An Abstract of the Thesis of

Sudheer Kumar Chekka for the degree of Master of Science in Computer Science presented on July 16, 2001.

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

Abstract approved: Redacted for Privacy

Margaret M. Burnett

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 thesis, 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.

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

by
Sudheer Kumar Chekka

A THESIS

submitted to

Oregon State University

In partial fulfillment of
the requirements for the
degree of

Master of Science

Presented July 16, 2001
Commencement June 2002

Master of Science thesis of Sudheer Kumar Chekka presented on June 16, 2001.

APPROVED:

Redacted for Privacy

Major Professor, representing Computer Science

Redacted for Privacy

Head of Department of Computer Science

Redacted for Privacy

Dean of Graduate School

I understand that my thesis will become part of the permanent collection of Oregon State University libraries. My signature below authorizes release of my thesis to any reader upon request.

Redacted for Privacy

Sudheer Kumar Chekka, Author

Acknowledgment

I am grateful to my major professor, Dr. Burnett of the important ideas, useful hints, precious comments she puts in my work. This thesis was extensively discussed in a series of meetings with her. My appreciation goes to Dr.Budd, Dr. Rothermel and Dr. Phil Sollins for their help and being my committee. Thanks also go to all Forms/3 group members who gave me their valuable suggestions.

Table of Contents

List of Figures

Figure Page

FAR: An End-User Language to support Cottage E-Services

Chapter 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 [HP 2000] 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 programming. Toward this end, we have created an end-user language named FAR ("Formulas And Rules") for programming just-in-time custom web pages.

The design philosophy underlying FAR is to provide programming devices aimed directly at supporting the "what" perspective of the user's goals (such as "I want to deliver a custom web page about a flower"), not the "how" perspective of the underlying software componentry required to fulfill the goals. The prototype of FAR is based upon e-speak middleware. FAR is a domain-specific WYSIWYG language allowing 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 [Burnett and Gottfried 98].

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 [Burnett and Gottfried 98]), 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. Thus, the user may choose to specify behavior by providing rules, if that seems more convenient. The choice between when to use formulas and when to use rules is up to the user, not the system.

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.

In Chapter Two, we discuss the related work and background required for FAR which also includes the e-speak infrastructure. In Chapter Three, we discuss the end-user language FAR followed by implementation details of FAR in Chapter Four. In Chapter Five, we give the details of FAR evaluation using Representation Design Benchmarks and Cognitive Dimensions. The current status and future work is given in Chapter Six and conclusion in Chapter Seven.

Chapter 2: Background and Related Work[1]

FAR combines programming ideas from three paradigms: layout, spreadsheet-based, and rule-based. The idea of allowing the user to specify layout of the input and output objects in a program simply by dragging them off some kind of tool palette and placing them as desired on the screen is now common in visual languages and environments. In FAR, these objects are various kinds of spreadsheet-like cells or cell groups, and FAR follows the spreadsheet paradigm.

2.1 Multi-paradigm languages

A multiparadigm programming language is a language that incorporates two or more of the conventional programming paradigms and a framework that does not force the programmer to use only one model. Creation of such languages is motivated by the observation that many complex problems contain subproblems whose solutions lend themselves to different programming paradigms. If a language were to possess the appropriate paradigms, the problem solution would be expedited by the fact that subproblems could benefit from the paradigm that best expressed a solution.

Some languages 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.

Some multiparadigm languages seek a true blending of paradigms in order to provide a more expressive programming vehicle for general problem solving. Leda [Budd 1995] exemplifies this approach. This approach strives for seamless transition from paradigm to paradigm. FAR aims towards this model.

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 [DiNucci 1997, Piersol 1986], whereas others allow the user to choose which paradigm to use within the same language [Munch 1998, Shiffer 1994]. 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).

---

[1] The contents of sections 2.1, 2.2 and 2.3 are modified from sections of the paper "FAR: An End-User Language to Support Cottage E-Services" co-authored with Margaret Burnett and Rajeev Pandey.

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 [Burnett and Gottfried 1998; Burnett et al. 2000; Burnett et al. 2001], which has been a research vehicle for a number of years. 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. Other influences of Forms/3 upon FAR include the technique of allowing multiple cells in a table to share the same formula and 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 multi-paradigm language and Forms/3 cannot be used to program e-commerce services.

The spreadsheet language Formulate [Ambler 1999; Ambler and Broman 1998] is another spreadsheet language that was born from the same roots as Forms/3. Formulate has been used primarily as a vehicle to research the support of matrix-oriented computations using multiple levels of formulas [Viehstaedt and Ambler 1992; Wang and Ambler 1996], and its support for multiple cells sharing the same formulas is much more elaborate than either Forms/3 or FAR's.

FAR's design goal is also to extend the spreadsheet paradigm adding as little as possible. Specifically, it aims to allow generalized (dynamic) web pages to be "programmed" using a combination of web-page layout and spreadsheet capabilities without requiring end users to have prior training beyond drag-and-drop web page layout and familiarity with ordinary spreadsheet formulas.

Smedley, Cox, and Byrne have incorporated the visual programming language Prograph and user interface objects into a conventional spreadsheet system in order to provide spreadsheet users with graphical interface capabilities [Smedley et al. 1996]. The Prograph approach includes imperative devices and side effects. SIV (Spreadsheet for Information Visualization) is a spreadsheet research effort aimed at supporting information visualization [Chi et al. 1998]. SIV formulas are state modification oriented: the syntax for formulas is "command result_cell arguments". SIV formulas and cellnames can also employ general Tcl code/variables, an approach also followed by Levoy's Spreadsheet for Images [Levoy 1994], an earlier project that influenced SIV. C32 [Myers 1991] is a spreadsheet language that uses graphical techniques along with inference to specify user interfaces. C32 is not a full-fledged spreadsheet

language; rather, it is a front-end to the underlying textual language Lisp used in the Garnet user interface development environment [Myers et al. 1990].

2.3 Rule-based languages

FAR also incorporates the rule-based paradigm. While spreadsheet programming is a "pull" paradigm—cell formulas "pull" the data they need from other cells of interest by referencing them—rule-based programming "pushes" data to necessary objects whenever the right conditions occur. Our motivation for including this paradigm came from our belief that for some kinds of problem-solving, the "pull" approach is easiest, whereas for others, the "push" approach is easier. Further, we wanted to let the user, not the language designer, be the one who decides which is the easiest approach for the problem at hand.

Rule-based programming was pioneered in the visual language community by AgentSheets [Repenning and Ambach 1996; Ioannidou and Repenning 1999] and KidSim/Cocoa/StageCast [Cypher and Smith 1995; Heger et al. 1998], two languages developed concurrently and influencing each other's development to some extent. 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 [Perrone and Repenning 1998; Repenning 1995] 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 support either of these kinds of generalizations, but does something different regarding what objects follow the same rules. Another difference is that in AgentSheets and Cocoa, rules are specified by demonstration, and in FAR they are not.

AgentSheets and Cocoa were both extended for the purposes of controlling robots as part of the Visual Programming Challenge of 1997/1998 [Heger et al.1998]. Altaira [Pfeiffer 1998] is a rule-based language that was created especially for the domain of robot control. One significant difference from AgentSheets and Cocoa 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 strategy sometimes called a "subsumption architecture" in the literature [Smedley et al.

1996], which means that there is a priority order to rules, and hence some rules subsume others. In contrast to this, FAR's definition of rules prevents overlapping rules. In Altaira, unlike AgentSheets, Cocoa, and FAR, the primary rule-entry mechanism is via a rule editor. Isaac [Pfeiffer et al. 2000], a successor to Altaira, uses fuzzy logic instead of the usual crisp rule-based reasoning.

2.4 End-User Programming

FAR is an end-user programming language to create e-services. This section deals with some other end-user languages and related work towards end-user programming.

End users are people who normally use computer applications, but are not professional programmers or computer specialists, and thus have had little or no prior experience in conventional programming. Our interest is in "ordinary" end users, not scientists or others with significant technical training.

One of the end-user programming problems is, "How can ordinary people, who are not professional programmers, program computers?" [Smith et al 1994]. The aim of end-user programming languages is to provide users without a formal programming background a way to do some programming. The idea is to make the power of computers fully accessible to users so that they are not limited to the capabilities of the software they are provided.

There are three widely used approaches to end-user programming, namely programming by demonstration (PBD)/programming by example (PBE), rule-based programming and spreadsheets. We have already discussed rule-based programming and spreadsheets in the previous two sections.

With programming by demonstration, the user instructs the system to "watch what I do", and a programming by demonstration system creates generalized programs from the recorded actions [Cypher 1993]. Programming by example can be considered as a subset of programming by demonstration. Programming by example is when the system generalizes from the example values provided by the user [Cypher 1993].

Cocoa, in addition to being an example of rule-based programming, is an example of PBD. Cocoa records the user's actions and converts them into an executable program. The user actions in Cocoa are entering the rules with before and after parts graphically. This makes Cocoa a PBD system. Cocoa allows children to build symbolic simulations [Smith et al 1994]. It is an object-oriented programming environment, where users create characters and give them rules that determine their behavior in the simulation [Heger et al. 1998]. It has been tested on kids and they

enjoyed working even after the class [Heger et al.1998]. Cocoa enabled them to program interesting and diverse microworlds. The idea behind the design was to compromise between the ease of use and power of the language. Other examples of PBD languages are DIGIS [Bouman et. al. 1994], DOODLE [Cruz 1994], ProDeGE+ [Sassin 1994] and Pursuit [Modugno and Myers 1994] [Modugno and Corbett 1997].

Today's programming may be difficult partly because it may require solutions to be expressed in ways that are not familiar or natural for beginners [Myers et al 2001]. The "natural programming" project examined the ways that end-users express solutions to problems that were chosen to be representative of common programming tasks [Myers et al 2001]. The purpose of examining users' ways of expressing solutions is to guide design decisions for programming languages. One of the results showed that the majority of the statements written by the participants were in a production-rule or event-based style, beginning with words like *if* or *when*. Imperative statements were also observed concluding that the mix of styles improves the usability of the language. One of the studies considered whether the participants (end-users) used pictures or diagrams in their solutions, and found that two-thirds of them did [Myers et al 2001]. Programming systems can accommodate this tendency by supporting some form of graphical specification to express programs.

The idea of FAR's design is that the end-users can use it without any prior special training, provided that they have fulfilled the minimum prerequisites of being comfortable with spreadsheet formulas, database tools like MS Access, and browsers. We have made use of the above mentioned findings from the related work in end-user programming in our design, such as by accommodating graphical specification (drag and drop layout), and by avoiding problematic formula operators such as loops or traditional database SQL-like operators.

2.5 E-commerce through e-speak

The services created in FAR are exposed to the clients using the e-speak infrastructure. The programs in FAR are stored internally in the XML format and the results of the incoming queries from the clients are stored in the XML file to which an XSL stylesheet is attached.

2.5.1 XML

The World Wide Web Consortium (W3C) defined a new standard for data interchange called XML. XML stands for Extensible Markup Language (extensible because it is not a fixed format like HTML). It is a markup language for documents containing structured information [http://www.xml.com]. Structured information contains both content (words, pictures, etc.) and some indication of what role that content plays. A markup language is a mechanism to identify structures in a document.

