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

Alex Repenning

CU-CS-547-91 September 1991

University of Colorado at Boulder
DEPARTMENT OF COMPUTER SCIENCE

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

Alex Repenning

Department of Computer Science and Institute of Cognitive Science
Campus Box 430
University of Colorado, Boulder CO 80309
303 492-1218, ralex@boulder.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 employes the notion of agents organized in a grid. The *spatial relationships* between agents are used to capture design properties independent of domain and programming language. Two types of spatial relations are distinguished called *strict-spatial relations* and *pseudo-spatial relations*. This paper gives a short introduction to Agentsheets, explains how Agentsheets address problems of construction kits, sketches sample applications, and evaluates the contribution of grid-based spatial reasoning to human-computer interaction.

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

1. Introduction

The design and implementation of human-computer interfaces is, no doubt, a verifiably hard task. Construction kits have been shown to be effective tools for human-computer interaction [5]. 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, i.e., they view integrated circuits as abstractions of compositions of simpler constituents.

Visual programming systems (VP), on the other hand, are supposed to help users to program computers by capitalizing on human spatial reasoning skills [3, 21]. 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 level of abstraction represented by the building blocks:

- *Construction Kits: High Level Building Blocks* provide powerful abstractions but are quite likely domain specific and therefore not applicable to a broad palette of different applications.
- *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 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, e.g., boxes representing procedures, functions, etc. These systems typically add only little value to their textual counter parts.

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 step between a building block level and the level of a conventional programming language used to implement the building-blocks is called the "Representation Cliff" [20]. 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 (e.g., a library consisting of a large set of functions), the behavior, and look of artifacts.

Agentsheets 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 (Figure 1). 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.

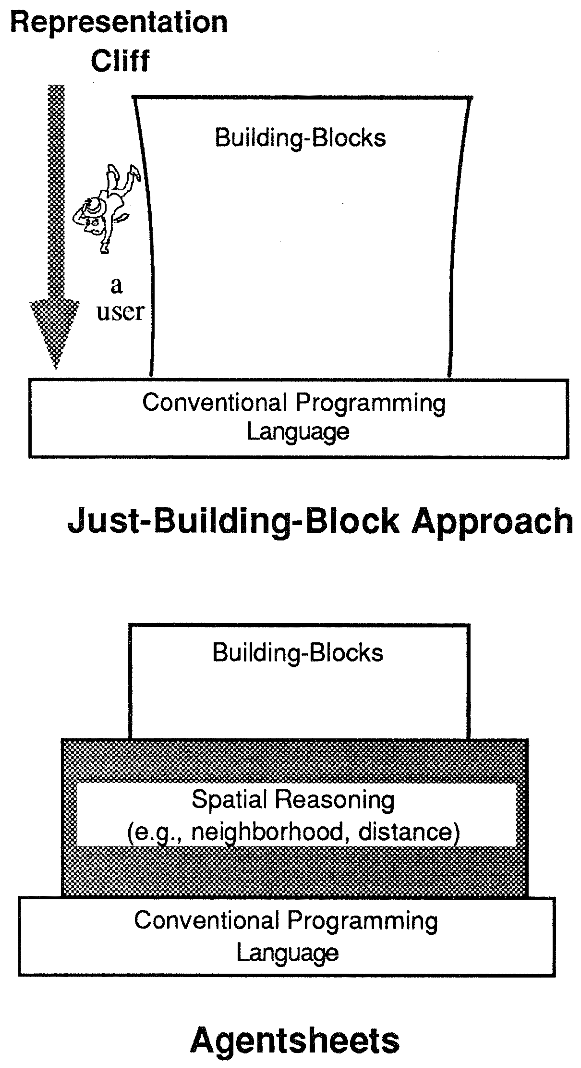
Representation Cliff
Building-Blocks
a user
Conventional Programming Language
Just-Building-Block Approach
Building-Blocks
Spatial Reasoning (e.g., neighborhood, distance)
Conventional Programming Language
Agentsheets

Figure 1. Spatial Reasoning Level

The spatial reasoning level employed by Agentsheets supports the design of graphical user interfaces by being:

- domain independent, and
- programming language independent.

Agentsheets is a graphical system builder. In a typical application of Agentsheets a designer will define the look and behavior of high level building blocks and thus make use of the spatial reasoning level. These building blocks constitute the elements of a construction kit which can be used readily by end users.

