Reprint

Proceedings of the 1992 ACM/SIGAPP Symposium on Applied Computing, Kansas City, March 1-3, 1992, pp. 1199-1207

Using Agentsheets to Create a Voice Dialog Design Environment[1]

Alex Repenning
Tamara Sumner

Department of Computer Science and Institute of Cognitive Science
Campus Box 430
University of Colorado, Boulder CO 80309
(303) 492-1218, ralex@cs.colorado.edu, sumner@cs.colorado.edu
Fax: (303) 492-2844

Keywords:

agents, agentsheets, construction kits, design, design environments, grids, human-computer interaction, iconic programming environments, object-oriented programming, phone-based interfaces, spatial reasoning, visual programming, voice dialog applications.

Abstract

Agentsheets is a tool for building interactive, graphical systems. It combines the ease of use of a construction kit with the flexibility of a visual programming environment. Agentsheets uses a grid structure to clarify spatial relationships such as adjacency, relative and absolute position, distance, and orientation. System designers can use the Agentsheets substrate to quickly create tailored, domain-specific graphical applications requiring simulation and spatial representations. The design and implementation of a voice dialog design environment using the Agentsheets substrate is described. By using Agentsheets, a mixed team of professional voice dialog designers and academic researchers were able to design and build a substantial core design environment in less than four months.

---

[1] Also University of Colorado technical report CU-CS-576-92, January 1992

Reprint

Using Agentsheets to Create a Voice Dialog Design Environment

Alex Repenning
Tamara Sumner

Department of Computer Science and Institute of Cognitive Science
Campus Box 430
University of Colorado, Boulder CO 80309
(303) 492-1218, ralex@cs.colorado.edu, sumner@cs.colorado.edu
Fax: (303) 492-2844

Abstract

Agentsheets is a tool for building interactive, graphical systems. It combines the ease of use of a construction kit with the flexibility of a visual programming environment. Agentsheets uses a grid structure to clarify spatial relationships such as adjacency, relative and absolute position, distance, and orientation. System designers can use the Agentsheets substrate to quickly create tailored, domain-specific graphical applications requiring simulation and spatial representations. The design and implementation of a voice dialog design environment using the Agentsheets substrate is described. By using Agentsheets, a mixed team of professional voice dialog designers and academic researchers were able to design and build a substantial core design environment in less than four months.

Keywords

agents, agentsheets, construction kits, design, design environments, grids, human-computer interaction, iconic programming environments, object-oriented programming, phone-based interfaces, spatial reasoning, visual programming, voice dialog applications.

Introduction

Users want to make computers do certain tasks for them. Traditionally, we refer to this ambition and the means to attain it as programming. Programming is difficult and many different approaches towards making it easier have been developed.

Construction kits have been shown to be effective tools for human-computer interaction [2]. Designers using construction kits create systems by laying out graphical building-blocks instead of implementing systems in a conventional programming language. These building blocks provide powerful abstractions but are usually domain specific and therefore not applicable to a broad range of different applications. In situations in which the building blocks are inadequate, users will be forced to resort to programming at a much lower level of abstraction.

Visual programming systems are supposed to simplify programing by capitalizing on human spatial reasoning skills [1, 9]. Visual programs are created by drawing building blocks and establishing relationships among them. The low level building blocks of general purpose visual programs are close in their semantics to conventional programming. Often visual programming systems can be viewed as syntactic variants of existing conventional programming languages, e.g., boxes representing procedures, functions, etc. The composition of non-trivial functionality from these building blocks is beyond the ability of a casual computer user.

Agentsheets is a tool for building interactive, graphical applications. It combines the ease of use of a construction kit with the flexibility of a visual programming environment. Agentsheets supports the creation and animation of a variety of graphical representations. A grid structure is provided which can be used to clarify spatial relationships between objects such as adjacency, relative and absolute position, distance, and orientation. A comparison of Agentsheets to other systems can be found in [7].

In a typical application of Agentsheets, a system designer will define the look and behavior of domain-specific building blocks. These building blocks constitute the elements of a high-level visual programming language which can be used readily by end users. End users arrange these building blocks in a work area. The work area has an underlying grid structure analogous to the rows and columns in a spreadsheet. Relationships between building blocks can be explicitly specified by connecting blocks with links or implicitly specified simply by position within the grid structure. Each building block has an associated behavioral component which allows the block to perform an action in response to some stimulus. At the user's request, the behaviors of building blocks can be executed.

