*C, predicate, consequent-expression*), where *C*'s value will be *consequent-expression* if *predicate* is satisfied, and

otherwise will be the value "no value" (displays as blank). A group *TC* of table cells with a `whose`-expression "*database* `whose` *field relation-operator value*" can similarly be described as $\{(C_{ij}, predicate, consequent\text{-}expression_{ij}) \mid C_{ij} \in TC\}$, where *predicate* = *database.field relation-operator value*, and $C_{ij}$'s value will be $consequent\text{-}expression_i$ (the j'th field in the i'th database entry that matches *predicate*). Any cell that does not have an `if`- or `whose`-expression has the predicate "always." Using the above terms, all cells with the same predicate in the above definitions are participants in the same rule. This rule can be described as (*predicate, {C, consequent-expression}*), for all *C* with predicate *predicate*.

**3.3.2 Using rules.** When the user selects a cell, its rule is automatically displayed in the Rules section. For example, in Figure 2(a), the user has selected cell `subtotal` (indicated by the black selection bar just above it). It has the same predicate ("always") as 10 other cells, and the rule involving all of these cells is displayed. The "whenever" label shows the predicate, and the "then/and" labels show the consequents for every cell affected by this predicate. The user can choose to edit the rule (predicate, consequents, or both) or any of these cells' formulas; the effects of the edit are propagated throughout the display so that the formulas and rules remain consistent. In other words, formulas and rules are alternative views of the same information, and either view can be edited at will.

**3.3.3 An example.** As Figure 2 indicates, the gardener has decided to expand the flower advice program. The gardener now has included some query cells allowing a customer to not only buy flower advice, but also to buy flowers according to the recommendation, if desired. If the customer does not provide quantity information, the sample values (such as 0 for `qty`) will remain unchanged in the gardener's FAR program, and the gardener wants the program to operate as before. However, there is a problem: under that circumstance, the labels and zero values for `subtotal`, `tax`, `total`, `ship to`, etc., will all display, which will look amateurish on a flower advice web page to be delivered back to a customer who never intended to buy physical flowers, only flower advice.

To solve this problem, the gardener needs to change




(a) (b)

Figure 2: The flower advice example has been expanded, and the separator bar has been dragged upwards to make more room for the rules to be visible. (a) The gardener has added cells allowing a customer to purchase the recommended flowers if desired. The predicate of all these cells is initially "always". (b) The gardener changes the predicate, and the results are immediately reflected in the web page section. Since `qty` is currently 0, the predicate is false, and the cells' values are currently "no-value" (displays as blank).

the formula for several cells so that they show values only when the `qty` cell is greater than 0. This can be programmed in a tedious manner by adding a predicate to every relevant formula individually ("`if (qty > 0) then ...` "), but without rules, all those duplicated predicates would introduce a maintenance problem. Expressing these semantics with a single rule solves these difficulties.

To do this, the gardener selects any one of the cells to be changed, such as `subtotal`. From this, all the cells having the same predicate are shown in the Rules section (at this point the predicate would be "`always`"). This is the point at which Figure 2(a) was captured. The gardener de-selects the cells that are not desired to be changed, and then changes the predicate "`always`" to "`qty > 0`" as in Figure 2(b). This also changes the formulas in the selected cells to "`if (qty > 0) then ...`", so the gardener can view and/or further edit these cells in either a rule-oriented way or a formula-oriented way. As a result, the non-applicable labels and values disappear.

### 3.4 The runtime system

When the user makes the service available by pressing the "Go Public" button, the runtime system is started. When it is first started, it connects to the e-speak community, and registers and advertises the service and the vocabulary to be used in retrieving the service.

Customers discover the gardener's service and request it through e-speak. No special features are required of FAR to make this happen. The fact that the customer needs to know the query terminology is taken care of by using vocabularies, as we have already described. At this point, the FAR engine simply goes to sleep until a query arrives from a customer via the local e-speak engine's connection with the rest of the e-speak community.