2. Types of Spatial Relations

We distinguish between two types of spatial relationships:

- *Strict-Spatial Relation*: A strict-spatial relation between objects is apparent by the actual positions of the objects. For instance in Figure 2 the roof and the frame of the house describe an **implicit** above relationship.

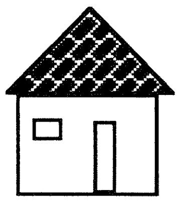

Figure 2. Strict-spatially related roof and frame of a house

- *Pseudo-Spatial Relation*: A pseudo-spatial relation between objects is independent of the actual positions of the objects. Instead **explicit** clues are used to represent a relation. In Figure 3 arrows are employed as explicit clues of the roof frame relationship. Although, there are strict-spatial relations between the roof and the arrow and the arrow and the frame the spatial relation between the roof and the frame is non-strict. That is, the relative position of the roof and frame are completely irrelevant.

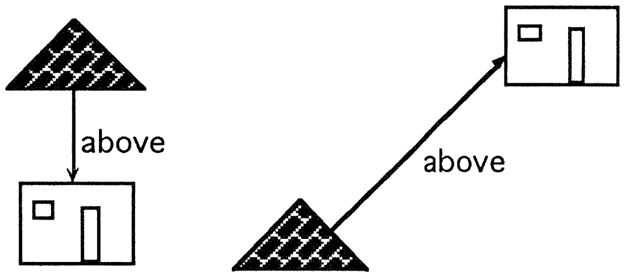

Figure 3. Pseudo-spatially related roofs and frames of houses

The necessity for explicit clues in pseudo-spatial relations can easily lead to cluttered, interconnected diagrammatic representations. Even in relatively simple diagrams these clues readily outnumber the related objects. As long as these clues represent highly abstract relationships without evident spatial meanings then their use might be unavoidable.

Some of the limitations of visual programming system result from their use of pseudo-spatial relations. Pseudo-spatial relations are chosen because of their general purpose nature. Brooks states his skepticism regarding visual programming as follows [2]:

> "A favorite subject for PhD dissertations in software engineering is graphical, or visual, programming - the application of computer graphics to software design. Sometimes the promise held out by such an approach is postulated by analogy with VLSI chip design, in which computer graphics plays so fruitful a role. Sometimes the theorist justifies the approach by considering flowcharts as the ideal program-design medium and by providing powerful facilities for constructing them.
>
> Nothing even convincing, much less exciting, has yet emerged from such efforts. I am persuaded that nothing will."

The components in a chip 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. The positions of flow-chart elements, in contrast, have no meaning. Infinitely many valid arrangements of the same set of elements representing the same topology might therefore range in their readability from excellent to chaotic without implying different semantics.

Strict-spatial relations appear to be preferable to pseudo-spatial relations in cases where the positional information of objects can be mapped to physically natural concepts understood by users (e.g., time). Agentsheets encourage the use of strict-spatial relations. That is, the meaning of an artifact composed by a user is defined by the position of its constituents.

3. The Agentsheets System

3.1. Agents and Agentsheets

The basic components of Agentsheets are agents [8, 14]. An agent is a thing (or person) empowered to act for a client. It 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. Figure 4 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 pieces are captured by agents.

