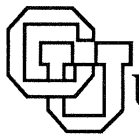
Iterative Design of a Voice Dialog Design Environment

Tamara Sumner, Susan Davies, Andreas C. Lemke and Peter G. Polson

CU-CS-546-91 September 1991



University of Colorado at Boulder
DEPARTMENT OF COMPUTER SCIENCE

Iterative Design

of a

Voice Dialog Design Environment

Tamara Sumner, Susan Davies, Andreas C. Lemke
and Peter G. Polson

CU-CS-546-91 September 1991

Department of Computer Science
University of Colorado at Boulder
Campus Box 430
Boulder, Colorado 80309-0430 USA

(303) 492-1218
email: ralex@boulder.colorado.edu

ITERATIVE DESIGN OF A VOICE DIALOG DESIGN ENVIRONMENT

Tamara Sumner, Susan Davies, Andreas C. Lemke, and Peter G. Polson*

Institute of Cognitive Science
University of Colorado
Boulder, CO 80309-0430
sumner@cs.colorado.edu
303-492-8136

*GMD-IPSI
Darmstadt, Germany

ABSTRACT

Iterative prototyping and participatory design methods were successfully applied to the development of a voice dialog design environment. The project demonstrated that choosing the right high-level substrate to build on and having the right participatory context can speed up and simplify the iterative process. Rather than choosing a general purpose tool, a task analysis combined with previous research were used to select a substrate that was well-matched to the needs of the project. Unique aspects of the substrate are highlighted and specific factors contributing to the project's success are discussed.

KEYWORDS: Participatory design, iterative design, voice interfaces, visual programming, construction kits, design simulation, design, design environments.

INTRODUCTION

In this project, we successfully applied iterative prototyping and participatory design to the design of a complex system — a voice dialog design environment. The success case for this project is really a story about how we got "this far this fast." In less than four months, a mixed team of professional voice dialog designers and academic researchers from computer science and cognitive psychology were able to design and build a substantial core design environment. There were several unique aspects of our approach that contributed to the project's success. These include selection of the right high-level system substrate on which to build our design environment and creation of a participatory context conducive to iterative design methods.

In early 1991, we began a research project to explore issues surrounding the design of computer systems to facilitate computer usage by non-programmers. This project is part of a collaborative research effort between the University of Colorado (CU) and US West's Advanced Technologies Division. A key aspect of this early phase was the identification of projects within US West which would provide a real-world context within which we could explore theoretical issues surrounding system design. Two voice dialog design groups within US West got together and presented a compelling case as to why the voice dialog domain would be an excellent framework for pursuing our research. One group specialized in the design of large-scale, innovative voice dialog applications. The other group, composed of voice dialog account executives, specialized in small-scale customized voice information systems.

Outline of Paper

In this paper we begin by describing the voice dialog domain and some design representations used by expert application designers. Next, we discuss how a preliminary task analysis and our previous research experience guided the selection of a high-level substrate to base our system on. The core of our paper is a discussion of issues that emerged during our iterative design process and how we addressed these issues in our design environment. We conclude with some lessons to build on and directions for future research. It should be noted that design and development of the voice dialog design environment is still in progress. We do not claim to have the final solution — more implementation and testing is needed before we can make such a claim.

THE VOICE DIALOG DOMAIN

Voice dialog applications are a relatively new design domain. Typical applications include voice mail systems, voice information systems, and touch-tone telephones as interfaces to hardware. Historically, voice dialog applications have been small in scale, with most applications offering only a handful of features; e.g. providing three options to hear a selection of recorded information. However, in the last few years, voice dialog applications have mushroomed in size. It is not unusual to have voice mail systems with 50 page instruction manuals, hundreds of features, and a two year development cycle. Industry deregulation combined with advances in hardware have triggered a rapid spread of voice dialog technology into new application areas. Thus, key challenges facing designers are large increases in complexity and rapid innovation within the application domain.

Design Representations

No standard design representations are used by most voice dialog designers. A common representation is the traditional written specification. Some designers, in an attempt to deal with increasing complexity, have moved to graphical representations similar to flow charts. Both voice dialog groups participating in this project were using graphical representations. While these representations were similar on the surface, there were many important differences. We chose to focus on the approach being used by the large-scale application design group.

