Reprint

Repenning, A., & Citrin, W. (1993). Agentsheets: Applying Grid-Based Spatial Reasoning to Human-Computer Interaction. In 1993 IEEE Workshop on Visual Languages, Bergen, Norway: IEEE Computer Society Press.

VL '93

Agentsheets: Applying Grid-Based Spatial Reasoning to Human-Computer Interaction

Alex Repenning[1], Wayne Citrin[2]

[1]Department of Computer Science and Institute of Cognitive Science
[2]Department of Electrical Engineering
Campus Box 430
University of Colorado, Boulder CO 80309
303 492-1218, {ralex, citrin}@cs.colorado.edu

Keywords:

Agents, agentsheets, cellular automata, construction kits, spatial reasoning, spreadsheets, spatial metaphor, temporal metaphor, human-computer interaction, object-oriented programming, data-flow, iconic programming environments, visual programming, grids, building blocks.

Abstract

This paper argues that grid-based spatial reasoning can significantly improve human-computer interaction. While grids constrain the user's ability to position objects on a screen on one hand, they greatly increase the transparency of *functional relationships* among these objects on the other hand. A system called Agentsheets employs a spatio-temporal metaphor of communicating agents sharing a structured space. This domain-independent metaphor can be used to create domain-oriented visual programming systems. This paper explains how Agentsheets fits into the spectrum of domain-orientation ranging from general purpose visual programming languages to domain-oriented construction kits, gives a short introduction of Agentsheets, sketches sample applications, and evaluates the contribution of grid-based spatial reasoning to human-computer interaction.

Agentsheets: Applying Grid-Based Spatial Reasoning to Human-Computer Interaction

Alex Repenning[1], Wayne Citrin[2]

[1]Department of Computer Science and Institute of Cognitive Science
[2]Department of Electrical Engineering
Campus Box 430
University of Colorado, Boulder CO 80309
(303) 492-1218, {ralex, citrin}@cs.colorado.edu

ABSTRACT

This paper argues that grid-based spatial reasoning can significantly improve human-computer interaction. While grids constrain the user's ability to position objects on a screen on one hand, they greatly increase the transparency of functional relationships among these objects on the other hand. A system called Agentsheets employs a spatio-temporal metaphor of communicating agents sharing a structured space. This domain-independent metaphor can be used to create domain-oriented visual programming systems. This paper explains how Agentsheets fits into the spectrum of domain-orientation ranging from general purpose visual programming languages to domain-oriented construction kits, gives a short introduction of Agentsheets, sketches sample applications, and evaluates the contribution of grid-based spatial reasoning to human-computer interaction.

1. INTRODUCTION

The design and implementation of a human-computer interface is, no doubt, a verifiably hard task. Construction kits have been shown to be effective tools for human-computer interaction [3]. Designers using construction kits create systems by composing building-blocks instead of implementing systems on a conventional programming language level. These building blocks serve as abstractions of complex functionality. Hardware designers, for instance, typically think in terms of integrated circuits and not on the level of individual transistors; that is, they view integrated circuits as abstractions of compositions of simpler constituents.

Visual programming systems, on the other hand, are supposed to help users to program computers by capitalizing on human spatial reasoning skills [2, 18]. Visual programs are created by drawing building blocks and establishing relationships among them.

The designers of tools having graphical user interfaces are faced with a dilemma regarding the degree of domain-orientation represented by the building blocks:

- *Construction Kits: High Level Building Blocks* provide powerful abstractions but are quite likely domain-oriented and therefore not applicable to a broad palette of different applications.
- *General-Purpose Visual Programs: Low Level Building Blocks* are used as a general purpose tool for a wide set of applications. However, the composition of non-trivial functionality from these building blocks might be beyond the reach of a casual computer user.

The low level building blocks of general-purpose visual programs are too close in their semantics to conventional programming. Often visual programming systems can be viewed as syntactic variants of existing conventional programming languages; for example, boxes and arrows representing procedures and procedure calls, etc. These systems typically add only little value to their textual counterparts.

The use of high level graphical building blocks in construction kits deserves more attention. In situations in which a construction kit is inadequate, either because it would lead to a very long-winded solution or because the set of building-blocks provided is incomplete, a user will be forced to resort to programming on a much lower level of abstraction. The gap between a building block level and the level of a conventional programming language used to implement the building-blocks is called the "Representation Cliff" [14]. Users not only have to understand the underlying programming language, they also have to know about the possibly very complex transformation between the language constructs (for example, a library consisting of a large set of functions), the behavior, and look of artifacts.