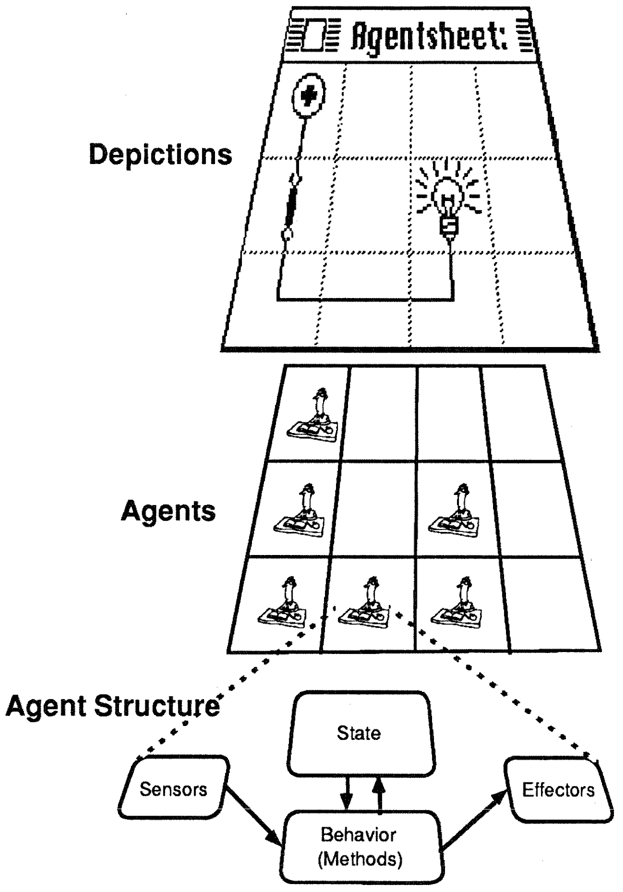

Figure 4. The Structure of an Agentsheet

The *depictions* in Figure 4 show the graphical representation of an Agentsheet as it is seen by an user. Each depiction represents the class of an agent, e.g., the symbol of an electrical switch denotes a switch agent. Furthermore, different states of an agent 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* of the agent. They are actively triggered by the user (e.g., clicking at an agent).
- *Effectors.* A mechanism to communicate with other agents by sending messages to agents either using grid coordinates (strict-spatial relations) or explicit links (pseudo-spatial relations). 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 [22].

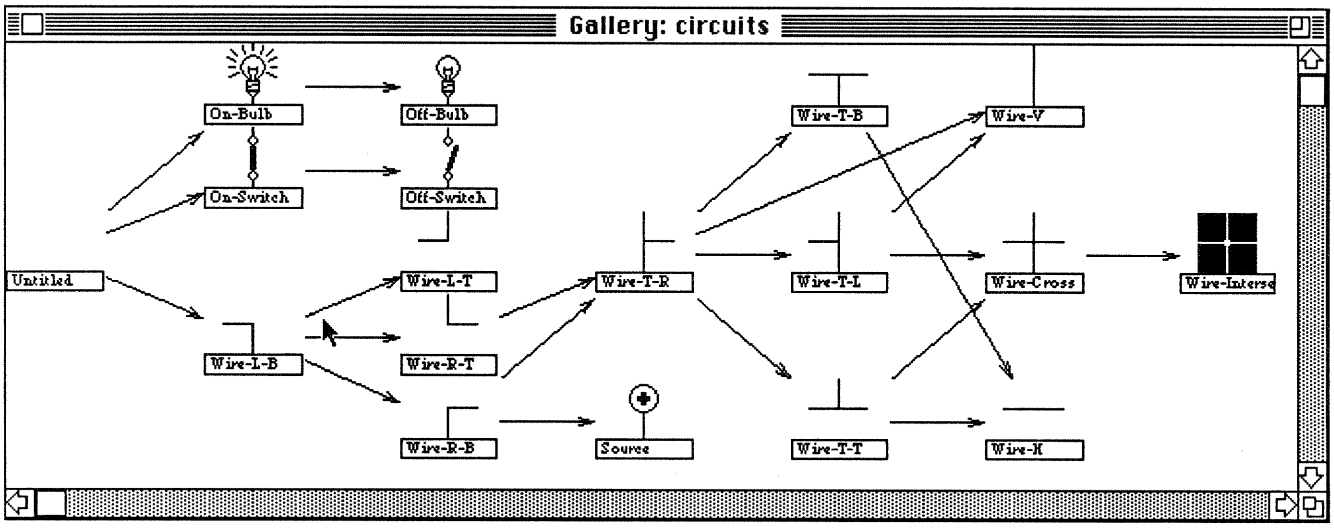

Figure 5. A Wire Gallery

- *State.* Describes the condition the agent is in.
- *Depiction.* The graphical representation of the class and state, i.e., the *look* of the agent.

3.2. Galleries

The depictions of agents are defined in the so called gallery. Depictions are defined incrementally by cloning existing depictions. In the below gallery only 5 depictions have been drawn by the designer. All remaining depictions have been created through cloning.