The large-scale application designer's primary design representation is a flow chart-like "structure chart". As illustrated in Figure 1, rectangles are used to represent voice menus, messages, prompts, and system actions. The example shown begins with a menu to create a mail list for a hypothetical voice mail application. The menu contains options for creating, editing, and deleting mailing lists. Diamonds are used to represent decision junctions. These states are linked by arrows. Arrows indicate paths that are taken once an action has been executed. Arrows and decision junctions control the pattern of flow throughout the design.

The large-scale application designer also used two additional design representations to augment the structure chart: a state description table and a phrasal database. The state description table is stored as records in a database — one record per structure chart rectangle (state). Each state's record contains its inputs, outputs, prompts or messages, and atomic phrase identifiers. To conserve storage space, atomic phrase elements are recorded and recombined on-the-fly into the application's prompts and message. A phrasal database stores these atomic phrase elements and maps each phrase to an identifier.

Design Tools

Currently many designers use generic applications such as MacDraw and general-purpose databases as the primary design tools. The graphic structure chart is constructed and maintained in MacDraw. The state table representation and the phrasal database are constructed and maintained in a database. Much of the large-scale application designer's time is spent maintaining consistency between these multiple design representations.

The Medium Gap

The design process of voice dialog applications is further complicated by the "medium gap" between the visually-based design representations and the audio-based end product. Halstead-Nussloch [5] identifies many important differences between screen-based and phone-based interfaces. Screen-based interfaces display information on the screen for a duration that is under user control. In phone-based interfaces, information is presented auditorally as spoken messages and signals and must therefore be processed serially with no memory aids whatsoever. Short term memory limitations are an important design consideration. It is difficult for the designer working with a visually-oriented, long duration, design representation to mentally bridge this medium gap and envision the auditory, serial, short-term memory-constrained end product.

The Difficulty of Voice Dialog Design

The following aspects combine to make voice dialog application design difficult:

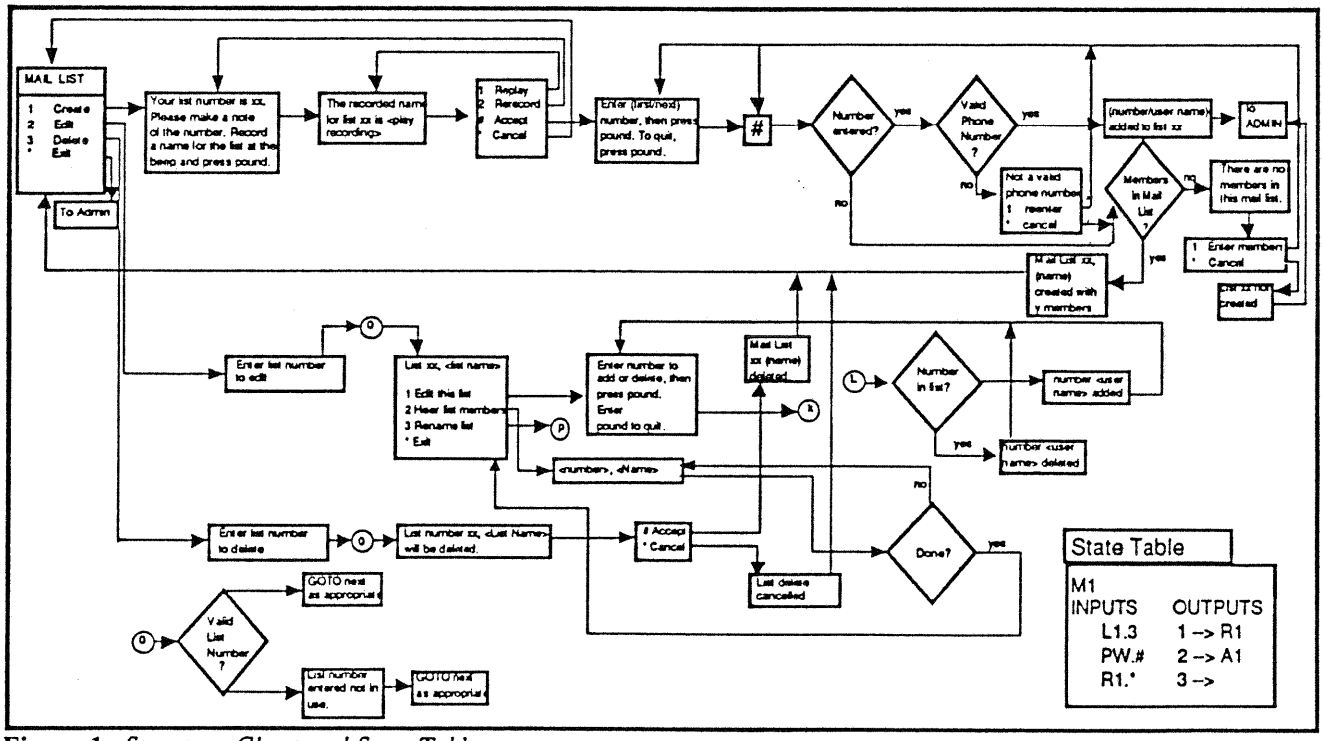


Figure 1. *Structure Chart and State Table*

- Voice dialog design is a relatively new domain with no standard design representations. Rapid innovation within the domain forces design representations to continually evolve.
- Designs are large and complex, containing many highly interconnected design elements.
- The domain suffers from a "medium gap" between the visually-oriented design representations and the auditory end-product. Current tools do not help the designer to bridge this gap.

THE VOICE DIALOG DESIGN ENVIRONMENT

The voice dialog design environment, as it stands today, is a sophisticated construction-kit approach that integrates functionality previously distributed between MacDraw and database applications. It provides an on-screen gallery of voice dialog design units, such as menus and prompts, and a work area for design construction and simulation. At any time, the behavior of the design can be simulated. Design simulation consists of a visual trace of the execution path combined with an audio presentation of all prompts and messages encountered. Thus, simulation helps the designer to bridge the medium gap between the visual representation and the audio artifact. The design of a voice mail system constructed in this environment is illustrated in Figure 2. Further details on specific features will unfold as our iterative process is described.

OUR DESIGN PROCESS

Previous sections have described the voice dialog domain and have introduced our voice dialog design environment. This section describes how we made the transition from current tools and practices to our resulting system. First, we discuss how current practices, future use visions, and our past research guided our selection of a substrate system to build on. Second, we discuss how the initial system was based on the structure chart representation. Finally, we describe how our selected substrate enhanced the iterative process by allowing us to quickly implement and evaluate new ideas.

Determining The Substrate

Our first step was a preliminary task analysis that would enable us to choose the appropriate substrate on which to build. Selection of this substrate proved to be a critical component of our early success. This section discusses how current issues in voice dialog design and our previous research in the area of design environments constrained our choices. Thus, instead of selecting a general purpose prototyping system, we selected a substrate closely aligned with our purposes and one that could accommodate rapid design iterations. Often, new distinctions could be designed, implemented, and tested within a week.

The Construction Problem. The voice dialog designers made it clear during interviews that they suffered from a construction problem:

- Current design tools had no domain semantics. Every aspect of the design had to be constructed from scratch out of rectangles, arcs, and text.
- It was difficult to modify and reuse parts of designs.
- No single tool met the demands of the domain. Different applications were used to construct and maintain each of the different design representations. Designers spend much effort maintaining consistency between these representations.

Our previous work in design environments [3] advocated a construction kit approach. A construction kit provides a gallery of domain-specific building blocks (design units) and a work area for constructing the design via direct manipulation. Such an environment has directly built into it operations and abstractions central to the given application domain. Specifically, we envisioned providing voice dialog abstractions such as prompt and menu design units and voice dialog operations such as establishing a flow direction between design units.

The Need For Simulation. The voice dialog designers also made it clear that an important feature of the future environment was design simulation. First, they envisioned simulation as a useful design debugging tool. Debugging current representations was difficult due to the medium gap between the paper design representation and the audio artifact. While some problems could be uncovered by tedious examination of the paper design representations, many problems did not appear until the prototype was available. Specifically, many errors resulted when recombining atomic phrases to generate prompts and messages. Proper inflection is essential for making a message intelligible. A phrase recorded in a rising inflection will be inappropriate when reused in a flatly inflected statement.