XML is intended to make it easy and straightforward to use SGML on the Web: easy to define document types, easy to author and manage SGML-defined documents, and easy to transmit and share them across the Web. SGML is the Standard Generalized Markup Language, the international standard for defining descriptions of the structure and content of different types of electronic document.

XML is not a single, predefined markup language: it is a metalanguage -- a language for describing markup languages. A predefined markup language like HTML defines a way to describe information in one specific class of documents. XML allows to defining customized markup languages for different classes of document. XML allows groups of people or organizations to create their own customized markup languages for exchanging information in their domain (music, chemistry, electronics, finance, etc.).

A DTD (Document Type Definition) is a file (or several files to be used together), written in XML, which contains a formal definition of a particular type of document. It sets out what names can be used for element types, where they may occur, and how they all fit together. In effect, a DTD provides applications with advance notice of what names and structures can be used in a particular document type. All the documents having the same DTD are constructed in a conformant manner.

XML removes the following constraints.

1. dependence on a single, inflexible document type (HTML);
2. the complexity of full SGML, whose syntax allows many powerful but hard-to-program options.

Some of the benefits of receiving data in XML format are as follows.

1. It **delivers data for local computation.** Data delivered to the desktop is available for local computation. The data can be read by the XML parser, then delivered to a local application such as a browser for further viewing or processing or the data can be manipulated through script or other programming languages using the XML Object Model.

2. It **gives users different views of structured data**. Data delivered to the desktop can be presented in multiple ways. A local data set can be presented in different views, based on factors such as user preference and configuration.

3. It **enables the integration of structured data from multiple sources into common logical views**. Data can be integrated from server databases and other applications on a middle-tier server, making it available for delivery to the desktop or to other servers for further aggregation, processing, and distribution.

4. It **describes data from a wide variety of applications**. Because XML is extensible, it can be used to describe data contained in a wide variety of applications, from describing collections of Web pages to data records. Because the data is self-describing, data can be received and processed without the need for a built-in description of the data.

5. It **improves performance through granular updates**. XML enables granular updating. Developers do not have to send the entire structured data set each time there is a change. With granular updating, only the changed element can be sent from the server to the client. The changed data can be presented without the need to refresh the entire page or table.

2.5.2 Style Sheets

A style sheet language is a mechanism to describe how the XML document should be displayed [http://www.w3schools.com]. One of these mechanisms is CSS (Cascading Style Sheets), but XSL (the eXtensible Stylesheet Language) is the preferred style sheet language of XML. CSS is limited when compared to XSL in

some aspects. It applies simple formatting to elements in a marked-up data file. It cannot reorder data, combine data from multiple sources, or add text [Darnell 1999].

XSL consists of three parts.

(1) a method for transforming XML documents

(2) a method for defining XML parts and patterns

(3) a method for formatting XML documents

XSL can be used to define how an XML file should be displayed by transforming the XML file into a format that is recognizable to a browser. One such format is HTML. Normally XSL does this by transforming each XML element into an HTML element. XSL can also add completely new elements into the output file, or can remove elements. It can rearrange and sort the elements, and test and make decisions about which elements to display.

2.5.3 E-speak

The FAR prototype's ability to actually accomplish electronic commerce comes about through its use of e-speak. E-speak is an open services software platform [http://www.e-speak.net] developed by Hewlett-Packard and designed specifically for the development, deployment, and intelligent interaction of e-services. (It is freely available at http://www.e-speak.hp.com.) An e-service is any service provided and consumed on the intranet or internet. Although e-speak has helped and influenced the way FAR is implemented, the end user does not know about this association.

At the core of e-speak is the notion of an "e-speak machine." An e-speak machine knows about services that have been made available and about services that clients are requesting, and can also talk to other e-speak machines about services requested and available. Basically, the e-speak machine is software that performs the primary e-speak functions of discovery, negotiation, mediation, and composition. It also provides the security features of e-speak.

2.5.3.1 Primary Functions of E-speak

E-services that are capable of intelligent interaction can dynamically discover and negotiate with each other, can mediate on behalf of their users, and can compose themselves into more complex services.

Discovery

Once an e-service is e-speak-enabled, the provider registers it with a host system connected to and accessible by the Internet. During registration, the provider creates a description of the e-service that consists of its specific attributes. Users looking for e-services then describe the type of service they want and e-speak discovers registered services that have the desired attributes. A vocabulary is a set of attributes and properties that defines a service.

Example: The attributes of a dynamic generation of a flower web page service could include the types of flowers, prices, type of soil, growing season, etc. Users who need to buy flowers or to get some information about some flowers could compose a request by stating the attributes that were most important to them, such as "pansy", "less than $30," and "grows in February".

Negotiation

After discovering e-service providers, e-speak negotiates between the requestor and the provider to weed out any that offer services outside the criteria of the request.

Example: Following the previous example, the user who wants to buy flowers under $30 might be matched with a list of two or three qualified providers. (In standard information portals, users generally only specify a type of service and frequently receive scores of "hits" to sift through.)

The FAR prototype does not include any features in the present implementation to illustrate the following *Mediation* and *Composition* functions of e-speak.

Mediation

Once a user and an e-service have been brought together, e-speak is able to continuously monitor service delivery and make adjustments, or "mediate," in real-time.

Example: A service provider offers streaming multimedia content over the Internet. A user logs onto the service and selects a particular training video. However, the user decides not to complete the video and logs off in the middle. Because e-speak can monitor the delivery of the service, the provider is able to bill the user only for the time the video was actually watched.

Composition

E-speak-enabled e-services can be combined into more complex, cascading e-services on-the-fly because part of the e-service can be to discover and use another e-service. In this way, e-speak services can interoperate among themselves.

Example: Suppose a customer in an e-services community has requested a type of service that is not offered by one specific e-service provider, but requires the capabilities of multiple e-services. An e-service from a provider can include a service call to combine with another e-service, within or outside of the local e-speak community, to create a compound service that fits the customer's need.

2.5.3.2 Details

An e-speak system is a federation of logical machines. Each logical machine has an active entity called the core. The core has a passive component called the Repository that holds all the metadata (metadata - definition and description of data, including vocabulary description, service description, etc.). The core acts as the mediation layer and routes messages to services. Communication between users/providers of services and the core is via messages put into mailboxes. Clients and service providers join the community through these cores. Figure 1 shows a typical interaction between clients and services in an e-speak community.

The owner of a service registers it with the core, providing the metadata. The core records this information in the Repository and assigns the metadata a repository

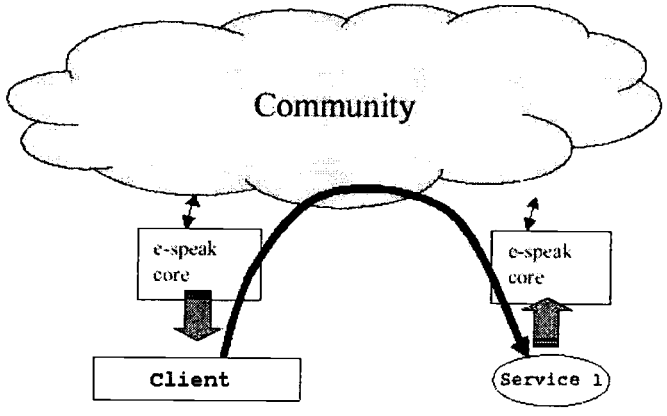

Figure 1: Interaction between clients and services (from www.e-speak.net).

2.5.3.3 Programming model

An e-speak service implements a set of interfaces defined in a service contract. An interface is a service description that enables services to be discovered. Clients search for services registered in the community, and when they are successful, they gain access to the service using a service proxy (stub), a proxy object of the remote object at the client side of the discovered service.

In addition, the service is advertised in a relevant vocabulary. A vocabulary consists of a set of associated attributes and properties. The properties associated with the attributes are the name of the attribute and the type of values to which the attribute is assigned. Figure 2 shows the relationship among services, contracts and vocabularies.

There are two interface options available with E-speak:

(1) J-ESI

(2) Web Access

J-ESI (Java E-Speak Interface) is library of Java classes that exposes all the functionality of e-speak to support development or use of e-services. It provides the interface for e-speak to environments that use programmatic environments such as Java. Using one of the classes in J-ESI, the connection between Clients and the E-speak infrastructure is established. After a connection is established, the Client receives a stub that is then used to communicate with the Server.

J-ESI acts as the Java system call interface to the E-speak core. The core is the

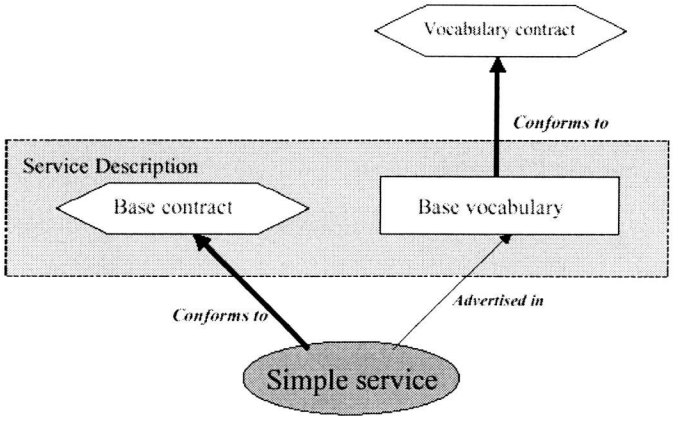

Figure 2: Relationships among services, contracts and vocabularies (from www.e-speak.net).

active entity of e-speak that acts as the mediation layer and routes messages to the services.

J-ESI supports a set of base functions that are required to get connected to the e-speak infrastructure and to create and find new user-defined Services. These functions include the following:

(1) Connection - used to connect and disconnect from the e-speak infrastructure.

(2) Vocabulary - allows the creation of new vocabularies and queries the properties of existing vocabularies.

(3) Contract - used to create and use contracts.

(4) Elements and Finders - elements are used to register services with the core, while finders are used to find services.

Web Access, based on XML, enables users to interact with the E-speak core or services through standard web browsers, by returning HTML or XML documents in HTML or XML.

We used J-ESI in our present FAR implementation.

2.5.3.4 Client-Provider communication scenario

The steps involved at the service provider side are

1. get connected to the core.
2. create a service element with a description of the service vocabulary and contract.
3. register with e-speak core.
4. start the service.

Clients find the desired services using the service finders available in the e-speak infrastructure through some interface like J-ESI or Web Access. Clients take the following steps to find and use a service.

1. give the attributes to the advertising agency to search for the service.
2. advertising agency returns the name bindings of the service.
3. on finding a match, the client gets a handle to a service stub that can be used to invoke the methods that are implemented by the service provider.

As this section shows, e-speak has a number of features, but using them is fairly involved. In fact, the very reason for FAR is to automatically and invisibly take

care of low-level details such as these, so that end users can run cottage e-commerce businesses beyond static web page services without having to hire professional programmers.

Chapter 3: Introduction to FAR[2]

3.1 Introduction

FAR is an end-user visual language to allow end users to offer e-services. It is aimed at end-user service providers, not at customers.