4. Related Work

Agentsheets are highly related to cellular automata (CA) [23]. 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 agents instead of simple cells. These Agents have a large set of sensors allowing them not only to perceive the state of their neighboring agents but also to react to user events (e.g., an 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 [7]. Like CAs, BITPICT operates on the pixel level.

Spreadsheets have shown to be powerful tools because they adopted an interaction format people were already familiar with [6]. Furthermore, the cell addressing scheme is equivalent to strict-spatial relations. Many extensions to spreadsheets have been proposed to increase their usability even more. Piersol suggested the use of object-oriented techniques for spreadsheets [18]. 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 [10, 12, 16, 25]. Agentsheets go one step further unifying the graphical user interfaces to be designed with the design tool. That is, cells in Agentsheets do not just **refer to** attributes of spatial representations (e.g., positions), instead they **contain** the spatial representation. There is no distinction between the artifact to be designed and the tool to design it.

5. Sample Applications

Agentsheets support strict-spatial as well as pseudo-spatial relations. In the following several application are shown to elaborate on the applicability of different spatial relation types. The river modeling system and the voice dialog designer have been tested by several users. An in-depth description, including empirical user testing, of an Agentsheet-based application utilized to create an user interface for a commercial expert system shell can be found in [20].

5.1. Pipes

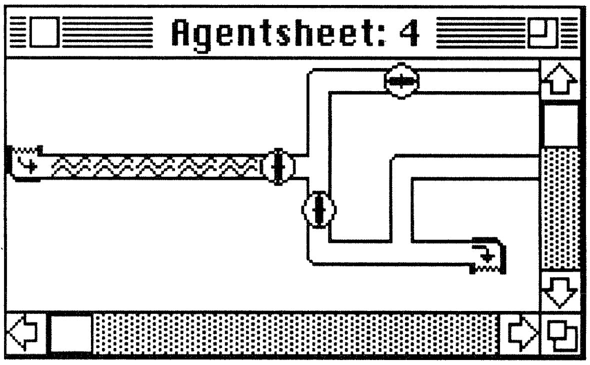

Figure 6. Pipes and Valves

Agents represent:

- *Pipes* to propagate water
- *Valves* to control flow
- *Sources and Sinks* generating or absorbing water

This simple model simulates water distribution in pipe system. Strict-spatial relations are used to model physical phenomena. For instance, the loss of water due to leaks or evaporation is modeled on the level of individual water pipes. A sequence of 10 pipe agents having 1% loss of water each will lead to an water output of about 90%. This behavior is specified for a straight horizontal pipe agent by a sensor definition:

```
SENSOR Value-From-Left OF Pipe-Agent (Value)
    "A pipe agents looks like:  "
    if Value > 0.0 then
        effect (0, 0, depiction, filled);
    else
        effect (0, 0, depiction, empty);
    effect (0, 1, value-from-left, Value * 0.99);
```

Activating the VALUE-FROM-LEFT sensor of a PIPE-AGENT will result in changing the current depiction of the pipe agent to either a filled or empty pipe. Then the agent computes the water loss and propagates the remaining value of water to its right neighbor. The effect statements enclosing relative grid positions are used to communicate with other agents, i.e., activate their sensors.

The coordinates are relative and increasing towards right and down. A reference to itself is (0, 0), a reference to the immediate neighbor one row down and one column to the right is (1, 1). Given this addressing scheme hooking pieces of pipes up comes for free. That is, pipes are connected by simple placing them next to each other.

The position of pipe agents on the screen manifest actual distances. By placing pipes into the agentsheet a user implicitly defines the distances and hence the water loss between any two points in the system. A system based on pseudo-spatial relations would require the user to explicitly specify the distances between individual points. This can be quite tedious as the geometry of pipes might be complex.

5.2. River Modelling

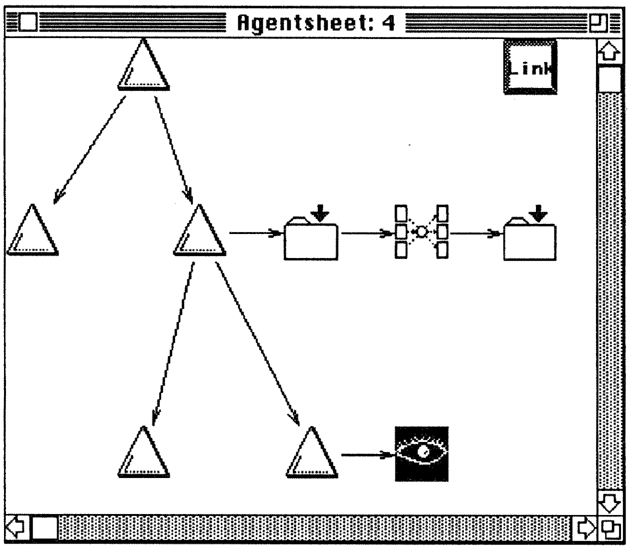

Figure 7. River System

Agents represent:

- *Reservoirs* having inflow and outflow
- *Data Sinks and Sources*, e.g., databases containing statistical rain fall data
- *Computational Units* massaging numerical information
- *Buttons* to create links and to start/stop the simulation
- *Guards* watching critical values

Links represent:

- *Water flow*
- *Information flow*, e.g., the level of a reservoir being sent to a data base

Discrete event simulation [24] is used to verify what-if scenarios in large scale water distribution system. Constraints can be attached to reservoirs (e.g., regarding their tolerated water level) to study the impact of different water distribution schemes. This system is described by Reitsma [19].

The simulation neglects physical effects like evaporation between reservoirs. Hence pseudo-spatial relations turned out to be sufficient. Positions of reservoirs on the screen are completely irrelevant and do not reflect actual positions. This Agentsheet application is therefore on the other extreme of the strict/pseudo spatial relation spectrum compared to the pipe simulation.

5.3. Voice Dialog Designer

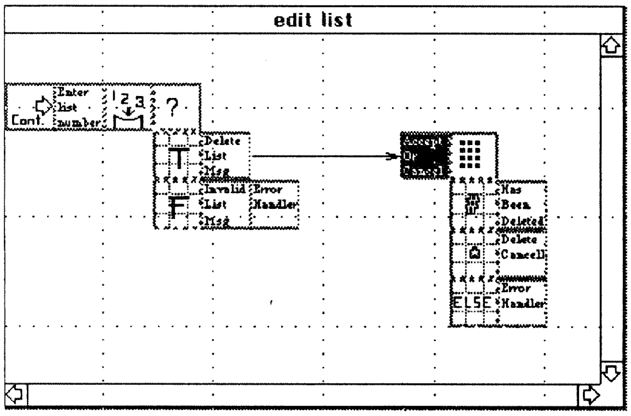

Figure 8. A Voice Dialog Specification

This system is used to design and simulate telephone voice systems. Agents represent user interaction like voice output and telephone keyboard input or nested voice dialog subsystems. This prototype started off using pseudo-spatial relations exclusively. The pseudo-spatial relations mimicked the existing paper-and-pencil design documents. As a result the prototype also inherited the limited readability of the original design documents. In a second attempt the concept of time and choice got mapped to strict-spatial relations leading to a much better structured and denser graphical representation. Only high level relationships remained in their pseudo-spatial relation form, e.g., the link in Figure 8. The emerging clusters in the diagram improved readability significantly.

5.4. Agentsheets

Self applicability of a system is a good indicator for its usability. Also, reducing the number of concepts in a system will increase *simplicity* as well as *uniformity*, which will amplify the usability of a system [1].

The first application implemented using Agentsheets has been Agentsheets:

- *The Gallery* is an Agentsheet (Figure 5). Agents represent items of a palette. Links represent cloning relationships among these items.
- The Bitmap Editor is an Agentsheet (Figure 9). A pixel agent flips its color by clicking at it.

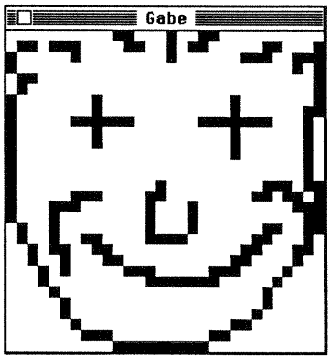

Figure 9. Bitmap Editor

All agents represent pixels.

6. Discussion

The rigid grid structure is one of Agentsheets' main strengths. Some application domains might be hard to be laid out in a grid structure. However, the design objective of Agentsheets is to support a limited set of domains elegantly rather than providing a very general but low-level tool.