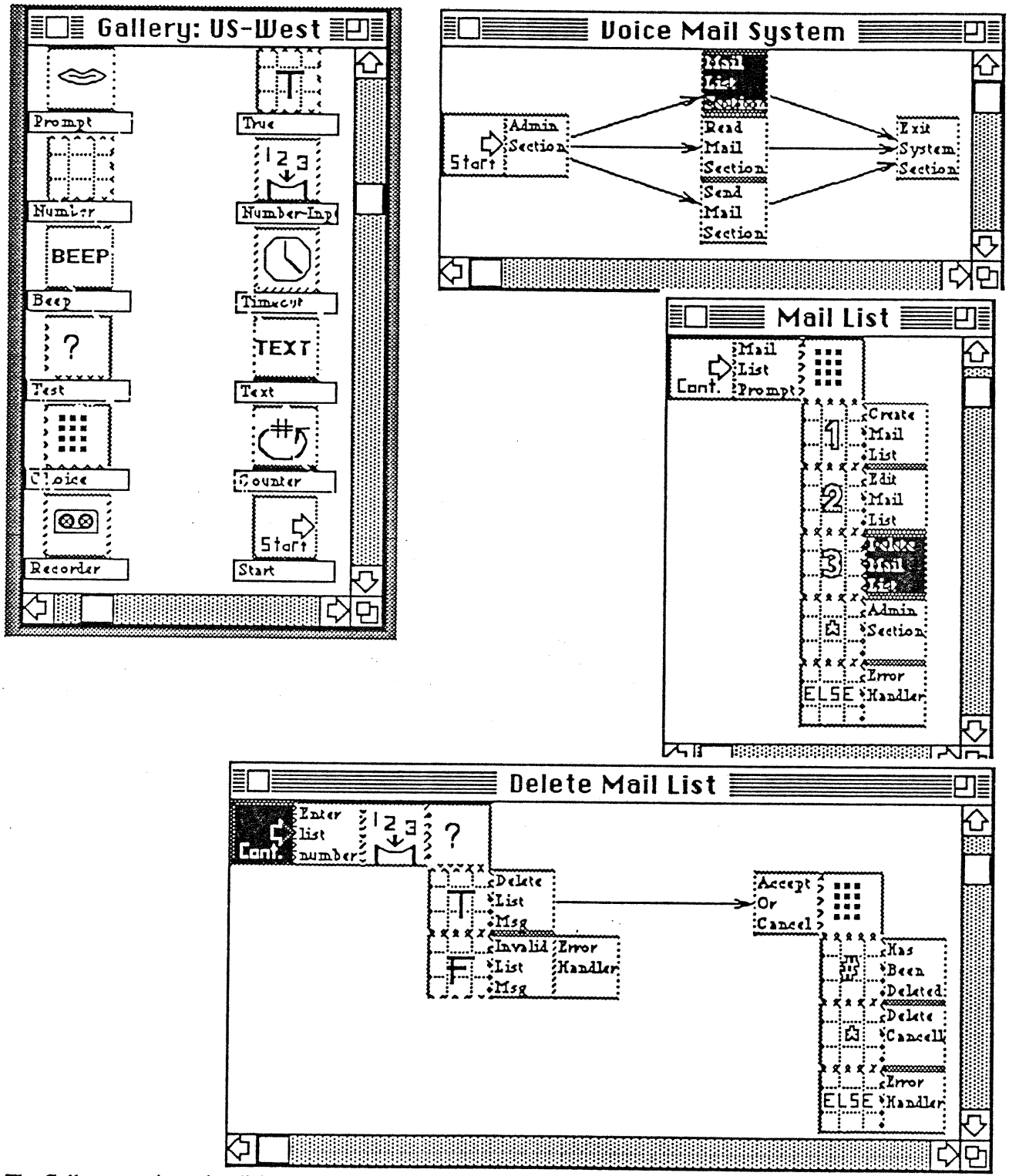
Second, simulation would allow designers to adopt an efficient iterative design methodology. Currently, implementing a prototype is an expensive and time-consuming process. While the product team is willing to build as many prototypes as necessary to get the design "right," oftentimes this process adversely affects schedules and budgets. By providing direct simulation of the design representation, the environment eliminates the need for a separate, time-consuming prototyping phase. The representation is the prototype — at all phases of the design, a working prototype is available at no extra cost.

Thirdly, simulation would be an important tool for communicating the design to the customer by allowing them to directly experience future use [1,2].

Agentsheets : The Right Substrate

We examined a variety of possible substrates and chose Agentsheets, an environment for building visual programming systems [6].

The Agentsheets system was designed to be a high-level substrate for building visual programming systems. It has



The Gallery contains voice dialog design units used in design construction. The remaining three windows are all design work areas. The Voice Mail System work area captures high level relationships between major application components. Double-clicking on the Mail List section design unit causes its constituents to be displayed in a seperate work area (the Mail List window). Within the Mail List window, double-clicking on the Delete Mail list design unit displays its constituents in the Delete Mail List work area. The Mail List work area and its Create, Edit, and Delete nested work areas together describe the same level of functionality depicted in the structure chart shown in Figure 1. Simulation of a work area (and its nested work areas) is initiated by double-clicking on any start or continue design unit.