During its design, we evaluated FAR using the Representation Design Benchmarks [Yang et al. 1997], a design-time evaluation tool for VPLs based on Cognitive Dimensions [Green and Petre 1996]. (See [Burnett and Chekka 2000] 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 can understood, because it requires the language designer to pair the language's design features with official barriers in the form of required prerequisites. For FAR, the audience prerequisites are familiarity with browsers, spreadsheet formulas, and at least surface familiarity with database productivity tools such as Excel or Access. The detailed evaluation using Representation Design Benchmarks and Cognitive Dimensions is included in the next chapter.

To briefly overview FAR, suppose a gardener (service provider) 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 and specifying rules and/or spreadsheet-like formulas for dynamically filling in parts of the page based on an incoming query, as in Figure 3. 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 versus drag-and-drop authoring tools to create a static web page accessible via search engines are:

---

[2] The contents of this chapter are modified from a section of the paper "FAR: An End-User Language to Support Cottage E-Services" co-authored with Margaret Burnett and Rajeev Pandey.

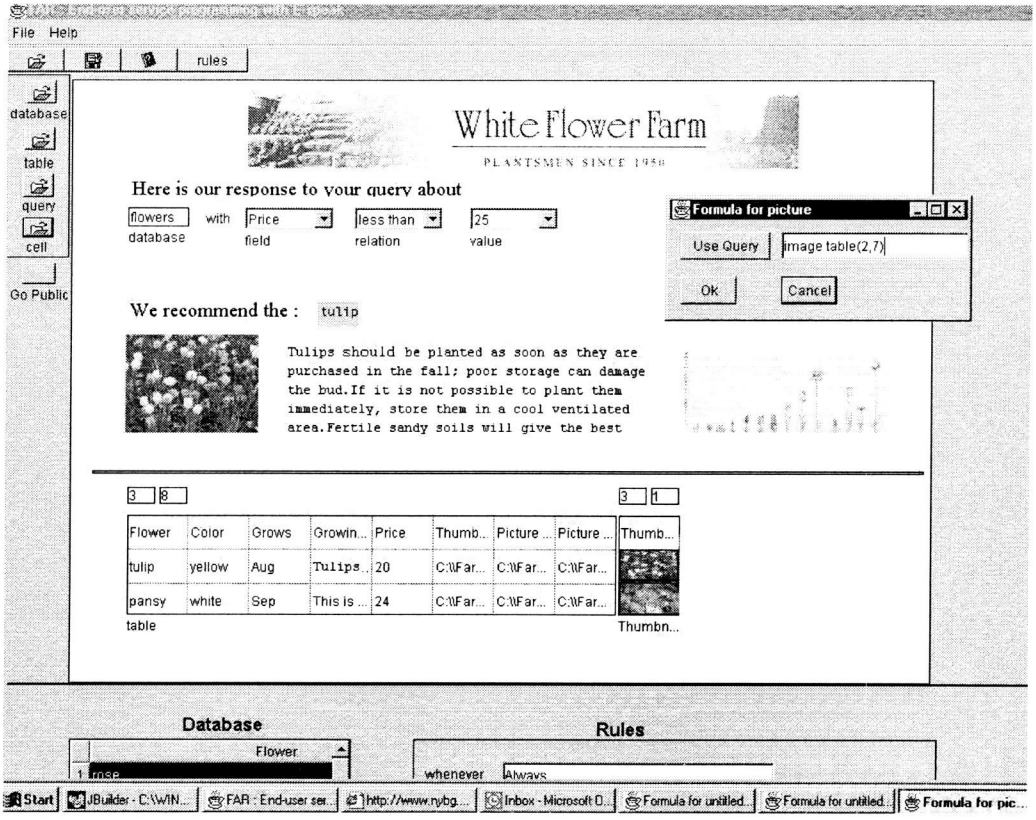

Figure 3: Snapshot of the flower advisor program in FAR as it is being created by the gardener. Everything shown in the web page area is a cell or table of cells, including the blocks of text, images, and so on. The gardener has just added the gray cell whose value is "tulip".

- Although the services 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 query always answers precisely the question the client 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.

3.2 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 as shown in Figure 3. Using drag and drop, the gardener can lay out a 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 a pop-up cell attributes menu.

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. (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 for the picture cell (the picture mid-left in Figure 3) 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 [Burnett et al. 2000; Burnett et al. 2001], Formulate [Ambler 1999; Wang and Ambler 1996], 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` operator 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`, he or she makes some of the columns 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.

3.3 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 shown in Figure 3, 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 the figure reflect the vocabulary to be used by the clients 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 clients 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.

3.4 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.

3.4.1 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 $consequent\text{-}expression_j$ (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 is of the form (*predicate, {C, consequent-expression}*), for all *C* with predicate *predicate*.

3.4.2 Using rules

When the user selects a cell, its rule is automatically displayed in the Rules section (bottom right of Figure 4). For example, in the Figure 4(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.

3.4.3 An example

As Figure 4 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 `quantity`) 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 values for `subtotal`, `tax`, and `total` will be all 0, which will look amateurish on a flower advice web page to be delivered back to a client who never intended to buy physical flowers, only to obtain flower advice.

To solve this problem, the gardener needs to change the formula for several

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

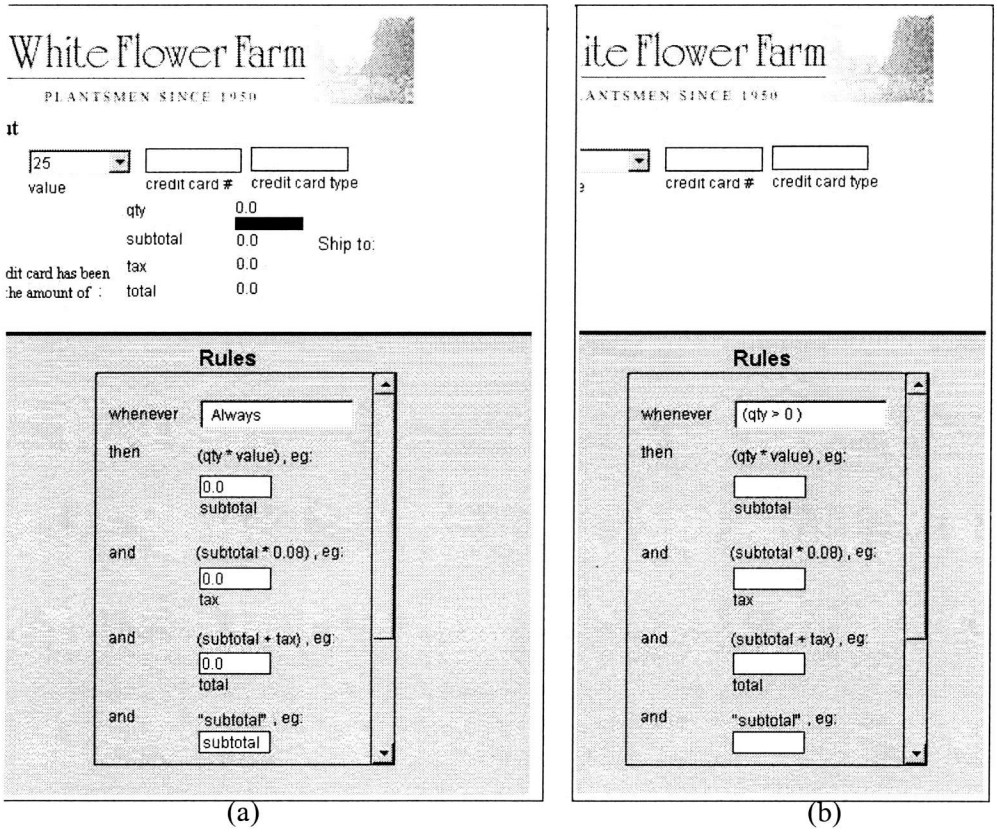

Figure 4: The flower advice example has been expanded, and the separator bar has been dragged upwards to make more room for the rules to be visible. (a) The gardener has added cells allowing a customer to purchase the recommended flowers if desired. The predicate of all these cells is initially "always". (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).

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 4(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 4(b). This also changes the formulas for all 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.

3.5 The Runtime System

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

The FAR runtime system generates a Server file to deploy the service. The steps involved in are as follows using the J-ESI class library.

(1) *Create connection to the e-speak core.*

(2) *Create the service description.*

Part of this step involves the generation of the vocabulary description for the service by the system with attributes being the query cells. These query cells are created by the end-user in the process of building the FAR program (e-service). The vocabulary description is stored in an XML format. The service description is then created using this vocabulary. Users looking for e-services describe the type of service they want and e-speak discovers services that match with the user specified attributes.

(3) *Register the service*

This registers the service with the e-speak repository.

(4) *Advertise the service*

The service is advertised with the local advertising service. This makes other cores in the community to discover and use the service.

(5) Start the service

This makes the service start servicing the client requests.

Clients 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 client 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 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 client 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 are electronically delivered to the customer in the form of an XML document. 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 the 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.

Chapter 4: Implementation Issues

This thesis introduces the FAR WYSIWYG programming language. The present implementation focused on one aspect of end-user e-services, the ability to offer their own customized services. We did not focus on the ability to consume services.

This chapter will deal with some of the implementation issues in the development of FAR. It has been developed on an NT machine in the JBuilder 3.5 Enterprise environment, which is used to develop Java applications.

The FAR project is divided into 3 parts, namely: Engine, GUI and the GUI-Engine packages.

The GUI has all the classes which are related to the graphical interface of FAR. The Engine has all the classes that do the actual implementation behind the visual interface. It is designed to minimize intermingling between the GUI and Engine. The communication between the GUI and Engine is carried over using the GUI-Engine package classes. The invariant this architecture is intended to preserve is that, if we discarded the GUI, none of the language's ability to compute, parse, etc., would be lost. Thus, the GUI package does not handle language-related decisions; it simply relays user communications back to the Engine and paints the result that eventually comes back.

4.1 Connection with the user's local database

FAR is presently implemented with the user's local database being an MS Access database. For this purpose, JDBC-ODBC Bridge is used. The JDBC-ODBC Bridge allows applications written in the Java programming language to use the JDBC (Java Database Connectivity) API with any ODBC (Open Database Connectivity) driver to access the database. We have used the ODBC driver for MS Access.

This part of the database connectivity is implemented in the FarDatabase.java and GuiDatabase.java files. When the user clicks on the "database" button in the tool palette, the system builds a small panel with the scrollbars added to it to view the database and makes the connection with the database using the JDBC API at the engine side. Figure 5 shows the sequence diagram for the database connection.

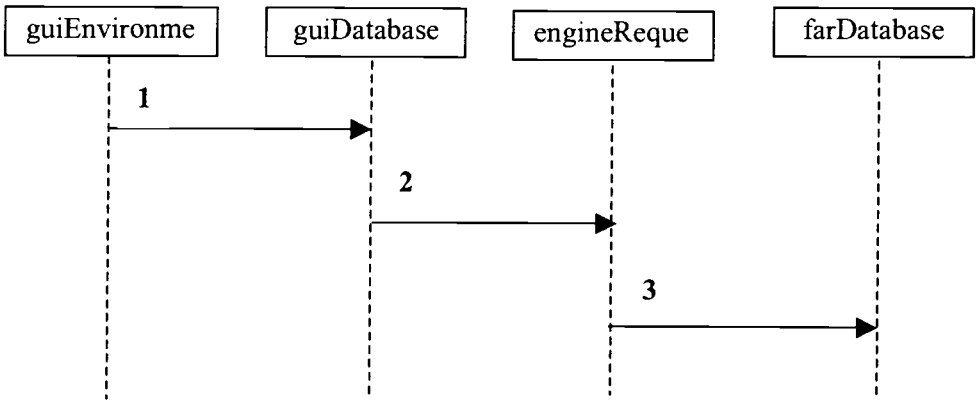

Figure 5: Sequence Diagram for the database connection.