Agentsheets is designed to be a high-level substrate to build on and not an end-product. There are a large number of existing applications from widely varying domains that have been built using the system, including a river basin management system, a children's storybook tool, a front-end to a power station's expert system [8], and a voice dialog application design environment [12].

This paper begins by describing the architecture of Agentsheets and explaining how a new graphical application is created using the system. Next, the properties of several spatially-oriented graphic representations are given and their use in visual programming environments is described. The remainder of the paper demonstrates how a voice dialog design environment was implemented using the Agentsheets substrate. The voice dialog environment used the grid-based spatial reasoning provided by Agentsheets to quickly create a new, easy-to-comprehend, graphic design representation. A voice dialog design simulation facility was implemented that combines a visual execution trace with audio output.

Architecture of Agentsheets

Agents and Agentsheets

The basic components of Agentsheets are agents [3, 5]. An agent is a computational unit either passively reacting to its environment, or, more typically, actively initiating actions based on its perception. These actions, in turn, may impact the environment.

The Agentsheet is a grid-structured agent container. Every agent has a graphical depiction that is visible in the Agentsheet. Figure 1 shows an Agentsheet depicting a simple electrical system. In this system, the look as well as the behavior of the system components like voltage sources, switches, bulbs and even individual wire segments are captured by agents.

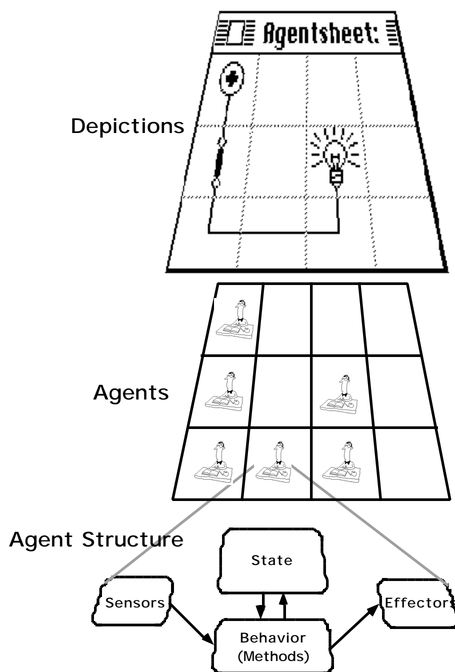


Figure 1. The Structure of an Agentsheet

The *depictions* in Figure 1 show the graphical representation of an Agentsheet as it is seen by a user. Depictions can represent the class or the current state of an agent. For instance, the symbol of an electrical switch denotes a switch agent. Furthermore, different states of the switch are mapped to different variations of depictions, e.g., an open switch versus a closed switch.

The agents, corresponding to the cells in the depiction level, consist of:

- *Sensors*. Sensors invoke *methods* or procedural actions of the agent. They are triggered by the user (e.g., clicking at an agent) or by another agent.
- *Effectors*. Effectors are mechanisms to communicate with other agents by sending messages. The messages, in turn, activate sensors of the agents to be effected. Additionally, effectors also provide means to modify the agent's depiction or to play sounds.
- *Behavior*: The built-in agent classes provide a default *behavior* defining reactions to all sensors. In order to refine this behavior, methods associated with sensors can be shadowed or extended making use of the object-oriented paradigm.
- *State*. The state describes the condition the agent is in.
- *Depiction*. The depiction defines the look or graphical representation of the class and state of the agent.

Galleries

The gallery is where depictions or graphic images of agents are defined and stored. Depictions are defined by using the provided bitmap editor or by modifying and combining existing depictions. Several features are provided to help create depictions such as graphic rotation, inversion, and overlay.

Hyperagents

Agentsheets provides an abstraction mechanism called *hyperagents*. A hyperagent is a placeholder for an entire Agentsheet. In other words, Agentsheets can be hierarchically nested inside of Agentsheets. The use of hierarchical representations is crucial for large and complex applications.

Platforms

Agentsheets is built on top of an object-oriented system [11]. It is written in Common Lisp [10] and has been implemented on Macintosh and Sun platforms. Agentsheets, agents, and galleries are all object classes. The system provides a rich set of built-in classes for creating graphic representations and for managing user interactions.