Figure 2. *The Voice Dialog Design Environment.*

an extensible and well-documented core of object-oriented classes to build on. Additionally, there was a large number of existing applications from widely varying domains that had already been built using the system, e.g, a river basin modelling system, a children's storybook tool, and a front-end to a power station's expert system [6]. Besides proving the utility of the system, these applications provided invaluable implementation examples from which to learn.

Agentsheets supported design by construction. The Agentsheets system consists of "agents" and a work area, called an agentsheet, for programming with agents. An agent is simply a graphical depiction (bitmap), an internal state, and an associated behavioral component. In our environment, each design unit is an agent. Figure 3 illustrates how the prompt design unit is an agent. Programming or construction with agents is achieved by arranging agents in the work area.

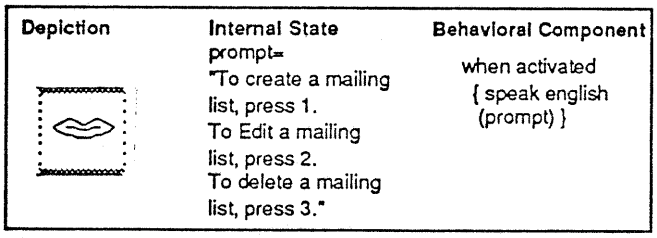


Figure 3. *A prompt design unit. Each design unit is an agent and has a depiction, an internal state, and a behavioral component.*

The Agentsheets system is a visual programming environment. In our system, constructing the design representation is a form of visual programming. Design simulation is simply a matter of executing this visual program. By its very nature, Agentsheets supported our simulation requirements.

Finally, Agentsheets ran on the Apple Macintosh, which is the same platform as the designer's current tools. Prototypes are most effective when users are in control of their use for extended periods of time within the work context [1]. We felt that leaving working versions of the prototype with the voice dialog designers throughout the project was essential to our iterative design process. This goal would only be achievable if our prototype was on a platform that was readily available to the voice dialog designers.

Determining the Initial Design Environment

We needed to determine a functional set of design units and what initial design representations the environment's work area should support. We began by studying in detail current design representations. This direction was motivated by the "Scandinavian approach" claim that design should be based on the traditions or prior work experience of the system's users [1,2]. However, this approach was complicated by the lack of standard representations within the domain. While many designers used a somewhat similar graphic structure chart, the design units within these charts differed in their graphic depictions and in the distinctions they represented.

We initially concentrated on the representations used by the large-scale application designer. Rapid innovation within the domain and rapid increases in application complexity were forcing him to evolve his representations to meet the demands of each new application design. His representations were more detailed and more internally consistent than the representations used by the account executives. The consistency made it easier to determine a core set of units while the detail lent itself to supporting simulation requirements. In a short series of meetings, the designer taught us the basics of his representational scheme. To enhance and verify our understanding, we also reverse-engineered existing applications into the appropriate design representations. Through this process, we determined a core set of design units our environment should provide and we decided the work area should reflect the left-to-right temporal ordering and connective arrows found in the structure chart.

The Iterative Process

During the last four months, the environment has gone through roughly twenty iterations. Some of these iterations concentrated on refining existing design units or adding new design units. In this section, we will highlight only the major shifts in our perspective as a result of this iterative process. Although we used a variety of participation techniques, such as video prototyping, interviewing, and paper mock-ups, a key factor in our success was our ability to rapidly prototype the resulting ideas and further explore them in active use situations. Furthermore, we hope to show iterative design is not a slow and tedious process. With the proper tools, a project can go through many informative iterations in a relatively short time period.

Changing The Design Representation. Subsequent meetings with the large-scale application designer made it clear that a large problem with the current structure chart was the sheer quantity of connective arrows. The resulting visual overload hindered both design construction and communication. The designer claimed that most customers and other non-technical product team members were so daunted by the complexity of the representation that only a few even tried to understand it. Furthermore, the designer rarely added the necessary arcs to describe error and time-out handling. Just describing normal operation flow created enough visual complexity that he didn't want to compound the problem by adding this additional information.

We envisioned two methods to help alleviate the arc problem. First, the environment could provide ways to selectively view or hide subsets of arcs. Or, more radically, the environment could eliminate the need for some or all of these arcs.