1. call to build the panel for the database
2. call to make the database connection at the engine side
3. call to the actual method for database connection

4.2 Parsing of the FAR formulas

During the development of FAR, there were many instances where the grammar for the formulas had to be changed. It was difficult to create a parser each time the grammar was changed. To make it easier, we made use of JavaCC (Java Compiler Compiler). It is a Java parser generator that reads a grammar specification and converts it to a Java program that can recognize matches to the grammar.

The grammar specification is composed in a .jj file and JavaCC creates a group of system specific class files and a parser class. This parser class instance aParser is used to parse the FAR formulas.

The FAR formulas are in the following format.

1. (a X b) , where X can be any mathematical operator
2. IF (a Y b) THEN b , where Y is any relational operator
3. a WHOSE (b Y c) , where Y is any relational operator
4. image a , where a is the path of some image or text file

Here a, b and c can be the actual values or cell references. Figure 6 shows the sequence diagram for parsing a FAR formula from the time the user enters it using the GUI.

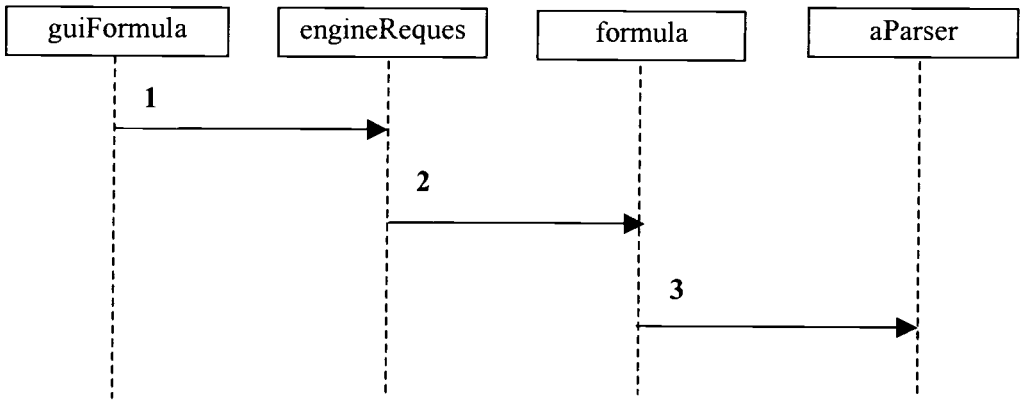

Figure 6: Sequence diagram for parsing the formula.

1. call to parse the formula which is sent as a parameter
2. call to parse the formula at the engine side
3. call to actual parse function on the parser object

4.3 Saving FAR programs and using existing FAR programs

FAR programs created by a user are stored in XML format on the user's local machine. We have created a DTD (Document Type Definition) for these XML documents, which is included in Appendix A. We used Oracle's DOM API to deal with the XML documents, which makes use of its parser for XML documents.

To open an existing FAR program stored in the XML format, the file is parsed and all the required information about the objects including position, attributes, formulas, etc. is retrieved from different tags to fill the relevant data structures. This is achieved by using DOM API. The sample XML file of a FAR program is included in Appendix B. There are many APIs available but we have used the Oracle's DOM API which makes use of its parser. This is implemented in XMLReader.java.

When the user clicks on the "save" button from the menu bar (with the little

floppy icon on it), a small filechooser window pops up through which the user can select the name for the developed program and save it on the local disk. The system reads all the data structures and stores the information in the corresponding XML tags according to the DTD defined earlier.

When the user clicks on the "open" button from the menu bar of FAR (it has a little open folder icon on it), a small filechooser window pops up from which the user can browse through the files on his or her machine and select an existing FAR program. Then the system parses the selected XML file, fills in the data structures at the engine side and displays all the objects with their formulas according to the information stored in the XML tags.

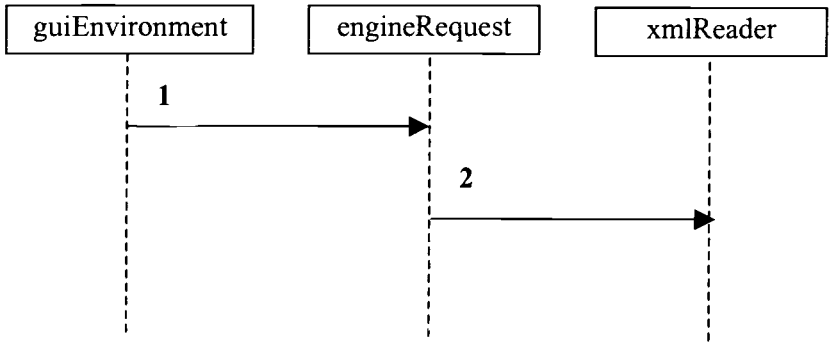

Figure 7: Sequence diagram for opening the existing FAR programs.

1. call to open the FAR program which is sent as a parameter at the engine side
2. call to actual method to open the XML file

4.4 Specifying the formula

Consider the scenario of the user specifying a formula for a cell or group of cells (table). The user clicks on the object for which the formula has to be specified, and a small formula window pops up prompting the user for the formula. The user can type a formula here or can use the "Use Query" button followed by clicking on the "Ok" button.

After the user clicks on the "Ok" button, the GUI relays the formula to the Engine, which parses the formula and then executes it followed by updating the value of the object. All the other objects are also updated to reflect the recent changes. Changed values are then communicated back to the GUI for displaying.

Figure 8 shows the sequence when the formula is specified for a cell or group of cells.

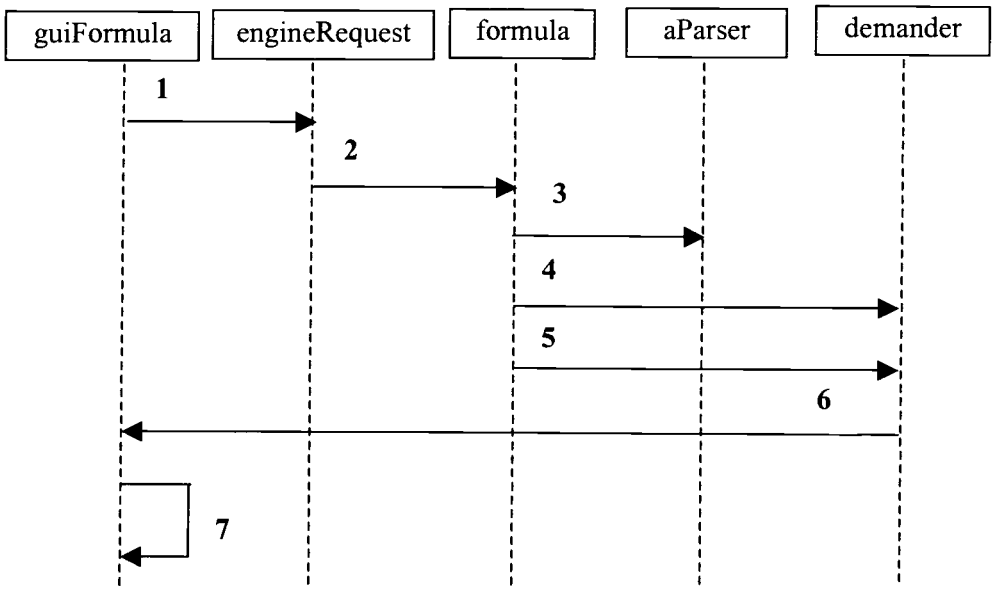

Figure 8: Sequence diagram when the formula is specified.

1. call to parse the formula which is sent as a parameter at the engine side.
2. call to parse the formula at the engine side
3. call to actual parse method on the parser object
4. call to execute the formula which is parsed
5. call to update all the other objects
6. return the values to the GUI side
7. call to display the results which are returned from the Engine

4.5 Viewing the Rules

The user of FAR has the option of shifting to the Rules paradigm at any point of the development of the service. The user can select one of the objects in the webpage worksheet (white background in Figure 3) by clicking on it once (small black border appears on top of it indicating that the object is selected) and clicking on the "rules" button at the top left corner in the same figure. All the objects having the same

predicate as the selected object then appear in the rules section (bottom right corner in same figure).

This is implemented in the GuiEnvironment.java, GuiRules.java and Rules.java files. GuiRules.java builds the panel to display the rules. It also creates the "whenever" section for the rules and calls the method at the engine side to derive the rules. Rules.java is the counterpart of GuiRules.java at the engine side to derive the rules. After the rules are derived, GuiRules.java displays the visual "then" part of the rules that were derived at the engine side.

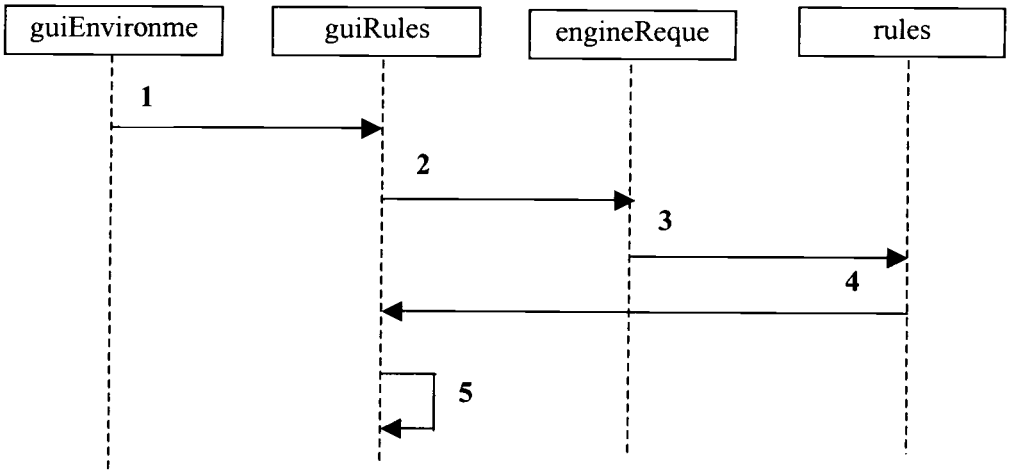

Figure 9: Sequence diagram for viewing the rules.

1. call to create the rules at the GUI side
2. call to create the rules at the engine side
3. call to actual method to create the rules
4. return the "then" part of rules to the GUI side
5. call to display the visual "then" part of rules

4.6 Editing the Rules

The user edit the rules in the rules section and the changes are updated in the formula section automatically by the system. The user can modify the rule in the rules section by editing the predicate in the textbox of the "whenever" block. When the focus is changed away from this, the GUI relays the old and new predicates to the

Engine, which updates formulas to reflect the user's modifications and then communicates back to the GUI to display the new values.

This is implemented in GuiRules.java, Rules.java and Demander.java files.