Applying Agentsheets

Complex applications cannot be constructed by casual computer users employing general purpose programming environments. Agentsheets anticipates two types of users: a system designer and an end user.

In a typical application scenario, a system designer uses Agentsheets to create a high-level visual programming system. This system is essentially a construction kit tailored to the user's specific domain. The system designer maps the application domain semantics to a set of graphical building blocks and defines the meaning of spatial relationships between these blocks. Together, the building blocks and the spatial relationships between blocks comprise a graphic representation for the domain. Each building block is an agent. A graphical depiction and a class must be defined for each type of agent. Defining the class includes the design of a data structure for managing the agent's internal states and a set of methods determining the agent's behavior. This class definition is implemented in Common Lisp. The behavior of an agent does not have to be defined from scratch; it can be constructed incrementally by refining existing agent classes. For instance, a comprehensive selection of fundamental sensors; i.e., user interactions such a selection, dragging and other mouse events, is inherited when refining built-in agent classes.

End users build programs with the resulting system by selecting familiar objects from a gallery and placing them into a work sheet. The layout of objects in the worksheet defines the program's collective meaning.

Agentsheets' Support for Spatial Reasoning

Human problem solving often involves spatial reasoning. Concepts and relationships of a problem domain are mapped to a diagrammatic representation, which is then used for reasoning and communication. These diagrammatic representations may be sketches of things in the world, e.g., a city map, or they may depict abstract entities, e.g., a flow chart. Some of the characteristics of spatial reasoning that are relevant to our work on Agentsheets include the distinction between pseudo and strict-spatial representations, domain-oriented representations, and the use of grids.

Pseudo versus Strict-Spatial Representations

Spatial information has many different properties. Part of designing a visual representation is choosing which properties to make use of. For instance graphical objects always have a position and a dimension. A representation may or may not assign meaning to these attributes. In flow charts, position and dimension are not interpreted. Relationships are specified by links between elements. In a blueprint, position and dimension carry the bulk of the representation's meaning. We call spatial representations like flow charts, where position and dimension are not interpreted, *pseudo-spatial* representations. *Strict-spatial* representations, like blueprints, are those in which position is interpreted.

- A strict-spatial representation uses the actual positions of objects to convey meaning. For instance in Figure 2, the representation describes an implicit "roof above frame" relationship.

Figure 2. Strict-spatially related roof and frame of a house

- A pseudo-spatial representation does not make use of actual positions of objects. Instead, explicit cues are used. In Figure 3 arrows are employed as explicit cues of the "roof above frame" relationship. That is, the relative positions of the roof and frame are completely irrelevant with respect to the semantics of the representation.

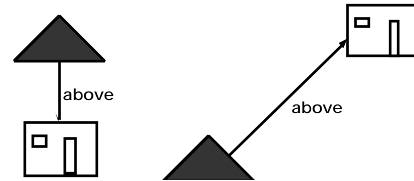


Figure 3. Pseudo-spatially related roofs and frames of houses

Some of the limitations of visual programming system result from their reliance on pseudo-spatial representations. Pseudo-spatial representations are chosen because of their flexibility and their general purpose nature. Strict-spatial representations appear to be preferable to pseudo-spatial representations in cases where the positional information of objects can be mapped to physically natural concepts understood by users. The components in a chip geometry design interface are related strict-spatially, whereas the elements of a flow-chart are only pseudo-spatially related. The relative positions of chip components have physically grounded meaning. In other words, the designer of a chip gets many more facts out of a chip layout than just pure topological information.

Using a Grid to Clarify Strict-Spatial Representations

Grids are well known in the area of graphic design, typography, and architectural design. Müller-Brockman characterizes the purpose of grids as follows [6]:

> "The use of a grid system implies the will to systematize, to clarify; the will to penetrate to the essentials, to concentrate; the will to cultivate objectivity instead of subjectivity;.."

Grids clarify strict-spatial representations by:

- *Avoiding Brittleness.* Without a grid, spatial relations can become very brittle. That is, moving an object on the screen one pixel may change its spatial relation to another object from an adjacent relation to a non-adjacent relation. While this might reflect the intention of a user, it is more likely to lead to non-evident problems.
- *Making Spatial Relationships Obvious.* The use of grids increases the transparency of spatial relationships considerably. The strict-spatial relationships of objects in a grid are obvious to the user. Thus, while grids limit the flexibility of placement, they help clarify the relationships between objects.
- *Making It Easy To Spot Substructures.* Common substructures occur frequently in many graphical representations. Constrained placement within the grid eases the recognition and location of commonly occurring substructures.