We took advantage of a unique aspect of the Agentsheets environment to create a new design representation with fewer arcs. In this environment, every work area has an underlying grid structure similar to the rows and columns

found in spreadsheet applications. We used this grid structure to define a spatially-oriented design language that can be used to determine the placement of and relationship between design units. Figure 4 shows an example of our design unit placement rules. Our language is very simple and is a straightforward extension of the structure chart concept:

- The Horizontal Rule: Design units placed physically adjacent to each other within a row are executed from left-to-right.
- The Vertical Rule: Design units placed physically adjacent to each other within a column describe the set of options or choices at that point in time in the execution sequence.
- The Arrow Rule: Arrows override all previous rules and define an execution ordering.

We quickly integrated these rules into the environment and showed the result to the voice dialog designers. Generally, the language was favorably received. The large-scale application designer liked it because, as he noted, it was a "simple hydraulic model" — like water, everything flowed to the right and down (Figure 2). The account executives liked using the grid structure to guide design unit placement but they wanted to rotate the model's orientation 90°; that is, sequences should be defined vertically and choices horizontally. Some of the account executives were already using placements similar to the vertical rule to describe choice points with more than two options.

Providing Levels of Abstraction. Reducing the quantity of arcs eliminated some of the visual complexity of the design representation. However, designs were still large, complex, and error-prone, since the representations described all necessary low-level operations in a flat structure space. The size and low-level nature of the representations also made it difficult to discern high-level application objectives from the representation.

To address the problems cited above, we decided to build into our environment an abstraction mechanism that would satisfy two objectives. First, it must help designers to control design complexity by structuring the design space into hierarchical layers of abstractions. Second, using the mechanism, a designer should be able to construct each layer such that the layer's high-level objectives are discernable by any viewer with less than ten minutes of effort.

Our abstraction mechanism was built using the hyperagent facility provided in Agentsheets. A hyperagent is an agent that represents a nested worksheet. Each hyperagent has a graphic depiction. In Figure 2, all agents shown in the Voice Mail System window are hyperagents. Double-clicking on a hyperagent's depiction opens its associated worksheet. Thus, double-clicking on the Mail List Section hyperagent opens the worksheet where all mail list operations are defined. Double-clicking on the Delete Mail List hyperagent opens another nested worksheet where the delete operation is defined. During design simulation, the flow of control passes through each nested worksheet and automatically returns to the calling worksheet. The Mail List window is the abstracted representation for the same operations illustrated in the structure chart in Figure 1.

Subsequent evaluation of this abstraction mechanism met with mixed results. The large-scale application designer saw the potential benefits of such a mechanism but noted that it required him to think about his design problems in a new way. In use situations, it became apparent that our current implementation wasn't helping him to overcome the learning curve associated with the new feature. The environment required the designer to use hyperagents in a premeditated fashion. They were chosen from the gallery and placed in the worksheet like all other design units. In one telling incident, when the design was getting large, the designer stated "Now I want to go back and select these clusters and make them into hyperagents but I can't." The account executives were skeptical about the new mechanism. Their designs were small and they liked having a flat structure with everything visible all the time. They