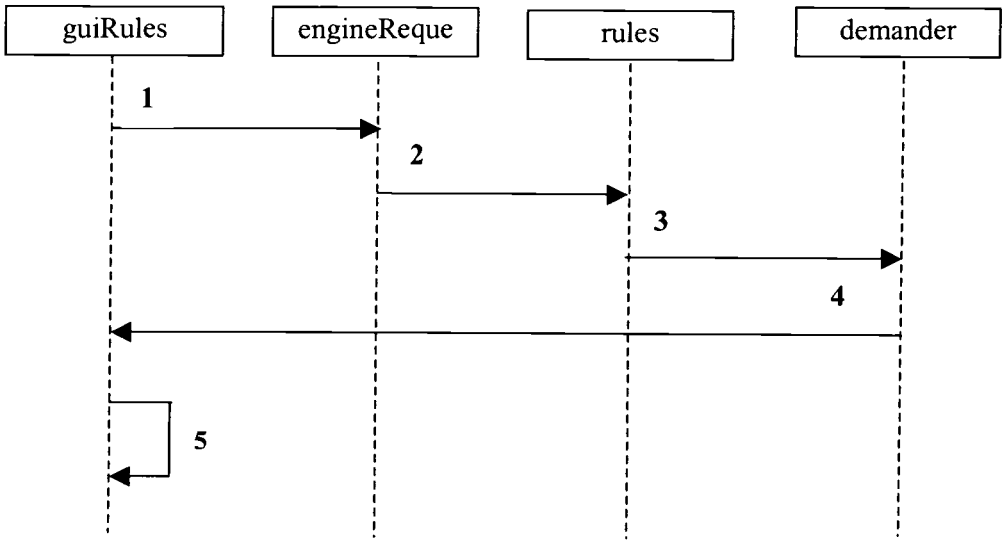

Figure 10: Sequence diagram for editing the rules.

1. call to update the rule where old and new rules are sent as parameters
2. call to update rules at the engine side
3. call to actual method to update the formulas to reflect the changes and calculate new values.
4. return the values to the GUI side
5. call to display the new values

4.7 Starting the service

The user of FAR can start the service by clicking on the "Go Public" button. The FAR system constructs the vocabulary description (an XML file) and then creates a server file (java file) with the service description using the vocabulary file to deploy the service in the e-speak infrastructure (explained in detail in the runtime section of Chapter 3 and E-speak section of Chapter 2). The FAR system registers the service with the e-speak engine, advertises with the local advertising agency after starting the e-speak core by executing the same generated java server file. Then the FAR system

starts the service and waits to service the client requests, which will eventually be delivered by the e-speak core. All communication with e-speak is taken care of in the EspeakSupporter.java file.

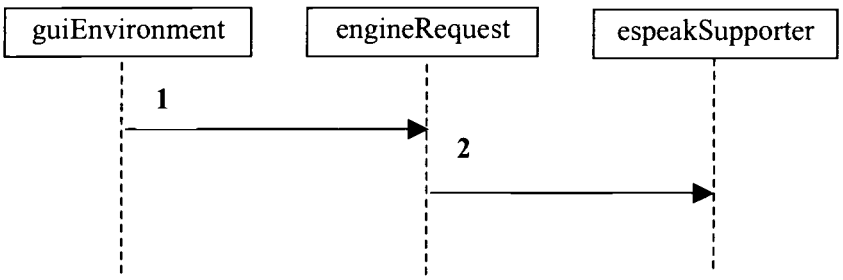

Figure 11: Sequence diagram for starting the service.

1. call to create the e-speak vocabulary and server files at the engine side
2. call to the actual method to create the e-speak vocabulary and server files

The actual design of the implementation as described above was supposed to start the e-speak engine automatically by the FAR runtime system and then the FAR generated java server file to register, advertise and start the service. But due to the inability to start the e-speak core at runtime, we had to compromise: in the current implementation, the user explicitly starts the e-speak core and then executes the java server file to start the FAR service as a temporary fix for the problem.

Chapter 5: FAR Evaluation

5.1 Representation Design Benchmarks

It is possible to bring research into cognitive issues of programming to bear upon visual programming language (VPL) design decisions by considering Green's and Petre's *Cognitive Dimensions*, a distillation of psychology of programming knowledge into a form usable by non-psychologists [Green and Petre 1996].

As a concrete application of the cognitive dimensions, *Representation Design Benchmarks* [Yang et al. 1997] were designed in an earlier collaboration between Oregon State University and Hewlett-Packard. The benchmarks are a flexible set of measurement procedures for VPL designers to use when designing new static representations for their languages. They focus on the static representation part of a VPL, and provide a designer with a yardstick for measuring how well a particular design fulfills design goals related to the static representation's usefulness to programmers. These benchmarks measure the VPL's navigable representation (S, NI) where S is the VPL's static representation and NI is the VPL's navigational instrumentation. *Navigational instrumentation* is the set of devices that take a static representation as input and map it to a subset of that static representation as output.

The purpose of representation design benchmarks is to provide a set of early (design-time) measures that VPL designers can use to measure and improve their design ideas. Any design problems found relating to eventual human usage at this early stage are considerably less expensive to both find and fix than would be the case if design problems were not found until a prototype complete enough for usability testing were ready.

The evaluation of FAR at early stages using these benchmarks follows.

5.1.1 Understandability Benchmarks

5.1.1.1 Visibility of Dependencies (D1, D2)

There is a *dependency* between P1 and P2 if changing some portion P1 of a program changes the values stored in or output reported by some other portion P2. Dependencies are the essence of common programming/maintenance questions such

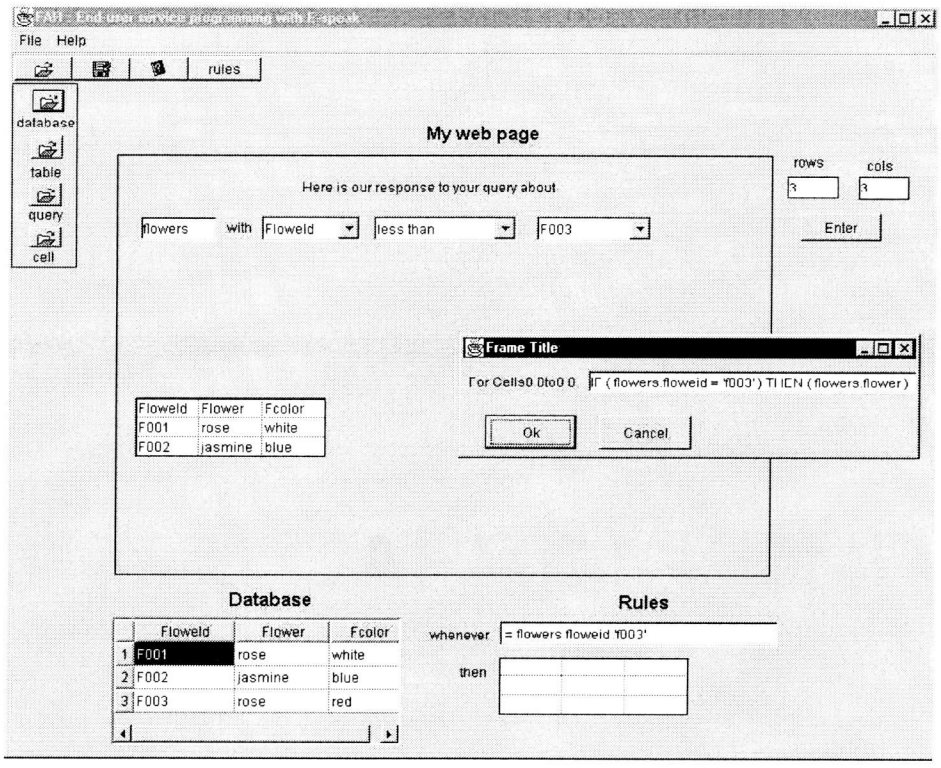

Figure 12: Snapshot of FAR from the early prototype used for the evaluation

as "What will be affected if P1 is changed?" and "What changes will affect P2?" Green and Petre noted hidden dependencies as a severe source of difficulty in understanding programs [Green and Petre 1996]. In FAR, program portions are databases, tables, queries, and cells (examples of each are in Figure 12).

D1 measures whether the dependencies are explicitly depicted in the static representation of a VPL. D1 = (Sources of dependencies explicitly depicted) / (Sources of dependencies in system). In FAR, the sources of dependencies are direct (2), transitive (2) and query (2) dependencies. In the figure, a formula is shown in the small frame for the selected cell (here the "rose" cell is selected by a single click on it). One direct dependency is shown: the cell references that are used directly in the formula. The other direction, which cells the "rose" cell affects, can be seen in the rules view. Query dependencies (the dependency of the results in the table on the query) are not explicitly shown at all. Also transitive dependencies (that the results in the table depend on the query which in turn depends on the database) are not shown. Since only two of these possibilities explicitly are shown, D1 = 2/6.

D2 is the worst-case number of steps required to navigate to the display of dependency information. It takes *n* steps to display all formulas (one at a time), where *n* is the number of cells, so D2 = n.

Design implications: D1's low score showed that the formula representation scheme needed improvement. The query dependency is visible now since the formulas use the query cell names in their formula representing the data of the query. The new D1 score is 3/6. We could reduce the D2 score by including a "Show All" button to display all the formulas corresponding to the cells. This would reduce D2 to n/2. We have not yet decided whether *n* is large enough to warrant this extra button.

5.1.1.2 Visibility of Program Structure (PS1, PS2)

Program structure is the relationships among all the modules of a program. From the programmer's standpoint, a depiction of program structure answers questions such as "What modules are there in this program?" and "How do these modules logically fit together?" Example depictions of program structure in other languages include call graphs, inheritance trees and diagrams showing the flow of data among program modules. In FAR, the main modules are the database, the query, and the sample web page.

PS1 measures the presence or absence of program structure in the static representation and takes Yes/No. In FAR, there is no explicit representation of how the above-mentioned modules of the program logically fit together, so PS1 = "No." However, since there are only three modules, we do not plan to change this.

PS2 measures the worst-case number of steps required for a programmer to navigate to the display of the program structure. Since PS1 is "No", PS2 is not applicable.

Design implications: None.

5.1.1.3 Visibility of Program Logic (L1, L2, L3)

If the fine-grained logic of a program is included in a static representation, we will say the program logic is *visible*. We will say the visibility of the program logic is *complete* if the representation includes a precise description of every computation in the program. Textual languages traditionally provide complete visibility of fine-grained program logic in the (static) source code listing, but some VPLs have no static view of this information. Without such a view, a programmer's efforts to obtain this

information through dynamic means can add considerably to the amount of work required to program in the language. For example, one study of spreadsheet users found that experienced users spent 42% of their time moving the cursor around, most of which was to inspect cell formulas [Brown and Gould 1987].

L1 measures whether the static representation provides visibility of the fine-grained program logic and takes Yes/No. The formulas of the cells in the corresponding formula frames do show how an element is computed, so L1 = "Yes." Alternatively, logic can be seen in the rule view.

L2 measures the worst-case number of steps required to display the code. As pointed out for benchmark D2 above, if the only way to show the formulas is one formula at a time, it will take *n* steps to display all the formulas (L2=n). If we add the "Show All" button, the average number of steps is n/2 (L2 = n/2).