Additionally, Agentsheet grids further enhance strict-spatial representations by:

- *Providing Implicit Communication Paths.* Communication among agents is accomplished implicitly by placing them into the grid. No explicit communication channel between agents has to be created by the user. In the circuit Agentsheet shown in Figure 1, the electrical components get “wired-up” simply by placing them into adjacent positions. The individual agents know how to propagate information (flow in this case), e.g., the voltage source agent will always propagate flow to the agent immediately below it.

Domain-Orientation

Application domain-oriented representations often rely on complex spatial representations. Examples of domain-oriented spatial representations include architectural blueprints and kitchen floor plans. Interpretation of these diagrams requires domain knowledge. For instance, when looking at a kitchen floor plan, it is only knowledge about kitchens that lets the viewer infer that nested rectangles represent top and bottom cabinets. Construction kits embody domain-oriented spatial metaphors by providing domain-specific building blocks and by tailoring the spatial representation supported in the work areas to reflect important domain distinctions. It is difficult to extend these environments to support other domains since it is usually not possible to redefine the spatial representation supported in the work areas.

Examples of Spatial Representations in Visual Programming Environments

We can characterize visual programming environments by their degree of domain orientation and by their use of spatial representations to represent relationships among objects:

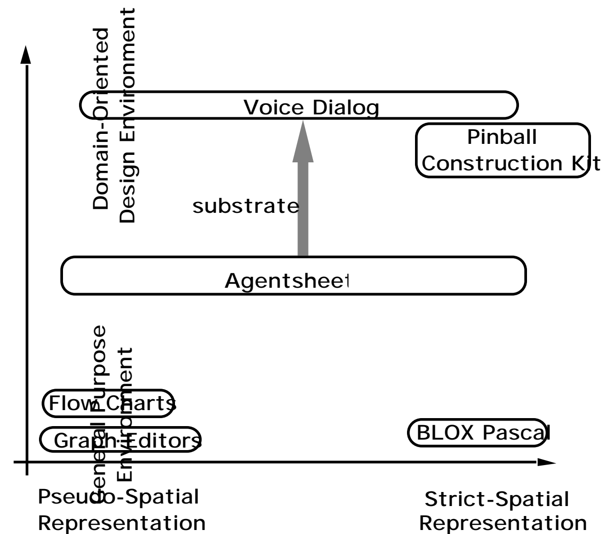


Figure 4. Domain-Orientation vs Use of Spatial Representations

In the lower left corner of Figure 4, we have general purpose systems like flow-charts and graph editors. These systems make use of pseudo-spatial representations, i.e., they ignore the positional information of graphical objects, and they lack any domain specific features.

BLOX Pascal is a visual programming environment based on strict-spatial representations, with no underlying grid. The semantics of a BLOX Pascal [4] diagram is given by the position of the individual blocks (Figure 5). A strict-spatial representation is used to express syntactic rules between Pascal primitives. There is no specific domain orientation in this visual representation of a general-purpose programming language.

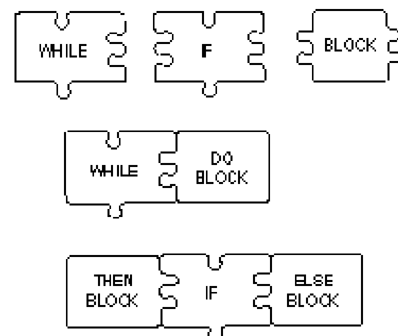


Figure 5. BLOX Pascal

The Pinball construction kit [2] is extremely domain specific and inflexible (Figure 6). For instance it would be a non-trivial task to use the pinball construction kit to add two numbers. The functionality of a designed pinball system is defined by the layout of its components, i.e., the representation is strict-spatial, again without the support of a grid.