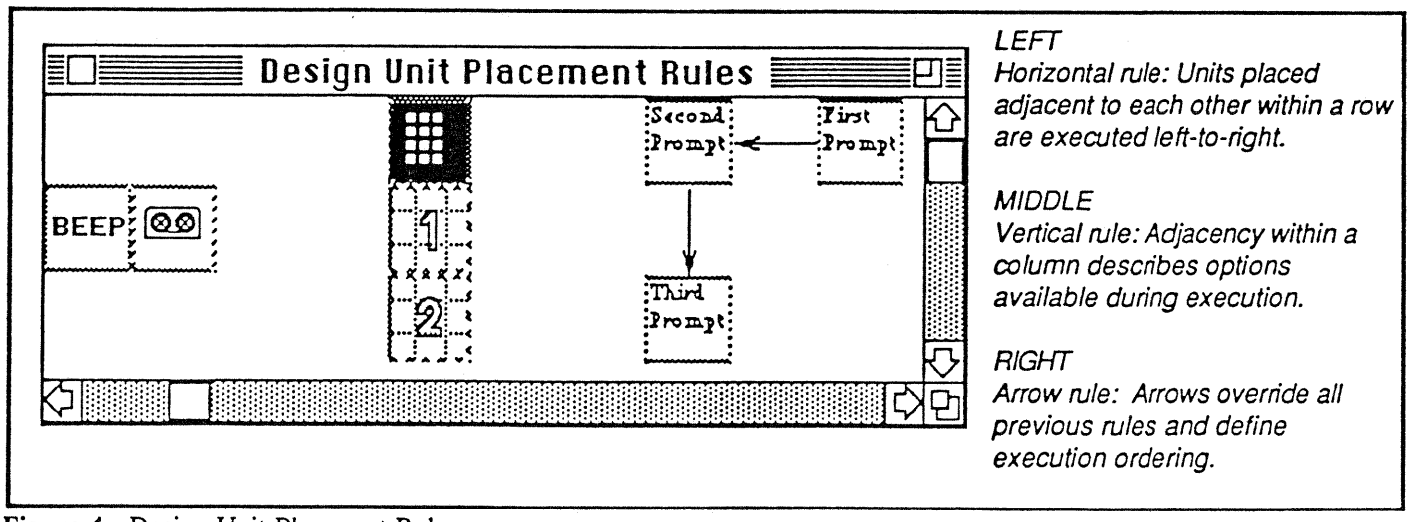


Figure 4. Design Unit Placement Rules.

found the distribution of the design into separate windows confusing. However, they did see some benefits in the reuse potential of these design pieces.

Supporting Reuse of Aggregate Design Units. There were several issues that arose in the previous iteration. First, we needed a mechanism to support post-hoc creation of hyperagents. Designer's should be able to turn a cluster of design units into a hyperagent at any time. Second, hyperagents or design unit clusters should be able to be saved back into the design unit gallery for later reuse. Thirdly, designer's should be able to place either the hyperagent or its constituent design units into the worksheet. Satisfying these requirements would allow the large-scale application designer to benefit from the hyperagent's abstraction mechanism and would allow the account executives to benefit from reuse.

We are extending the Agentsheets system to support cutting, copying, and pasting of design unit aggregates. We are also integrating management of design unit aggregates into the hyperagent concept. Once an aggregate or hyperagent has been cut or copied, it can be pasted into a worksheet or back into the design unit gallery. When an aggregate is pasted into the gallery, the designer will be asked to specify its name and graphic depiction. At this point, the aggregate is turned into a hyperagent and its name and depiction are its representation in the gallery. An additional menu command will be provided that expands a selected hyperagent into its constituent parts in situ. Thus, aggregates can be turned into hyperagents and vice versa. These modifications are currently being carried out and will be evaluated shortly.

LESSONS TO BUILD ON

Our success case for this project is that we got "this far this fast" in a domain that was both complex and innovative. Gould, Boies, and Lewis [4] noted that many development organizations don't practice iterative design methods because they are perceived as being too difficult and time-consuming. We have demonstrated in this project that this is not necessarily the case. By picking the right high-level substrate to build on and by having the right participatory context, we were able to easily iterate through many design changes in a relatively short time. In this section, we will highlight specific factors that contributed to our success.

The Right Participatory Context. As claimed by the Scandinavian design approach [1,2], we benefited from active end-user participation and from having the right kind of participants. Specifically, three factors contributed to the speed with which we were able to proceed.

First, participating end-users have enough organizational status that they are in control of their choice of tools and they are able to make the decision to commit time to the project. Also, a real problem exists that needs to be solved. Shortening product development time and saving money are big motivational factors in any organization.

Second, the project's most influential user participant is a voice dialog design expert. His domain expertise drives our design process. He is continually pushing existing tools and practices to their limits, and thus he has specific ideas based on practical experience about what works, what doesn't, and where existing tools fail in the face of complexity. Relying heavily on his expertise, we were able to choose an appropriate substrate and build the initial system in less than one month.

Finally, cooperative design sessions are frequent, short, and focused. The voice dialog designers are extremely busy people, working hard to get products "out-the-door" while participating in this project. For us, getting together once a week for no more than one hour is optimal. Our job as system designers is to provide "objects-to-reflect-with" to add focus to each of these sessions. These objects are created to highlight specific design issues. Through explorations of these objects, all project participants are able to build a shared understanding of design issues and future directions. Most often, we reflect using the system prototype. We also used video prototyping to explore alternative interfaces to construction, such as a forms view for voice menu specification, and interfaces to a future catalog component containing previous designs and design templates. We used paper mock-ups to explore how to best capture voice dialog design knowledge in the environment.