Agentsheets [15, 16] take the edge off the representation cliff by introducing an intermediate level of abstraction between high-level building-blocks and the level of conventional programming languages called the *spatial reasoning* level. Agentsheets make use of a *grid structure* to clarify essential spatial relationships such as *adjacency*, relative and absolute *position*, *distance*, and *orientation*. These relationships, easy for the user to understand and manipulate, allow the system to create implicit communication channels between agents.

The spatial reasoning level employed by Agentsheets supports the design of domain-oriented visual programming systems by:

- providing a *spatial metaphor* of communicating agents sharing a structured space
- providing a *temporal metaphor* of simple kinetic entities behaving like autonomous processes that can move in space
- being *domain-independent*; the spatial and the temporal metaphor are not application domain-oriented
- being *programming-language-independent*; the spatial and the temporal metaphor do not reflect the underlying programming language

2. THE AGENTSHEETS SYSTEM

The basic components of Agentsheets are agents [6, 10]. Generally, an agent is a thing (or person) empowered to act for a client. Each agent, in the Agentsheets sense, is a fine grained, autonomous unit of behavior. In contrast to many other agent-based systems, the Agentsheets agents **are not** user interface agents playing the role of mediators between the user and an application. Instead, the Agentsheets agents **are** the application. That is, the interaction between a user and the agents is, in fact, the main focus of an Agentsheets application.

2.1 Human-Agent Interaction

The Agentsheets system is a tool for *visual programming system designers* and *end users*. A spatio-temporal metaphor consisting of "communicating agents sharing a structured space" (see Figure 1) is used by a visual programming system designer to create a visual language tailored to a problem domain. End users program by placing agents in meaningful arrangements within a grid where what is "meaningful" is determined by the application domain.

The human-agent interaction in Agentsheets combines direct manipulation techniques [17] with simulation. Agents can simply react to user or agent sensor stimuli (for instance, mouse clicks, dragging, application of tools, detecting other agents) or they can take the initiative and autonomously invoke actions, such as moving around in an agentsheet, changing their appearance, playing a sound, or involving other agents in the process of computation through effectors. The ability of agents to move extends the spatial metaphor with a kinetic component. Hence, visual programming systems based on Agentsheets are not limited to static, topological relationships between visual entities on the screen.

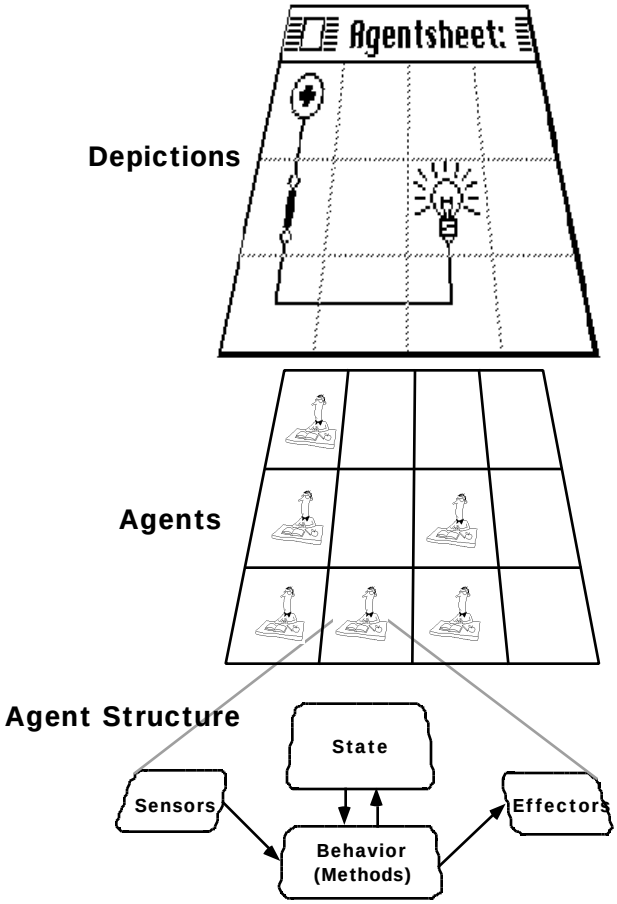

Figure 1. The Structure of an Agentsheet

The *depictions* in Figure 1 show the graphical representation of an agentsheet as it is seen on the screen by users. Each depiction represents the class of an agent; for example, the symbol of an electrical switch denotes a switch agent. Furthermore, different states of an agent are mapped to different depictions; for example, an open switch versus a closed switch. The agents, corresponding to the cells in the depiction level, consist of:

- *Sensors*. Sensors invoke *methods* of the agent. They are triggered by the user (for example, clicking at an agent) or by the Agentsheets process scheduler.
- *Effectors*. Effectors are a mechanism to communicate with other agents by sending messages to agents either using grid coordinates or explicit links (for example Petri Nets: Figure 4). The receiving agents may be in the same agentsheet, in a different agentsheet on the same

computer or even in a different agentsheet on a different computer (connected via network). The messages, in turn, activate sensors of receiving agents.

- *Behavior*: The built-in agent classes provide a default *behavior* defining reactions to all sensors. In order to refine this behavior incrementally subclasses of agents can be defined making use of the object-oriented paradigm [19].
- *State*. The state describes the agent's condition.
- *Depiction*. The graphical representation of the class and state; that is, the *look* of the agent.

2.2 Domain-Oriented Metaphors

The essence of Agentsheets is to use the domain-independent "communicating agents sharing a structured space" spatio-temporal metaphor to create a new spatio-temporal metaphor reflecting application domain-oriented semantics. In the Circuits application (Figure 2) of Agentsheets a visual programming system designer has created a set of agents modeling the behavior of simple electric components like wire pieces, switches, bulbs, electromagnets, and electromagnetic switches. A user creates a circuit by composing it from components. At any time a user can interact with the circuit (for example, by opening and closing switches, adding and removing components).

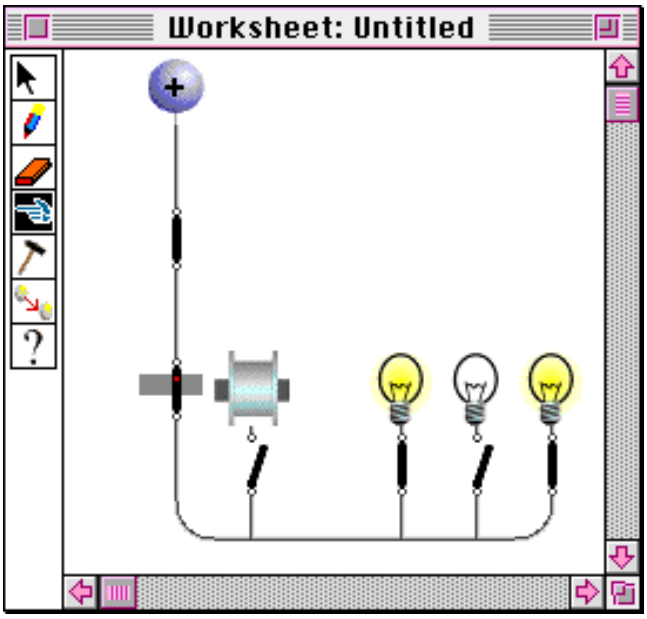

Figure 2. Agentsheets Application: Circuits

The consistency between a real-world situation and a domain-oriented spatio-temporal metaphor is controlled by the person creating the application-domain-tailored visual programming system. The Circuits application maps concepts like the connectivity of components and electromagnetic fields to the adjacency concept being part of the "communicating agents sharing a structured space" metaphor. In Figure 2 a solenoid is created simply by putting an electromagnet and an electromagnetic switch next to each other (the electromagnet it above the leftmost switch in the bottom row of switches).

2.3 Defining Depictions and Behaviors of Agents

The depictions of agents are defined by the visual programming systems designer using the Agentsheets icon editor. The icon editor is an agentsheet in which every pixel is a very simple pixel agent. The collection of all depictions is stored in another agentsheet called the gallery. The gallery serves as agent repository for the visual programming system designer as well as for the end user. Special agent classes are provided that have dynamic depictions, including spread sheet cell agents and colorable agents.

The visual programming designer defines the *reactive* behavior of agents by attaching specialized Lisp code to sensors and defines the *proactive* behavior of agents by setting up tasks that are performed spontaneously at regular intervals. A higher, but less powerful, level of programming is provided to the end user through graphical rewrite rules.

2.4 Platforms

Agentsheets has been implemented on Macintosh computers using Macintosh Common Lisp and on SPARC Stations using Allegro Common Lisp and the Garnet tool kit.