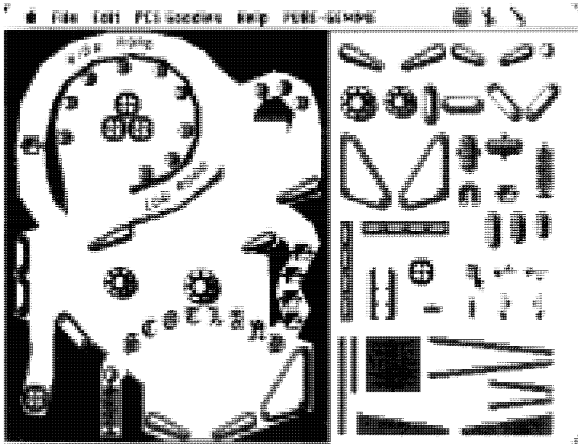


Figure 6. Pinball Construction Kit

Agentsheets supports strict-spatial representations as well as pseudo-spatial representations. Furthermore, the user definable correspondence between spatial representations and semantics is neither tied to the level of general programming languages nor is it limited by domain-specific issues. Agentsheets, therefore can be viewed as a *domain-tailorable spatial reasoning substrate* embracing a wide range of spatial representations. In the second part of the paper, a voice dialog design environment implemented using Agentsheets is described. The paper explains why the initial implementation focusing on pseudo-spatial representations has migrated towards a second implementation favoring a strict-spatial representation.

The Voice Dialog Design Problem

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. Our involvement in this field was part of a collaborative research effort between the University of Colorado and US West's Advanced Technologies Division. Designers within US West presented a compelling case as to why the voice dialog domain would be an excellent framework for pursuing our research. The designers were facing challenging design problems - innovation and increasing complexity within the voice dialog application domain was making it harder to design and develop products within the necessary time and cost constraints.

Understanding the Voice Dialog Application Domain

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 that are only now being investigated. Thus, key challenges facing designers are large increases in complexity and rapid innovation within the application domain.

Some designers, in an attempt to deal with increasing complexity, have moved from textual design specifications to graphical representations similar to flow charts, called "structure charts". As illustrated in Figure 8, 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. Design elements 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. Currently, structure charts are constructed and maintained in MacDraw.

For complex applications, structure charts can grow very large and span tens of pages. These structure charts depict a flat design space; all aspects of the design must be depicted at the same low level of detail. A major problem facing voice dialog designers is the comprehension and maintenance of these large, unstructured representations. While it is easy to draw a rectangle to represent a subsystem, simple graphics programs such as MacDraw provide very little assistance in managing and viewing subsystems in hierarchically decomposed representations.

A second difficulty voice dialog designers must overcome is the conceptual gap between the visually-based design representation and the purely audio end product. For instance, to conserve storage space, phrases are recorded and recombined on-the-fly into the application's prompts and messages. Errors often occur when combining phrases. These purely auditory design errors are uncovered only when a design prototype with voice output is constructed and all prompts and messages can be heard.

Figure 8. Structure Chart

Why Choose Agentsheets for Voice Dialog Design?

We chose Agentsheets for several reasons. First, it runs on the same platform (Macintosh) as current voice dialog design tools. Second, Agentsheets fulfills the requirements uncovered in our preliminary task analysis:

- Designers rely on graphic design representations consisting of domain-specific design units such as prompts and voice menus as well as general purpose programming design units such as decision junctions. These representations suggest combining a construction kit approach with a visual programming environment.
- Designs are large and complex, containing many highly interconnected design units. Spatial relationships between design units is the essence of the graphic representation. Specifically, we wanted a substrate that already supported creating pseudo-spatial design representations.
- It is difficult to represent increasingly complex application designs in the flat, low-level structure chart representation. An abstraction mechanism that made it easy to create, view, and manage voice dialog subsystems was needed.
- The voice dialog domain suffers from a conceptual gap between the visually-oriented design representations and the auditory end-product. This gap can be bridged by prototyping or by design simulation. In Agentsheets, design simulation is simply a matter of executing the behavior associated with each building block.

Using Agentsheets to Create a Voice Dialog Design Environment

This section focuses on how Agentsheets supported the iterative design and implementation of a voice dialog design environment. What we hope to convey is how picking a well-suited high-level substrate enabled a substantial system to be built in less than four months. We begin with a brief description of the current version of the system.

Overview of the Current Voice Dialog Design Environment

The voice dialog design environment provides an on-screen gallery of voice dialog design units, such as menus and prompts, and a work area for design construction and simulation. “Designing” involves placing design units into the work area in accordance with three design unit placement rules. These rules are described in the following section. 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. The design of a voice mail system constructed in this environment is illustrated in Figure 9.