L3 is the number of sources of misrepresentations of generality. There is a source of misrepresentation in the use of concrete values in the displayed formulas (when a formula has to be general since it has to be replaced with the incoming query from the client), such as "f003" in the formula in Figure 12. The comparison as shown is not really the formula being used by the system; behind the scenes, the formula actually refers to abstractly to the current query's value.

Thus L3 = 1, which indicates there is a problem.

Design implications: The misrepresentation problem revealed by L3 needed to be solved to avoid misleading the user as to exactly what computation is being done here. As mentioned earlier, in the present implementation, the formulas can have references to the query cells or concrete values and so this misinterpretation is removed. A formula can refer to a constant such as "f003" in its formula or to a query cell reference.

In Figure 12, the formula in the frame is "IF (flowers.flowerid = 'f003') THEN (flowers.flower)". We replaced the usage of database related IF...THEN formula with a WHOSE formula. So the above formula in the present implementation is "databasetable WHOSE (field relation value) " where the databasetable, field, relation and value are the names of the query cells having the values flowers, flowerid, = and f003 respectively. The user also has the option of typing this.

This no longer misrepresents the logic but it is hard to imagine an end user understanding that this is the formula needed. Thus a "Use Query" button has been added to automatically generate this formula from the query cells.

5.1.1.4 Display of Results with Program Logic (R1, R2)

This group of benchmarks measures whether it is possible and feasible to see a program's partial results displayed with the program source code fragment that produces each partial result.

The ability to display fine-grained results (values of each cell, etc.) at frequent intervals allows fine-grained testing while the program is being developed, which has been shown to be important in debugging [Green and Petre 1996].

R1 measures whether or not it is possible to see the partial results displayed statically with the program source code and takes Yes/No. It is possible to do this in FAR. In both formulas and rules views, results of the query and all formulas are displayed immediately with the code, so R1 = "Yes."

R2 is the worst-case number of steps required of the programmer to display results with the source code. All final and partial results are automatically and immediately displayed and updated, so R2 = 0.

Design implications: None.

5.1.1.5 Secondary Notation: Non-Semantic Devices (SN1, SN2)

Secondary notation is the collection of optional non-semantic devices that a programmer can include in a program to clarify its meaning, such as comments in traditional languages. Changing an instance of secondary notation, such as a textual comment, does not change a program's behavior. Petre argues that secondary notation is crucial to the comprehensibility of graphical notations [Petre 1995]. The use of secondary notations allows clarifications and emphases of important information such as structure and relationships. Four such devices are measured by SN1 and SN2: optional naming, layout devices with no semantic impact, textual annotations and comments, and static graphical annotations.

SN1 measures the presence of these notational devices. SN1 = SNdevices/4. In FAR, naming is optional and the tables and cells in the rules view can be grouped. It is possible to add comments and static graphic notations anywhere simply by adding cells that have the attribute "hidden", and whose formulas produce strings or graphics.

Layout devices have semantic impact in the web page view, because they affect the position of elements on the final output. However, each unique if-condition defines a rule antecedent (labeled "whenever" in the figure's rules view), and the union of all the corresponding then-conditions comprises the rule's "then." When the rules are displayed in the rules view, the user can rearrange the elements within a

"then" without semantic impact, so in this view, layout devices are not semantic. Even so, the rules view is transient, so layout devices used are not permanent parts of the program; thus we consider them about "half" present in SN1. Thus, SN1 = 3.5/4.

SN2 is the worst-case number of steps required to navigate to the secondary notations. Most of the above-mentioned devices are always visible, except for the hidden cells, and a "Show Hidden" button is planned to make those visible in 1 button click. Thus SN2 = O(1) in the worst case for most of the non-semantic devices. However, the rules view requires one or more cells to be selected first, and the worst-case cost to get all cells in that view is O(r), where *r* is the number of unique rule conditions (< n, the number of cells).

Design implications: Both SN1 and SN2 can be improved if the rules view layout is made a permanent part of the saved program.

5.1.2 Scalability Benchmarks

5.1.2.1 Abstraction Gradient (AG1, AG2)

The term *abstraction gradient* refers to the extent to which a VPL supports abstraction. Abstraction is a well-known device for scalability in programming languages, because it usually reduces the number of logic details a programmer must understand in order to understand a particular aspect of a program. It also allows a larger fraction of a program to fit on the physical screen, since replacing a collection of details by an abstract depiction almost always saves space.

AG1 measures the sources of details that can be abstracted away from a representation. In a VPL, 4 such sources of details that can be abstracted away are data details, operation details, other fine-grained portions of the program, and details of navigational instrumentation devices.

AG1=AGsources / 4. In FAR, the data details (table, etc.) can be abstracted away only to some extent by adjusting the table's size, which brings up scrollbars (score for this: 1). The operational details in formulas are abstracted by the "Hide All / Show All" button or by collapsing the formulas one at a time (1). The other portion of the program, the database, can be shrunk to the same extent as the tables (1). The navigational instrumentation devices like control panel that includes tool palette cannot be abstracted away (0). Summing up these partial scores gives AG1 = 3 / 4.

AG2 measures the worst-case number of steps required to abstract the above details away. In FAR, as mentioned above, data details of a table can be abstracted

away by grabbing a table and shrinking it (O(n) where *n* is the number of tables) and operation details in *n1* (or *n*1/2 steps using a "Hide All / Show All" feature as mentioned above) where *n1* is the number of cells.

We can combine these factors by taking a worst-case view that the number of tables is the same as the number of cells, giving AG2 = O(n).

Design implications: The lack of abstraction ability for the navigational instrumentation devices does not seem like a serious problem, since there aren't many such devices so far.

However, the navigational cost of O(n) to abstract away the other kinds of details could become excessive if the user includes a large number of tables on the web page. Thus, this is an aspect that may need further attention in the future.

5.1.2.2 Accessibility of related information (RI1, RI2)

This measures a programmer's ability to display desired items side by side. Green and Petre argued that viewing related information side by side is essential, because the absence of side-by-side viewing amounts to a psychological claim that every problem is solved independently of all other problems.

RI1 measures whether it is possible to include all related information, side by side, in a VPL's static representation. The formulas view in FAR is not a strong Yes, because rearranging cells and tables on the sample web page also rearranges the final output. However, the rules view is a Yes, since any set of cells and tables can be arranged side by side in the rules view.

RI2 measures the number of steps by the user to accomplish this. In the rules view, all related tables could be displayed and rearranged as desired, so in this sense, side-by-side placement of related information is possible. Still, the window's size is limited, and scrollbars eventually replace physical space if too many related tables are present, so RI2 in the worst case = O(n), where *n* is the number of objects the user must scroll past to view the desired part of the rules view. Further, this navigation must be done each time the program is re-loaded, since the rules view is transient.

Design implications: See SN1 and SN2.

5.1.2.3 Use of Screen real estate (SRE1, SRE2)

Screen real estate denotes the size of a physical display screen, and connotes the fact that screen space is a limited and valuable resource. This group of benchmarks provides measures of how much information a representation's design can present on a physical screen without obscuring the logic of the program.

SRE1 is the maximum number of program elements that can be laid out on the common screen size expected for the particular VPL. SRE2 measures the number of intersections and overlaps that occur while measuring SRE1. In the case of FAR, expected screen size is an ordinary PC-sized screen. Making enough space for the file menu, tool palette, database and rules view as shown in the snapshot, but expanding the window's width to fill a PC-sized screen, the total number of cells that can be placed—without formulas—in the formulas view is 228 without obscuring each other or overlapping the web page section border (allowing SRE2 to be 0). If formulas are shown below each cell (average length: about 4 value-sized columns), half the number of rows and one fourth the number of columns are possible. Thus, with formulas showing, 228/8, or approximately 28 cells, can be viewed at once if SRE2 is held to 0.

Design implications: The ability to view 28 formulas at once (including the accompanying values) seems adequate for defining a web page template, so no changes are planned.

5.1.3 Audience-Specific Benchmarks (AS1, AS2, AS3)

These benchmarks compare the representation elements with the prerequisite background expected of the VPL's particular audience. This considers the question of whether programming in a given language is familiar to the way its audience might have done other tasks without using FAR. ASn = ASyes's/ASquestions—where ASyes's = the number of "yes" answers, and ASquestions = the number of itemized questions—given questions of the general form: "Does the static representation element look like the *object / operation / spatial composition mechanism* in the intended audience's prerequisite background?" Note that this set of benchmarks does not predict whether the task is easy or hard for the user. Rather, it compares what the language designers are prepared to require as the prerequisite background to program in the language with the way programs in that language appear on the screen. For FAR, the programmers are end-users. Our prerequisite set of their background is that a FAR user has worked with some spreadsheet product such as MS Excel, has worked with some database product such as MS Access, and also has basic familiarity with the web.

AS1 compares all the *objects* in the representation with the corresponding elements in the audience's experience and background. In FAR the objects are free-standing cells, queries, tables, and databases. Except for the freestanding cells, the others are in accordance with the audience background; that is, we assume from their prerequisites that they have seen queries, tables, and databases that look similar to the ones in FAR. Hence AS1 = 3/4.

AS2 considers *operation*s. In FAR, these are the formulas and rules. Although spreadsheet formulas are generally within their background, many of these formulas use the *if* operator, which is not commonly known by most spreadsheet users. Further, many of them do not have prior experience using rules. Thus, we do not consider FAR operations to be in accordance with the audience background, and AS2 = 0.

AS3 considers *spatial composition mechanisms* for the objects and operations. The web page will show the results of a query in the traditional database style of rows and columns. Further, the web page layout depicts the intended layout of a web page. Thus, these spatial composition mechanisms are the same as those in the prerequisite background, and AS3 = 1.

Design implications: AS2's low score shows that neither the formulas nor the rules are within the audience's prior experience. Of course, the score does not actually prove that they will not find it easy, and in fact some rule-based formats have been empirically shown to be successful for end-user programmers. Still, this could be critical to the users' understanding of FAR. We are reworking the syntax of

conditional formulas, but leaving the rules unchanged at this point. If pilot studies later show that the rules are difficult for FAR users, a more graphical rule syntax, such as those used by Stagecast [Heger et al.1998] or Visual AgentSheets [Perrone and Repenning 1998], could be considered.

In the present implementation, we changed the database related formula to a different form where the database table name is used.

As mentioned in the AS2 audience-specific benchmark, the *if* operator does not fall in the prerequisites of the FAR end user. The *if* operator is not used in browsers. It is available in the spreadsheet applications like MS Excel, but is not commonly used. It does not exist in database like MS Access. In databases, keywords like *from, where,* etc. are used. So we decided to have the keyword *whose* for the database related formulas, which is closer to the above mentioned keywords.

5.2 Cognitive Dimensions

The Cognitive Dimensions framework is a broad-brush evaluation technique for visual programming environments designed to capture the cognitively-relevant aspects of structure, and shows how they can be traded off against each other [Green and Petre 1996]. We evaluated FAR using Cognitive Dimensions and explain in detail as follows.

5.2.1 Abstraction Gradient

This is covered in the Representation Design Benchmarks section (Subsection 5.1.2.1).

5.2.2 Closeness of Mapping

Programming requires mapping between a problem world and a program world. Closeness of Mapping is mapping of the entities in the user's problem domain, directly onto specific program entities, and the operations on these problem entities can be mapped directly onto the program operations. The problem domain is creating the web pages and various operations involved in the process of creating web pages like including images, tables, etc., and offering and advertising the services. We are referring to the end users with the prerequisites of having worked with spreadsheets like MS Excel, database related products like MS Access and browsers (as discussed in the Audience-Specific Benchmarks subsection, i.e. Subsection 5.1.3). The program domain is the FAR programming environment.