When a query arrives, the query cells in the sample web page are updated with this query from the customer. For example, if the query was a request for a flower that starts growing in September, the table of flower choices at the bottom of the page will be different, and the recommendation cells at the top will contain a different value. This activity is all done automatically in background mode, and does not generate screen activity on the gardener's screen.

FAR is evaluated lazily, subject to the constraint that every object on the screen is always kept up-to-date. When a formula needs another object's value, the latter's value is demanded. This strategy amounts to about the same thing as eager evaluation during the programming process, since everything is on the screen at that point. However, when the program is later invoked to satisfy a customer request, the lazy evaluator strategy allows omitting computations that are not necessary for the particular request. Efficiency of serving customer requests matters to the gardener, since the program can become active anytime a query electronically arrives, regardless of whether the gardener is using the PC for other purposes at the same time.

When evaluation is complete, the web page's current values and formatting information are electronically delivered to e-speak and hence to the customer in the form of an XML document based on the flower-based vocabulary. Since our language deals with providing services (not purchasing services), FAR does not control what the customer does with this XML document. However, in our prototype implementation, query results are delivered as an XML document to which an XSL (Extensible Stylesheet Language) is attached. Style sheets describe how documents are represented, and using style sheets with structured documents like XML documents, some browsers (such as Internet Explorer) automatically display XML documents according to the stylesheets.

## 4. Current status and future work

Our research prototype of FAR implements all of the features described in this paper except the ability to make cells entirely invisible, relative referencing in a formula, making use of an existing vocabulary, and deselecting objects in a rule. The prototype is written in Java and runs on PCs. It currently allows database interfacing only to Access databases, although the language could easily allow access to other popular PC software as well. FAR programs are stored in XML format.

FAR is a new project, and there are many issues left unaddressed. Perhaps the most pronounced is the fact that, although we have used early evaluation devices such as representation benchmarks [23] and cognitive dimensions [11] to help guide the design of this language [5], there have been no experiments involving human users to point out mismatches with the intended audience. Other issues that we have not explored include database updating, such as to log transactions, and credit card authorization and charging. We expect to address the latter by automatically delegating these tasks to e-services available from other providers.

Another interesting opportunity for future work arises from the fact that the goal of FAR has been only to support the e-business owners, not the customers. As such, we have assumed the presence of customer-side software that helps customers discover appropriate e-speak vocabularies, services, etc., and to request such services. These subtasks and others, such as automatically deciding which services to request and what to do with the information that is ultimately delivered, might be well-

served by a customer-side end-user language, and we are considering ways to proceed in this direction.

## 5. Conclusion

FAR is an end-user language for small e-business owners. It supports these users in devising, advertising, communicating about, and delivering electronic services.

FAR is a three-paradigm language that draws upon demonstratedly usable paradigms for end users—drag and drop layout, spreadsheets, and rule-based. The advantage of combining the spreadsheet paradigm with the rule-based paradigm is that it allows the user to express computations either in a "pull"-oriented way or a "push"-oriented way. That is, the user can encapsulate all the logic affecting a cell in that cell's formula, or alternatively can encapsulate all logic about what the cell affects in a rule.

The combination of these paradigms is not simply a matter of supporting both paradigms and deciding for the user which is best. The choice is left to the user, and can be made *before or after* writing code. This is because the use of the rule-based paradigm is as an alternative view of the logic expressed by spreadsheet formulas. (In other words, if the user chooses to view them as such, spreadsheet formulas are an alternative view of the logic expressed by rules and vice versa.) Thus, the user can opportunistically switch from one paradigm to the other.

## Acknowledgments

We thank the members of the Visual Programming Research Group at Oregon State University for their feedback and help with the implementation. This work was supported in part by Hewlett-Packard.

## References

[1] A. Ambler, The Formulate Visual Programming Language, *Dr. Dobb's Journal*, August 1999, 21-28.

[2] A. Ambler and A. Broman, Formulate Solution to the Visual Programming Challenge, *Journal of Visual Languages and Computing* 9(2), April 1998, 171-209.

[3] T. Budd, Blending Imperative and Relational Programming, *IEEE Software* 8(1), 1991, 58-65.