Using the voice dialog design environment, designers work with meaningful domain abstractions such as prompts and menu design units. A portion of the design environment’s gallery of

design units is shown in Figure 9. Each design unit is an agent, consisting of a graphical depiction (bitmap), an internal state, and an associated behavioral component. Figure 10 illustrates how the prompt design unit is an agent. Some design units, such as the prompt unit, allow the designer to replace the generic graphic depiction with a meaningful text description.

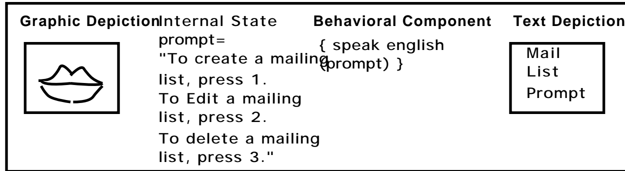


Figure 10. A prompt design unit is an agent.

All voice dialog design units are built on top of existing Link and Value agents. By refining these built-in agents, all design units automatically inherited functionality for:

- basic user interactions;
- allowing designers to replace the graphic depiction with a textual depiction; and
- establishing and managing connective arrows between design elements.

We have described the current state of the system. The next sections discuss how the graphic representation evolved during the design process.

Creating a Voice Dialog Design Representation

Our first prototype of the design environment closely followed the structure chart representation used by the the designers at that time. However, a major shortcoming of the current structure chart representation was the sheer quantity of connective arrows. One voice dialog designer claimed that in meetings with customers and other non-technical product team members, most 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 cancellation handling since this would only compound the complexity problem.

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 the grid-based, strict-spatial reasoning provided by Agentsheets to create a new design representation with fewer arcs. 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 11 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.

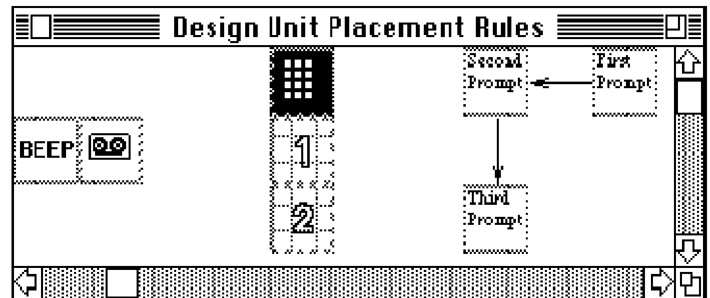
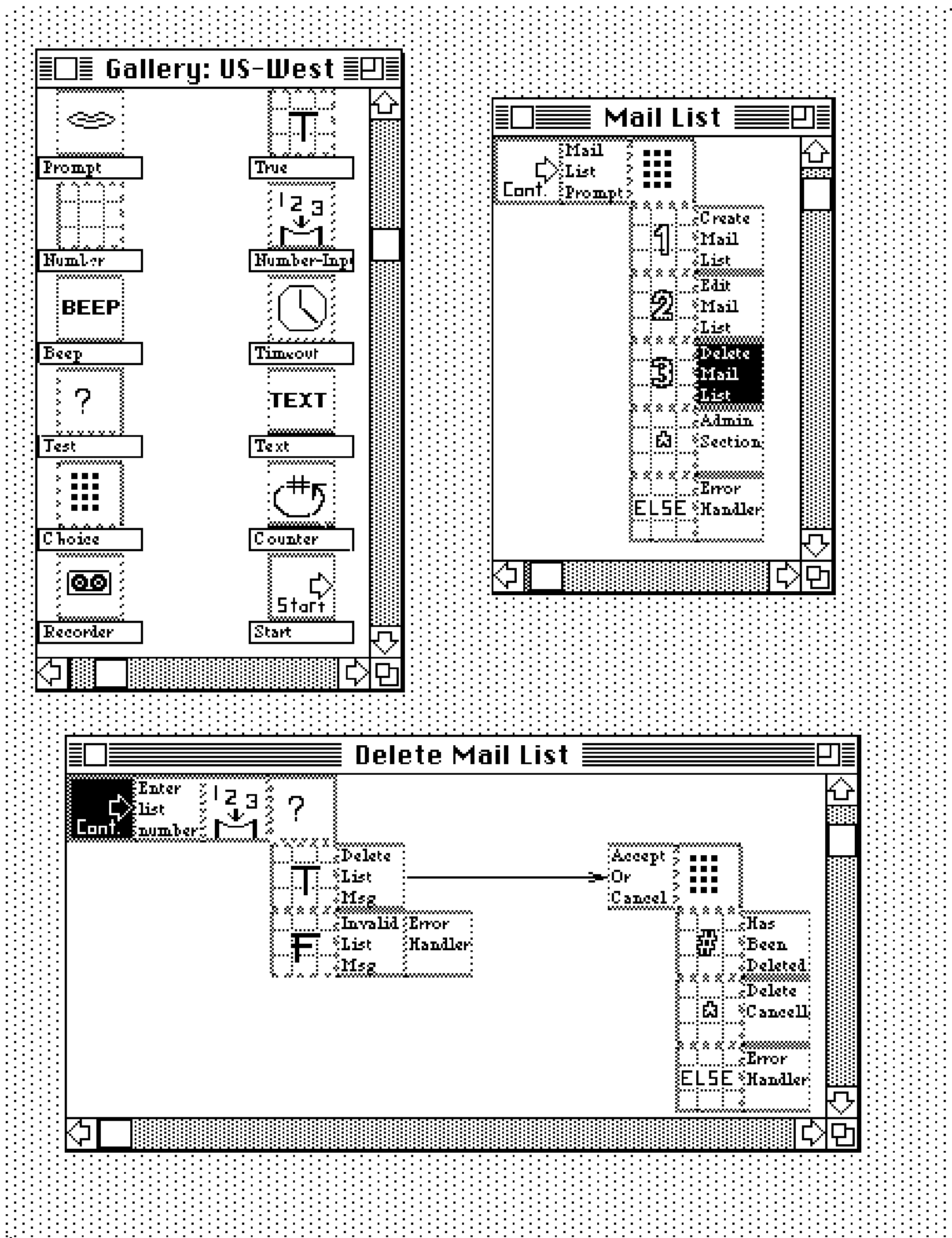