The formulas used in FAR are intended to reduce the gap between this problem domain and the program domain. We attempted to map the user's way to express the solutions to FAR problem entities by choosing keywords like IF...THEN and WHOSE and also simple mathematical expressions. We chose these keywords because the results of the experiments conducted in the "natural programming" project showed that end users write statements in production-rule or event-based style, beginning with words like *if* and *when* [Myers et al 2001].

In the problem domain, a list of items is often represented in the form of a

table of rows and columns in a web page. Similarly, tables are used in FAR to represent them.

Positioning and sizing objects such as tables on the web page (problem domain) of the end user, are mapped to drag-drop and direct manipulation to resize, delete and name in the FAR environment (program domain).

5.2.3 Consistency

When a person knows some of the language structure, consistency measures how much of the remaining language structure can be successfully guessed.

In FAR, we tried to maintain consistency for the end user to use remaining FAR capabilities when little of FAR is known. In FAR, if the user knows how to use (click-drag-drop on the worksheet) a *cell* tool from the tool palette, it is possible for the user to guess the usage of other tools like *table*, *query* in the tool palette which is achieved by maintaining the consistency in using the tools. The same thing is true for entering the formula for any of the objects.

It is also attempted to maintain some consistency in the syntax for the formulas. The expressions used in the formulas (regular mathematical expressions, IF...THEN and WHOSE) are written in the same way within the parenthesis. For example, the formulas "(a * b)" and "IF (a < b) THEN (a + b)" are both legal.

There is consistency maintained in the usage of formulas and rules. The expressions used in the predicates for formulas and rules are written in the same way. For example a predicate in formulas and rules is "(a < b)" as in "IF (a <b) ..." and "Whenever (a < b) ..." respectively.

We included graphics in FAR maintaining consistency with the usage of numbers as a result of formulas and rules. End users can use same kind of predicates for graphics that are applied for numbers. For example, "IF (a < b) THEN (image table(1,3))".

5.2.4 Diffuseness/Terseness

This dimension measures how much material is used to display the results. If the end user were to scan a lot of material, it would not be convenient to scan everything frequently to retrieve the required information from the given material. On the other hand if there were over-terseness, the user might find it again hard in the visibility point of view.

The FAR program fits in a single window (frame). The FAR program is not very terse, because all the information is not cramped into a small window or small screen size. The FAR program can exceed the screen size and the relevant material to some extent can be viewed simultaneously using the separator and scrollbar in the programming environment. We can say that there is no unrestrained diffuseness because the FAR program is not spread over in many windows/frames.

5.2.5 Error-proneness

Here the parts of program design and coding that lead to slips (mistakes committed by the user in the process of creating a program) are measured.

There are some slips possible in FAR, like entering the wrong cell reference name as part of the formula, which could be due to mistyping. This error-proneness could be reduced by adding a new editing feature to FAR, where the user clicks the referenced cell, and drags and drops the name wherever required instead of typing the name manually.

Another, more fundamental, source of error is the usage of parentheses in the formulas. When the user is typing in the formula, there is a possibility of entering an unbalanced formula, i.e. the parenthesis could be misplaced or left out. For some formulas, this is taken care of in FAR by including a "Use Query" button in the formula frame (could be viewed in Figure 3). This generates formulas involving the "WHOSE" keyword in the present implementation of FAR. However, this certainly does not eliminate all sources of parenthesis errors. If further improvements in this area could be made, this would be very good, because parenthesis errors are a known source of problems for end users [Myers et al 2001].

5.2.6 Hard Mental Operations

This cognitive dimension concentrates on the problematic mental operations at the notational level rather than at the semantic level.

In FAR, the source of hard mental operations could be when many levels of cell references are used. The user makes the mental operations for this case on a paper by making annotations of the intermediate values to make the operation easy but is not possible on a screen with scrolling up, down, right and left.

5.2.7 Hidden Dependencies

Hidden dependency is a relationship between two components such that one of them is dependent on the other, but that the dependency is not fully visible. We evaluated FAR for hidden dependencies in Representation Design Benchmarks section (Subsection 5.1.1.1).

5.2.8 Premature Commitment

In some programming environments, the user is forced to make a decision before the information is available. The problem arises when there are many internal dependencies or when there are constraints in the order of doing things.

In FAR, the end user does not have to decide in advance on what objects are going to be used in the process of designing the web page. The end user can select any of the tools from the tool palette and create desired objects during the development of the program and also can move the objects in the worksheet after creating the objects (although moving the objects is not supported in the prototype yet). This helps the end user to delay the decision of the concrete design: rather the user decides in the process of creating the program by viewing it.

In the case of internal dependencies, the user does not have to create the objects in a particular order according to the dependencies. After creating objects in any order, the user names the objects and enter then formulas in any order.

5.2.9 Progressive Evaluation

The FAR programming environment supports progressive evaluation of partially finished programs. The FAR program is always live in the sense that the up-to-date results of all the formulas are always displayed. When the user edits or adds a formula, the FAR engine demands the formulas to update the results and display up-to-date output of the FAR program. With this, the user can evaluate the program at every stage of the development of the program. The user can start the FAR program (offer the service in the e-speak infrastructure) at anytime during the development to check the partial output. It can also be viewed in the Rules section at any time of the development of the program.

5.2.10 Role-expressiveness

The dimension of role-expressiveness is to describe how easy it is to tell what each part of the programming environment is for.

FAR's design is intended to have good role-expressiveness. All the entities in FAR are divided into different modules to make it clear what each entity is for. These are database, Webpage section, Rules section, and tool palette. Database is the actual database selected by the end-user from the local machine. The Webpage section is to layout the web page that is delivered to the client. The Rules section shows the alternate view of the Webpage section. The tool palette is used for creating the objects during the development of program. The tool palette is divided into more fine parts like database, cell, table and query buttons.

But the explicit representation of how these different modules logically fit together is not shown.

5.2.11 Secondary Notation and Escape from Formalism

Secondary notation allows the user to add information to be carried by other means than the formal syntax such as indenting, commenting, naming conventions and grouping of related statements.

One place of usage of secondary notations in FAR is comments. This is possible by using additional cells entering the comments in it with the appropriate attributes set (like hide border, hide name, etc. of the cell). Grouping of objects related through the same predicate is not necessary by the user, because it can be seen by selecting any object in the Webpage section and viewing all the related objects having the same predicate in the Rules section. However, objects can be related in other ways like placing all the cells related to billing (`qty`, `subtotal`, `tax`, `total`, etc.) together. But this will change the output, so is not the secondary notation. The user names the objects created (naming convention is one of the secondary notation tools).

5.2.12 Viscosity

Viscosity is resistance to local change. It measures how much work the user has to put in, to effect a small change. Studies of programming showed that changes and revisions infest the whole course of programming activity, from specifying to

designing to coding [Green and Petre 1996].

Considering a situation in the FAR environment, where the common predicate in the formula of many objects has to be changed in the Webpage section. The traditional way would be to select each of these objects and edit the formula. Here the viscosity is large. In an attempt to reduce this viscosity, the rules paradigm is used. The above mentioned problem is solved by grouping all these objects having same predicate in the Rules section by selecting one of them in the Webpage section and then changing the selected predicate with a single edit. The formulas of all the selected objects in the Webpage section (can be viewed in Figure 4) are updated.

One more situation where the viscosity is reduced is when laying out the web page in the Webpage section. To change the existing design laid out in the Webpage section, the end user can select the desired objects and move them to place them in the new desired position or delete the selected object that is not desired.

5.2.13 Visibility and Juxtaposability

Visibility denotes whether the user can readily make the required material visible or readily access it in order to make it visible or readily identify it to access. It measures the number of steps needed to make a given item visible. Juxtaposability is the ability to see any two portions of the program on screen side-by-side at the same time.

The user can make the formulas of the objects visible with a single click on the corresponding object. As mentioned in the previous section on Representation Design Benchmarks, "Show All" button is used to make all the formulas visible with a single click. (The latter is not implemented in the present FAR prototype.)

With the help of a separator and the scrollbars in the FAR environment, juxtaposability is achieved in some cases. Suppose the worksheet in the Webpage section is too large to fit in a single screen and the end user wants to view some objects in the worksheet and the corresponding rules in the Rules section. This is achieved by scrolling worksheet as desired in the Webpage section till the required objects are visible and the separator is used to move the Rules section up to view the required part of the Webpage section and the Rules section side-by-side.

The same would not work if the user wants to view some objects at the top and some objects at the end of a large worksheet, side-by-side.

5.2.14 Conclusion

As a result of applying the Cognitive Dimensions to FAR, we discovered some issues that could be improved. These include:

1. Error-proneness issue of the end user while entering the formula for objects.

2. Hard mental operations when too many levels of cell references are involved. This could be reduced by including the dependency arrows as described in the Representation Design Benchmarks section.

3. Role expressiveness could be improved by having some kind of explicit representation of how different modules logically fit together.

4. Juxtaposability can be improved to some extent if a copy of the FAR program window can be created temporarily so that different parts of the program on each of the windows can be viewed side-by-side at a time.

6. Current Status and Future Work

Our research prototype of FAR implements all of the features described in this paper except the ability to make some cells in a table 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 [Yang et. al. 1997] and cognitive dimensions [Green and Petre 1996] to help guide the design of this language [Burnett and Chekka 2000], there have been no experiments involving human users to point out mismatches with the intended audience.

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 client-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 client-side end-user language, and we are considering ways to proceed in this direction.

Other issues that we have not explored include database updating, such as log transactions, and credit card authorization and charging. We believe the latter could be accomplished by automatically delegating these tasks, via the composition ability of e-speak, to e-services available from other providers.

Chapter 7: 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.

Bibliography

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

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

[Bouman 1994] P. Bouwman, H. Bruin and F. Bos, Proceedings of IEEE Symposium on Visual Languages, St. Louis, MO, October 4-7, 1994, 40-47.

[Brown and Gould 1987] P.Brown and J.Gould, Experimental Study of People Creating Spreadsheets, ACM Transactions on Office Information Systems 5, 258-72, 1987.

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

[Burnett and Chekka 2000] 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.

[Burnett and Gottfried 98] 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.

[Burnett et al. 2000] 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.

[Burnett et al. 2001] 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, (to appear).

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

[Cruz 1994] I. Cruz, Proceedings of IEEE Symposium on Visual Languages, St. Louis, MO, October 4-7, 1994, 224-231.

[Cypher and Smith 1995] A. Cypher and D. Smith, KidSim: End User Programming of Simulations, ACM Conference on Human Factors in Computing Systems, Denver, CO, May 7-11, 1995, 27-34.

[Cypher 1993] A.Cypher, Watch What I Do Programming by Demonstration, The MIT Express, 1993.

[Darnell 1999] R. Darnell, HTML 4 Unleashed, Techmedia, 1999

[DiNucci 1997] 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.

[Green and Petre 1996] 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.

[Heger et al. 1998] 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.

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

[Ioannidou and Repenning 1999] A. Ioannidou and A. Repenning, End-user programmable simulations, Dr. Dobb's Journal, August 1999, 40-48.