6.1. Spatial Metaphors

Even if a spatial representation of a application domain is non-obvious it might be worth while to think of new ways of the problem. Ehn states [4]:

"The spatial metaphor can be used to create completely new concrete tools for tasks that earlier have been purely formal or abstract."

Hence, instead of relying on general purpose visual programming tools supporting only pseudo-spatial relations, the use of a graphical system builder helps to create high level, domain specific tools. These tools may improve human-computer interaction by employing new powerful spatial metaphors.

6.2. Why Grids?

Agentsheets use grids for reasons of visual and conceptual integrity [13]. Grids are well known in the area of graphic design, typography, and three dimensional design. Müller-Brockman characterizes the purpose of grids as follows [15]:

"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 for grids are:

- *Brittleness*: Without a grid spatial relations can become very **brittle**. That is, moving an object (agent) on the screen one pixel may change its spatial relation to an other object from a adjacent relation to a non-adjacent relation. While this might reflect the intention of an user it is more likely to lead to non evident problems. In this respect grids offer an abstraction of details related to positions of objects.

- *Relational Transparency*: The use of grids increases the transparency of spatial relationships considerably. The strict-spatial relationships of object in a grid are obvious to the user.
- *Implicit Communication*: Communication among agents is accomplished *implicitly* by placing them into the grid. That is, no explicit communication channel between agents has to be created by the user. In the circuit Agentsheet shown in Figure 4, the electrical components get "wired-up" simply by placing them to 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.
- *Regularity*: Grids also ease the location of common regular substructures like one dimensional vectors or sub matrices.

7. Conclusions

Agentsheets are not meant to replace general purpose visual programming techniques like data flow [11], flow charts like system [9], Nassi-Shneiderman structure diagrams [17] etc, because for most of these established visual representation techniques very elegant implementations already exist. Instead, Agentsheets provide a method to build domain specific construction kits without forcing the kit designer to adapt to preconceived notions introduced in current visual programming systems. The kit designer maps domain specific concepts to spatial relationships. These spatial relationships can be employed, but are not limited to, represent existing visual representations techniques. In other words, the designer has the freedom to introduce new, non-traditional, means of visual representations without the need to implement his "custom semantics" construction kit on the low level of abstraction provided by conventional programming languages.

Acknowledgements

The HCI and the AI Research groups at the University of Colorado have provided invaluable and insightful help supporting this work. This research was funded by the United States Bureau of Reclamation through the Advanced Decision Support program.

References

[1] A. Borning and T. O'Shea, "Deltatalk: An Empirically and Aesthetically Motivated Simplification of the Smalltalk-80 Language," *ECOOP '87*, 1987, pp. .

[2] F. P. Brooks Jr., "No Silver Bullet: Essence and Accidents of Software Engineering," *IEEE Computer*, , pp. 10-19, 1987.

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

[4] P. Ehn, *Work-Oriented Design of Conputer Artifacts*, arbetslivscentrum, Stockholm, 1989.

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

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

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

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

[9] E. P. Glinert and S. L. Tanimoto, "Pict: An Interactive Graphical Programming Environment," *IEEE Computer*, , pp. 265-283, 1984.

[10] 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.

[11] D. Ingalls, S. Wallace, Y.-Y. Chow, F. Ludolph and K. Doyle, "Fabrik: A Visual Programming Environment," *OOPSLA '88*, San Diego, CA, 1988, pp. 176-190.

[12] 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.

[13] A. Marcus, "Proportion and Grids: Keys to Successful Layout," *Computer Graphics Today*, Vol. 3, pp. 1986.

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

[15] 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.

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

[17] I. Nassi and B. Shneiderman, "Flowchart Techniques for Structured Programming," *SIGPLAN NOTICES*, , pp. 72-78, 1973.

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

[19] R. Reitsma and J. Behrens, "Integrated Basin Manager (IRBM): A Decision Support Approach," *Proceedings of the 2nd international conference on Computers in Urban Planning and Urban Management*, Oxford, 1991, pp. .

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

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

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

[23] T. Toffoli and N. Margolus, *Cellular Automata Machines*, MIT Press, Cambridge, Massachusets, 1987.

[24] P. S. van der Meulen, "INSIST: Interactive Simulation in Smalltalk," *OOPSLA '87*, Orlando, FL, 1987, pp. 366-376.

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