Figure 11. Design Unit Placement Rules

These rules eliminated most of the normal flow arcs but had little effect on cancellation and error flow arcs. To assist the designer when constructing or debugging these flow types, we also added support for color-coded, typed arcs between design units. Normal flow arcs are green, error flow are red, and cancellation arcs are yellow. This functionality was easily added by refining existing Link-agent and Link-Agentsheet built-in classes. Mechanisms for selectively viewing only arcs of a specific type are provided.



The Gallery contains voice dialog design units used in design construction. The remaining three windows are all nested, design work areas depicting the graphic representation of a hypothetical voice mail application. Simulation of a work area (and its nested work areas) is initiated by double-clicking on any start or continue design unit.

Figure 9. Voice Dialog Design Environment

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 graphic representation.

To address the problems cited above, we created a decomposition mechanism based on Agentsheets' hyperagent abstraction mechanism. Each hyperagent has a graphic depiction which the designer can replace with meaningful text. In Figure 9, many agents in the Mail List window are hyperagents. Double-clicking on a hyperagent's depiction opens its associated worksheet. For instance, double-clicking on the Delete Mail List hyperagent opens the worksheet where the delete mail list 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 hierarchically abstracted representation for the same operations illustrated in the structure chart in Figure 8.

Design Simulation

Prototyping is an important phase during the design and implementation of complex voice dialog applications. Prototyping currently serves several roles. First, it helps bridge the medium gap and discovers purely auditory defects in the design. Second, the prototype is an important vehicle for communicating the design to the customer. Customers cannot envision the end-product from the graphic design representation and rely mainly on the prototype to verify that the design corresponds to the product they desired. Additionally, all usability testing is performed with prototypes.

Unfortunately, a single prototype is rarely satisfactory. Invariably, implementing many prototypes adversely impacts the project's cost and time constraints. Design simulation reduces the amount of expensive and time-consuming prototyping required.

In the voice dialog design environment, every design unit has an associated behavioral component that executes an action during design simulation. Some of these actions result in audio output, some collect user touch-tone input, and others perform internal system actions such as managing data or evaluating conditions.