The Right Substrate To Build On. Choosing the Agentsheets environment was a critical factor in our success to date. While many writers have advocated the need for prototyping tools [1,2,4], no one has really addressed the issue of how to pick the best tool for the task at hand. In this section, we will highlight important criteria to consider when choosing a substrate.

First, we chose a substrate that ran on the same computational platform as existing tools — the Apple Macintosh. This helped us in two ways. First, systems were readily available to participating designers. Taking the latest prototype iteration to a cooperative design session was easy; we just copied a few files onto a diskette and loaded these files onto the designer's system. Second, we bypassed interaction learning curve problems. As long as we kept within the Macintosh user interface guidelines, designers were able to quickly adapt to new functionality since they were already familiar with such interaction techniques.

Second, rather than choosing a general-purpose prototyping tool, we constrained our choice of substrate to one that was designed to solve a particular class of problem elegantly [6]. The class of problem Agentsheets was designed for was well-matched to the voice dialog domain and the functionality envisioned in the future system. Specifically, Agentsheets supported the creation of high-level domain-specific building blocks, design by construction, simulation, and spatial reasoning based on its grid structure. The voice dialog design environment took advantage of all these intrinsic features. Much complex functionality

required little implementation effort on our part. A prototype was built to assist the expert voice dialog designer in evaluating the environment's potential. This prototype was built in a few minutes and consisted of one new bitmapped icon and about a dozen lines of code. In Bodker, Greenbaum, and Kyung [1], two projects are described. In the first, iterative prototyping is facilitated because Hypercard is well-matched to the needs of the future patient record application. In the second project, prototyping is less successful due to the inflexibility of the ORACLE system and its inappropriateness for what they needed to model. We conclude that iterative design would benefit from a wide range of prototyping tools where each tool is optimized for a particular class of problem. Projects should select the appropriate tool based on the application domain and envisioned future use.

Thirdly, we benefitted from the following general properties of Agentsheet:

- Due to the nature of its object-oriented architecture, it is extremely modifiable and extensible.
- It provides a rich set of classes to build on.
- It provides a variety of sample applications to learn from.
- It addresses not only user-interface issues but underlying system functionality as well.

FUTURE DIRECTIONS

Our future research will focus on two themes — how can the design environment further promote building better artifacts and how can the environment enhance design communication within a product team. In the voice dialog domain, there exists a plethora of user interface design standards. In many cases, these standards conflict on basic design issues such as whether or not voice menus items should be mnemonic. Designers must be fluent in all standards since the customer chooses which one to follow. We are beginning to investigating how to incorporate this often conflicting design knowledge into our environment.

Our current efforts have mainly focused on supporting the designer in isolation. However, a large part of the designer's job is communicating the design to the entire product team. We will investigate these communication needs and how best to accommodate them through the generation of alternative design perspectives tailored to intended audiences and the task at hand. We believe these tailored perspectives will facilitate communication between product team members with differing objectives and backgrounds, e.g,. marketing, testing groups, and customers.

ACKNOWLEDGEMENTS

Special thanks go to Mike King and Alex Repenning — without them, this project would not exist. Terry Roberts and the Audiotex group provided much valuable voice dialog and video prototyping expertise. We also thank John Rieman, Gerry Stahl, Scott Henninger, Gerhard Fischer, and Clayton Lewis for their comments on this paper. This research was funded by a grant from US West Advanced Technologies.

BIBLIOGRAPHY

1. Bodker, S., Greenbaum, J. and Kyung M. Setting the Stage for Design as Action, in Design at Work: Cooperative Design of Computer Systems. Lawrence Erlbaum Associates Hillsdale, New Jersey 1991.

2. Ehn, P. Work-Oriented Design of Computer Artifacts, Arbetslivscentrum, Stockholm, 1989 (Second Edition).

3. Fischer, G. and Lemke, A. Construction Kits and Design Environments: Steps Toward Human Problem-Domain Communication Human-Computer Interaction 3, 3 (1988) 179-222.

4. Gould, J.D., Boies, S.J., and Lewis, C. Making Usable, Useful, Productivity-Enhancing Computer Applications, CACM, 34, 1 (Jan. 1991)

5. Halstead-Nussloch, R. The Design of Phone-Based Interfaces for Consumers. In Proc. CHI'89 Human Factors in Computing Systems (Austin, April 30-May 4, 1989) ACM Press, pp. 347-352.

6. Repenning,A. Creating User Interfaces with Agentsheets. To appear in ACM/IEEE Proceedings of SAC 91 (Kansas City, March ,1991).