[Levoy 1994] M. Levoy, Spreadsheet for Images, ACM Siggraph, Computer Graphics 28(4), 1994, 139-146.

[Modugno and Corbett 1997] F. Modugno and A. Corbett, Graphical representation of programs in a demonstrational visual shell—an empirical evaluation, ACM Transactions on Computer-Human Interaction 4(3), September 1997, 276-308.

[Modugno and Myers 1994] F. Modugno and B. Myers, Proceedings of IEEE Symposium on Visual Languages, St. Louis, MO, October 4-7, 1994, 304-311.

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

[Myers 1991] B. Myers, Graphical Techniques in a Spreadsheet for Specifying User Interfaces, ACM Conference on Human Factors in Computing Systems, New Orleans, LA, April 28 - May 2, 1991, 243-249.

[Myers et al. 1990] B. Myers, D. Guise, R. Dannenberg, B. Vander Zanden, D. Kosbie, E. Pervin, A. Mickish, and P. Marchal, Garnet: Comprehensive Support for Graphical, Highly Interactive User Interfaces, Computer 23(11), November 1990, 71-85.

[Myers et al. 2001] J. Pane, C. Ratanamahatana and B. Myers, Studying the language and structure in non-programmers' solutions to programming problems, International Journal of Human-Computer Studies 54(2), February 2001, 237-264.

[Perrone and Repenning 1998] 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.

[Petre 1995] M. Petre, Why Looking Isn't Always Seeing: Readership Skills and Graphical Programming, Communications of the ACM 38(6), June 1995, 33-44.

[Pfeiffer 1998] 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.

[Pfeiffer et al. 2000] J. Pfeiffer, R. Vinyard, and B. Margolis, A Common Framework for Input, Processing, and Output in a Rule-Based Visual Language, 2000 IEEE Symposium on Visual Languages, Seattle, Washington, September 10-13, 2000, 217-224.

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

[Repenning 1995] A. Repenning, Bending the Rules: Steps toward Semantically Enriched Graphical Rewrite Rules, Proceedings of IEEE Symposium on Visual Languages, Darmstadt, Germany, September 5-9, 1995, 226-233.

[Repenning and Ambach 1996] A. Repenning and J. Ambach, Tactile Programming: A Unified Manipulation Paradigm Supporting Program Comprehension, Composition and Sharing, 1996 IEEE Symposium on Visual Languages, Boulder, Colorado, September 3-6, 1996, 102-109.

[Sassin 1994] Sassin, M, Proceedings of IEEE Symposium on Visual Languages, St. Louis, MO, October 4-7, 1994, 153-160.

[Shiffer 1994] 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.

[Smedley et al. 1996] 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, Gubbio, Italy, May 27-29, 1996, 148-155.

[Smith et al 1994] D.C. Smith, A.Cypher and J.Spohrer, KidSim: programming agents without a programming language, Communications of the ACM 37, 55-67.

[Tanimoto and Runyan 1986] S. Tanimoto, and M.Runyan, AFIPS Conference Proceedings, Spring Joint Computer Conference, 1963, 2-19.

[Viehstaedt and Ambler 1992] G. Viehstaedt and A. Ambler, Visual representation and manipulation of matrices, Journal of Visual Languages and Computing 3(3), September 1992, 273-298.

[Wang and Ambler 1996] G. Wang and A. Ambler, Solving display-based problems, IEEE Symposium on Visual Languages, Boulder, Colorado, September 1996, 122-129.

[Yang et al. 1997] 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.

Appendices

Appendix A - DTD (Document Type Definition) for FAR program

The following is the DTD used when saving a FAR program internally and it is used when opening an existing program using the DOM API.

```
<!DOCTYPE environment [
<!ELEMENT name (#PCDATA)>
<!ELEMENT database (#PCDATA)>
<!ELEMENT tableName (#PCDATA)>
<!ELEMENT field (#PCDATA)>
<!ELEMENT relation (#PCDATA)>
<!ELEMENT data (#PCDATA)>
<!ELEMENT query (tableName, field, relation,data)>
<!ELEMENT rows (#PCDATA)>
<!ELEMENT cols (#PCDATA)>
<!ELEMENT rowPos (#PCDATA)>
<!ELEMENT colPos (#PCDATA)>
<!ELEMENT Lx (#PCDATA)>
<!ELEMENT Ly (#PCDATA)>
<!ELEMENT Rx (#PCDATA)>
<!ELEMENT Ry (#PCDATA)>
<!ELEMENT op (#PCDATA)>
<!ELEMENT formula (#PCDATA)>
<!ELEMENT internalformula (op*)>
<!ELEMENT region (Lx,Ly,Rx,Ry,formula,internalformula)>
<!ELEMENT environment (name, database, query, table*)>
<!ELEMENT table (name, rows, cols, rowPos, colPos, region+)>
]>
```

Appendix B - Saved FAR program

Consider the following sample FAR program that has a query block and a table to display the results of the query. Figure 13 shows the snapshot of this sample program.

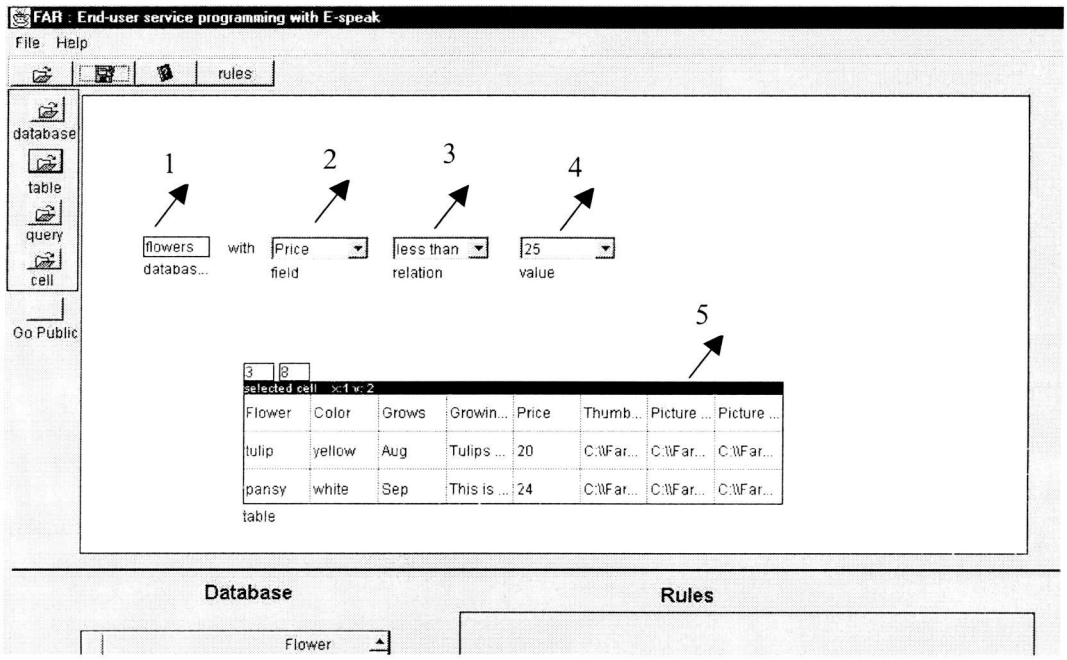

Figure 13: A sample FAR program snippet

This program is stored in an XML format using the DTD mentioned in Appendix A. The XML file of the FAR program in Figure 13 is as follows.

```
<?xml version = "1.0"?>
<!DOCTYPE environment [
<!ELEMENT name (#PCDATA)>
<!ELEMENT database (#PCDATA)>
<!ELEMENT tableName (#PCDATA)>
<!ELEMENT field (#PCDATA)>
<!ELEMENT relation (#PCDATA)>
<!ELEMENT data (#PCDATA)>
```

```
<!ELEMENT query (tableName, field, relation,data)>
<!ELEMENT rows (#PCDATA)>
<!ELEMENT cols (#PCDATA)>
<!ELEMENT rowPos (#PCDATA)>
<!ELEMENT colPos (#PCDATA)>
<!ELEMENT Lx (#PCDATA)>
<!ELEMENT Ly (#PCDATA)>
<!ELEMENT Rx (#PCDATA)>
<!ELEMENT Ry (#PCDATA)>
<!ELEMENT op (#PCDATA)>
<!ELEMENT formula (#PCDATA)>
<!ELEMENT internalformula (op*)>
<!ELEMENT region (Lx,Ly,Rx,Ry,formula,internalformula)>
<!ELEMENT environment (name, database, query,table*)>
<!ELEMENT table (name,rows,cols,rowPos,colPos,region+)>
]>
<environment>
  <name>MYflowersWeb</name>
  <database>Z:\Pro_espeak\db1.mdb</database>
  <query>
    <tableName>flowers</tableName>
    <field>Price</field>
    <relation>less than</relation>
    <data>25</data>
  </query>
  <table>
    <name>databasetable</name>
    <rows>1</rows>
    <cols>1</cols>
    <rowPos>97</rowPos>
    <colPos>-11</colPos>
    <region>
         <Lx>0</Lx>
         <Ly>0</Ly>
         <Rx>0</Rx>
         <Ry>0</Ry>
         <formula><op>"flowers"</op></formula>
         <internalformula>
            <op>"flowers"</op>
         </internalformula>
      </region>
```

1

```
</table>
<table>
  <name>field</name>
  <rows>1</rows>
  <cols>1</cols>
  <rowPos>0</rowPos>
  <colPos>0</colPos>
  <region>
       <Lx>0</Lx>
       <Ly>0</Ly>
       <Rx>0</Rx>
       <Ry>0</Ry>
       <formula><op>"price"</op></formula>
       <internalformula>
          <op>"price"</op>
       </internalformula>
    </region>
</table>
<table>
  <name>relation</name>
  <rows>1</rows>
  <cols>1</cols>
  <rowPos>0</rowPos>
  <colPos>0</colPos>
  <region>
       <Lx>0</Lx>
       <Ly>0</Ly>
       <Rx>0</Rx>
       <Ry>0</Ry>
       <formula><op>"less than"</op></formula>
       <internalformula>
          <op>"less than"</op>
       </internalformula>
    </region>
</table>
<table>
  <name>value</name>
  <rows>1</rows>
  <cols>1</cols>
  <rowPos>0</rowPos>
  <colPos>0</colPos>
```

2

3

```
            <region>
                 <Lx>0</Lx>
                 <Ly>0</Ly>
                 <Rx>0</Rx>
                 <Ry>0</Ry>
                 <formula><op>"25"</op></formula>
                 <internalformula>
                    <op>"25"</op>
                 </internalformula>
              </region>
          </table>
          <table>
            <name>table</name>
            <rows>3</rows>
            <cols>8</cols>
            <rowPos>180</rowPos>
            <colPos>117</colPos>
              <region>
                <Lx>0</Lx>
                <Ly>0</Ly>
                <Rx>2</Rx>
                <Ry>7</Ry>
                <formula>flowers WHOSE ( price less than 25
)</formula>
                <internalformula>
                    <op>WHOSE</op>
                    <op>flowers</op>
                    <op>lessthan</op>
                    <op>price</op>
                    <op>25</op>
                 </internalformula>
             </region>
          </table>
        </environment>
```

4

5