3. RELATED WORK

Agentsheets are related to cellular automata (CA) [20]. Similar to CAs, they define complex global behavior in terms of simple, local relations. CAs also make use of the high degree of regularity furnished by grids. In contrast to CAs, however, Agentsheets contain heterogeneous agents instead of simple homogenous cells. These Agents have a large set of sensors allowing them not only to perceive the state of their environment, i.e., the agentsheet, but also to react to user events (for example, a user clicking at an agent). Furthermore, the state of an agent is visualized by an entire bitmap instead by a single pixel on the screen.

Furnas' BITPICT system employs graphical, two dimensional rewriting rules to augment human spatial problem solving [5]. Like CAs, BITPICT operates on the pixel level.

MAGES [1] shares the idea of agents and grids with Agentsheets. However, the agents employed are very

abstract; that is, they are not intended to represent application domain-oriented entities. Furthermore, the Agentsheets agents are not bound to cells; they can move around in the agentsheet, and multiple agents can inhabit the same cell simultaneously.

Spreadsheets have shown to be powerful tools because they adopted an interaction format with which people were already familiar [4]. Many extensions to spreadsheets have been proposed to further increase their usability. Piersol suggested the use of object-oriented techniques for spreadsheets [13]. In his system a cell may not only be represented with a piece of text, but also with a bitmap. However, in Piersol's system individual bitmaps are not intended to be part of a large composite picture.

Not only are spreadsheets user interfaces, they can also be employed to design user interfaces [7, 8, 12, 21]. Agentsheets go one step further by unifying the graphical user interfaces to be designed with the design tool. There is no distinction between the model of the artifact to be designed and the model of the tool to design it.

4. EXAMPLE APPLICATIONS

Agentsheets is a substrate to build applications like domain-oriented visual programming systems, and simulation environments. Since the appearance of the first prototype in 1990 about 40 applications have been built using Agentsheets ranging from very simple educational games to very complex design environments. Hence, at the cost of giving an in-depth explanation of one application, it seems appropriate to illustrate the versatility of Agentsheets by briefly describing four different applications.

The spatial organization of agents in a grid and their ability to communicate with each other can be used to simulate the semantics of flow. Individual agents, representing flow propagators (for example, pieces of wire or water pipes), are connected simply by placing them next to each other; that is, connectivity is represented by spatial adjacency. Figures 2 and 3 show applications relying on flow semantics. Users can interact with more complex conductor agents like switches and valves to control the flow.

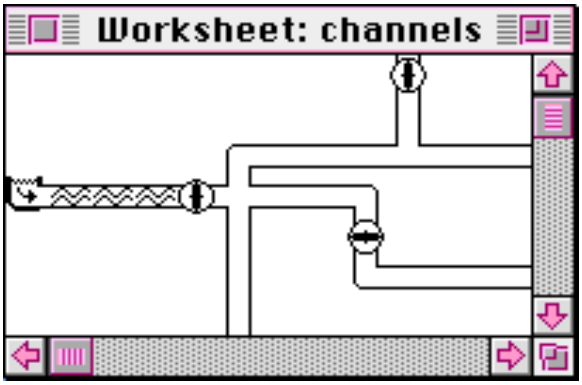

Figure 3. Flow Semantics: Channels

In the case of the Channel system logical relationships and physical relationships (for example Euclidean distance) between components are crucial. In the Petri net application, the position of so-called places and transitions are irrelevant. Agentsheets provides "explicit relationships" objects, such as links, but it is generally suggested that designers minimize their usage since they generally result in a weaker spatial metaphor.

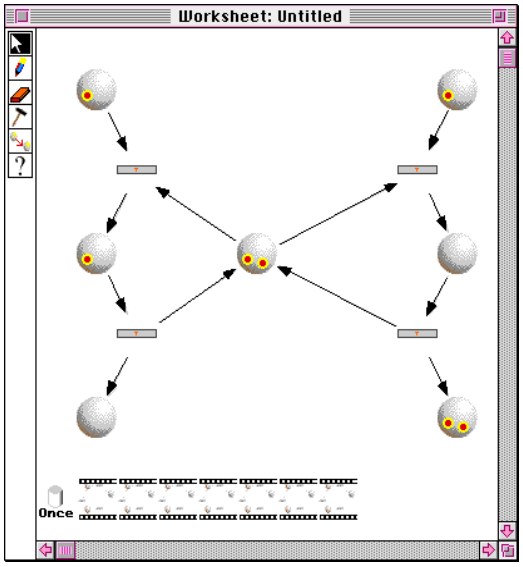

Figure 4. Petri Net

The ability to animate agents (change look, move, play sounds) can be used to implement very different Agentsheets applications. Figure 5 shows an application to simulate ecosystems. End users build environments consisting of mountains, grass tundra, and so on. Then, they set out animals (for example, wolfs, and bear) in those environments. The behavior of the animals is controlled by spatial relations including adjacency and overlap.