[4] T. Budd, *Multiparadigm Programming in Leda*, Addison-Wesley, Reading, MA, 1995.

[5] M. Burnett and S. Chekka, FAR: An End-User WYSIWYG Programming Language for E-speak: Interim Report, TR 00-60-10, Oregon State University, October 2000.

[6] M. Burnett and H. Gottfried, Graphical Definitions: Expanding Spreadsheet Languages Through Direct Manipulation and Gestures, *ACM Transactions on Computer-Human Interaction 5(1)*, March 1998, 1-33.

[7] M. Burnett, J. Atwood, R. Djang, H. Gottfried, J. Reichwein, and S. Yang, Forms/3: A First-Order Visual Language to Explore the Boundaries of the Spreadsheet Paradigm, *Journal of Functional Programming* 11(2), March 2001, 155-206.

[8] M. Burnett, N. Cao, J. Atwood, Time in Grid-Oriented VPLs: Just Another Dimension? *Proceedings of IEEE Symposium on Visual Languages*, Seattle, WA, September 10-13, 2000, 137-144.

[9] E. Chi, J. Riedl, P. Barry, and J. Konstan, Principles for Information Visualization Spreadsheets, *IEEE Computer Graphics and Applications*, July/August 1998.

[10] C. DiNucci, Tolerant (Parallel) Programming with F-Nets and Software Cabling, *Proceedings of the Workshop on Software Engineering for Parallel and Distributed Systems (PDSE'97)*, Boston, MA, May 1997, 198-209.

[11] T. Green and M. Petre, Usability Analysis of Visual Programming Environments: A 'Cognitive Dimensions' Framework, *Journal of Visual Languages and Computing* 7(2), June 1996, 131-174.

[12] B. Hailpern, Multiparadigm Languages and Environments, *IEEE Software* 3(1), January 6-9,1986.

[13] N. Heger, A. Cypher, and D. Smith, Cocoa at the Visual Programming Challenge 1997, *Journal of Visual Languages and Computing* 9(2), April 1998, 151-169.

[14] Hewlett-Packard, E-speak Architectural Specification, Hewlett Packard Developer Release X.03.03.00, September 2000.

[15] S. Hudson, User Interface Specification Using an Enhanced Spreadsheet Model, *ACM Transactions on Graphics* 13(4), July 1994, 209-239.

[16] M. Münch, A. Schürr, and A. Winter, Integrity Constraints in the multi-paradigm language PROGRES. *IEEE Symposium on Visual Languages*, Halifax, Canada, August 1998, 84-85.

[17] C. Perrone and A. Repenning, Graphical Rewrite Rule Analogies: Avoiding the Inherit or Copy & Paste Reuse Dilemma, *Proceedings of IEEE Symposium on Visual Languages*, Halifax, Canada, September 1-4, 1998, 40-46.

[18] J. Pfeiffer Jr., Altaira: A Rule-based Visual Language for Small Mobile Robots, *Journal of Visual Languages and Computing* 9(2), April 1998, 127-150.

[19] W. Piersol, Object Oriented Spreadsheets: The Analytic Spreadsheet Package, *Proceedings of ACM OOPSLA*, September 1986, 385-390.

[20] S. Shiffer and J. Fröhlich, Concepts and Architecture of Vista -- a Multiparadigm Programming Environment, *IEEE Symposium on Visual Languages*, St. Louis, MO, Oct. 4-7, 1994, 40-47.

[21] T. Smedley, P. Cox, and S. Byrne, Expanding the Utility of Spreadsheets Through the Integration of Visual Programming and User Interface Objects, *Advanced Visual Interfaces '96*, Gubbio, Italy, May 27-29, 1996, 148-155.

[22] G. Wang and A. Ambler, Solving Display-Based Problems, *IEEE Symposium on Visual Languages*, Boulder, Colorado, September 1996, 122-129.

[23] S. Yang, M. Burnett, E. DeKoven, and M. Zloof, Representation Design Benchmarks: A Design-Time Aid for VPL Navigable Static Representations, *Journal of Visual Languages and Computing*, October/December 1997.