The spatial relationships between design units describes the order of execution flow when simulating the design representation. Execution of the design representation proceeds by following horizontally adjacent design units or by traversing connective arcs. When a vertical choice point is reached, the appropriate condition is evaluated and one of the execution options defined within the column is taken. During design simulation, a visual trace of the execution path is combined with an audio presentation of all voice prompts and messages encountered.

Summary

Agentsheets is a high-level substrate facilitating the creation of domain-specific visual programming environments. A system designer maps the domain semantics to a set of graphical building blocks and defines the meaning of spatial relationships between these blocks.

Agentsheets were used to design and build a real-world, complex system - a voice dialog application design environment. Key aspects of Agentsheets such as the underlying grid structure and the rich set of object-oriented building blocks were used to quickly prototype and implement new voice dialog design representations. Agentsheets' visual programming infrastructure allowed a design simulation tool to be constructed with very little effort.

The following Agentsheet properties contributed to the success and rapid development of the voice dialog design environment:

- *High-level Support for Spatial Representations.* By building on top of existing agent classes, we were able to quickly implement an initial design environment based on the existing structure chart representation. Using the underlying grid structure, we created a new, easier-to-comprehend graphic representation that combined both strict and pseudo-spatial representations.
- *Design by Construction.* Voice dialog designers can construct new applications by manipulating meaningful domain elements such as menus and prompts. Much of the functionality of these design units did not have to be implemented from scratch, but was inherited by refining built-in agent classes.
- *Design Simulation.* Design simulation was a natural by-product of using the Agentsheets system; we only had to define an action for each design unit and a graphic representation dictating execution flow.
- *A Hierarchical Abstraction Mechanism.* Existing voice dialog design representations suffered from too much complexity in a flat design space. Using hyperagents, we were able to create a hierarchical design

decomposition mechanism that made it easy to create, view, and simulate nested voice dialog subsystems.

Acknowledgements

This research was funded by the United States Bureau of Reclamation through the Advanced Decision Support program, Hewlett-Packard Switzerland and Advanced Technologies of US WEST. Many thanks to our reviewers: Clayton Lewis, John Rieman, Carrol Marra, Scott Henninger, Brent Reeves, Andreas Girgensohn, and David Redmiles.

References

1. S.-K. Chang, *Principles on Visual Programming Systems,* Prentice Hall, New Jersey, 1990.

2. G. Fischer and A. C. Lemke, "Construction Kits and Design Environments: Steps Toward Human Problem-Domain Communication," *HCI,* Vol. 3, pp. 179-222, 1988.

3. M. R. Genesereth and N. J. Nilson, *Logical Foundations of Artificial Intelligence,* Morgan Kaufman Publishers, Inc., Los Altos, 1987.

4. E. P. Glinert, "Towards "Second Generation" Interactive, Graphical Programming Environments," *IEEE Computer Society, Workshop on Visual Languages*, Dallas, 1986, pp. 61-70.

5. M. Minsky, *The Society of Minds,* Simon & Schuster, Inc., New York, 1985.

6. J. Müller-Brockmann, *Grid Systems in Graphic Design: A Visual Communication Manual for Graphic Designers, Typographers and Three Dimensional Designers.,* Verlag Arthur Niggli, Niederteufen, 1981.

7. A. Repenning, "Agentsheets: Applying Grid-Based Spatial Reasoning to Human-Computer Interaction," *Technical Report,* CU-CS-547-91, Department of Computer Science, Campus Box 430, University of Colorado at Boulder, Boulder, Colorado 80309-0430, 1991.

8. A. Repenning, "Creating User Interfaces with Agentsheets," *1991 Symposium on Applied Computing*, Kansas City, MO, 1991, pp. 190-196.

9. N. C. Shu, "Visual Programming: Perspectives and Approaches," *IBM Systems Journal,* Vol. 28, pp. 525-547, 1989.

10. G. Steele L., *Common LISP: The Language,* Digital Press, 1990.

11. M. Stephik and D. G. Bobrow, "Object-Oriented Programming: Themes and Variations," *The AI Magazine,* , pp. 40-61, 1984.

12. T. Sumner, S. Davies, A. C. Lemke and P. G. Polson, "Iterative Design of a Voice Dialog Design Environment," *Technical Report,* CU-CS-546-91, Department of Computer Science, Campus Box 430, University of Colorado at Boulder, Boulder, Colorado 80309-0430, 1991.