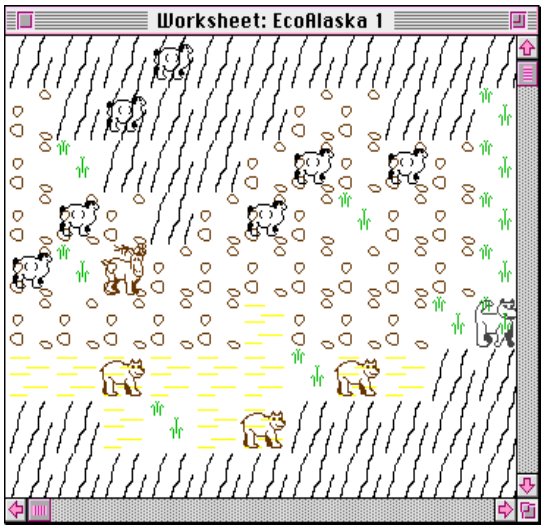

Figure 5. Ecosystem Simulation

The individual patches of the environment and the animals are both agents. End users define simple behaviors for all ecosystem agents. The objective for users is to create ecologically stable environments as well as to get an intuition for the relationships between local behavior and global consequences.

Figure 6 below depicts a more "traditional" visual programming environment based on Agentsheets used to design and run phone-based voice dialog applications [16]. The semantics of voice dialog diagrams are captured by 3 rules: horizontal adjacency or an arrow indicates a temporal relationship; vertical adjacency indicates a choice for the user or by the system. This application includes two different interfaces for two types of users:

- ***the customer interface*** simulates the very limited touch-tone-button-input/voice-output user interface of an ordinary telephone.
- ***the voice dialog designer interface*** shows a voice dialog designer a trace of usage and allows the designer to modify a design, while the system is in use, based on customer's suggestions or problems.

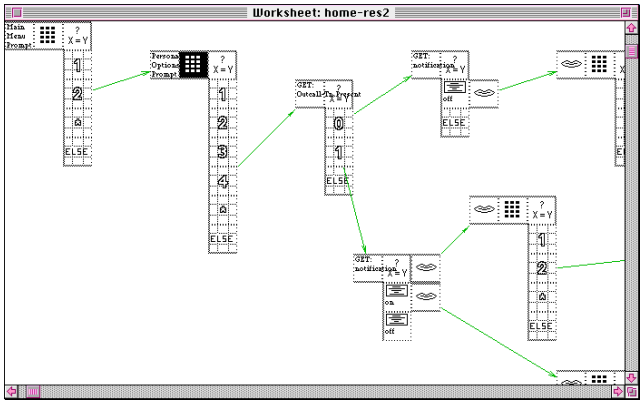

Figure 6. Phone Based Voice Dialog Environment

5. DISCUSSION

One of the central problems in visual programming is to parse pictures. A desirable goal is to find spatial representations that can be parsed efficiently by machines as well as human beings. Often the ease of parsing through humans is achieved by adding very explicit *relationship objects*, such as arrows connecting icons (as seen in data flow diagrams), to spatial representations. To reduce screen clutter these explicit relationship objects can frequently be replaced with implicit topology information. Unconstrained topological information, however, may be hard for humans to parse. A grid structure is one way to disambiguate topological information.

Grids are well known in the area of graphic design, typography, and three-dimensional design. Müller-Brockman [11] characterizes the purpose of grids as follows:

> "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;..."

The main reasons to use grids in Agentsheets are:

- *Avoiding Brittleness*: Without a grid spatial relations can become very brittle. A brittle spatial representation is a representation in which a small change to the visual manifestation of a program on the screen by a user may result in a dramatically different interpretation by the machine. For instance, moving an object (agent) on the screen one pixel may change its spatial relation to an other object in an underlying machine interpretation model from an adjacent relation to a non-adjacent relation. Grids reduce brittleness by discretizing the positions of objects and, therefore, they reduce the chance of mismatches between the interpretation of a picture by a human and the machine.
- *Relational Transparency*: The use of grids increases the transparency of spatial relationships considerably.

- *Implicit Communication*: Communication among agents is accomplished *implicitly* by placing them into the grid. That is, no explicit communication channel between agents needs to be created by the user. In the circuit Agentsheet shown in Figure 1 and Figure 2, the electrical components get “wired-up” simply by placing them in adjacent positions. The individual agents know how to propagate information (flow in this case); for example, the voltage source agent will always propagate flow to the agent immediately below it.
- *Regularity*: Grids also ease the location of such common regular substructures as one dimensional vectors or submatrices.

6. CONCLUSIONS

Agentsheets is a tool for generating iconic programming environments [9]. It is not intended to replace general-purpose visual programming techniques because for most of these established visual representation techniques very elegant implementations already exist. Instead, the spatio-temporal metaphor of Agentsheets supports the creation of domain-oriented programming and simulation environments in a domain-independent way. As such, Agentsheets can be viewed as a generic substrate to quickly prototype and implement new approaches to programming that make use of new, innovative spatial and temporal metaphors.

ACKNOWLEDGMENTS

This research is supported by the National Science Foundation under the grant MDR-9253425, Apple Computer Inc., and US WEST Advanced Technologies. The HCC group at the University of Colorado has provided invaluable and insightful help supporting this work.

REFERENCES

1. T. Bouron, J. Ferber and F. Samuel, "MAGES: A Multi-Agent Testbed for Heterogeneous Agents," *Decentralized A.I. 2*, Saint-Quentin en Yvelines, France, 1991, pp. 195-214.

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

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

4. G. Fischer and C. Rathke, "Knowledge-Based Spreadsheets," *7th National Conference on Artificial Intelligence*, St. Paul, MI, 1988, pp. 1-10.

5. G. W. Furnas, "New Graphical Reasoning Models for Understanding Graphical Interfaces," *Proceedings CHI'91*, New Orleans, Louisiana, 1991, pp. 71-78.

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

7. S. E. Hudson, "An Enhanced Spreadsheet Model for User Interface Specification," *Technical Report*, TR 90-33, University of Arizona, Department of Computer Science, Tucson, AZ, 1990.

8. C. Lewis, "NoPumpG: Creating Interactive Graphics with Spreadsheet Machinery:," *Technical Report*, CU-CS-372-87, Department of Computer Science, University of Colorado at Boulder, Boulder, Colorado 80309-0430, 1987.

9. D. W. McIntyre and E. P. Glinert, "Visual Tools for Generating Iconic Programming Environments," *Proceedings of the 1992 IEEE Workshop on Visual Languages*, Seattle, 1992, pp. 162-168.

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

11. 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.

12. B. A. Myers, "Graphical Techniques in a Spreadsheet for specifying User Interfaces," *Proceedings SIGCHI'91*, New Orleans, LA, 1991, pp. 243-249.

13. K. W. Piersol, "Object Oriented Spreadsheets: The Analytic Spreadsheet Package," *OOPSLA '86*, 1986, pp. 385-390.

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

15. A. Repenning, "Agentsheets: A Tool for Building Domain-Oriented Visual Programming Environments," *INTERCHI '93, Conference on Human Factors in Computing Systems*, Amsterdam, NL, 1993, pp. 142-143.

16. A. Repenning and T. Sumner, "Using Agentsheets to Create a Voice Dialog Design Environment," *Proceedings of the 1992 ACM/SIGAPP Symposium on Applied Computing*, Kansas City, 1992, pp. 1199-1207.

17. B. Shneiderman, "Direct Manipulation: A Step Beyond Programming Languages," in *Human-Computer Interaction: A multidisciplinary approach*, R. M. Baecker and W. A. S. Buxton, Ed., Morgan Kaufmann Publishers, INC. 95 First Street, Los Altos, CA 94022, Toronto, 1989, pp. 461-467.

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

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

20. T. Toffoli and N. Margolus, *Cellular Automata Machines*, MIT Press, Cambridge, Massachusetts, 1987.

21. N. Wilde and C. Lewis, "Spreadsheet-based interactive graphics: from prototype to tool," *Proceedings CHI'90*, Seattle, WA., 1990, pp. 153-159.