Repenning, A. and T. Sumner, "Programming as Problem Solving: A Participatory Theater Approach," *Workshop on Advanced Visual Interfaces '94*, Bari, Italy, 1994, pp. 182-191.

Reprint:

Programming as Problem Solving: A Participatory Theater Approach

Alex Repenning
Tamara Sumner

Department of Computer Science and Institute of Cognitive Science
Campus Box 430
University of Colorado, Boulder CO 80309
492-1349, ralex@cs.colorado.edu, sumner@cs.colorado.edu
Fax: (303) 492-2844

Keywords:

Delegation interfaces, direct manipulation, participatory theater, problem solving, visual programming, opportunistic design, spatial metaphors, temporal metaphors, agents, agentsheets, construction kits, human-computer interaction, object-oriented programming

Acknowledgments

We thank the HCC group at the University of Colorado and Clayton Lewis, who contributed to the conceptual framework and the systems discussed in this paper. We wish to particularly thank Jim Sullivan, the creator of the EcoWorlds applications. We also wish to thank Jonathan Ostwald, Kumiyo Nakakoji, and Gerry Stahl for their comments on early drafts of this paper. This research was supported by: the National Science Foundation under grant No. MDR-9253425, Apple Computer Inc., and US West Advanced Technologies.

Programming as Problem Solving: A Participatory Theater Approach

Alex Repenning
Tamara Sumner

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

Abstract

Spatial and temporal metaphors can play an important role in making the task of programming serve problem-solving processes. Visual programming research hopes to capitalize on innate human perceptual skills to make the programming task easier by using visualization to simplify program construction at the syntactic level. Instead, we advocate that the role of visualizations, and the consequent use of spatial metaphors, is not to simplify programming per se but instead to *support the problem-solving process*. To that end, environments endorsing spatial metaphors should support: creating and changing external representations of the problem, and opportunistic design strategies necessary for exploring problem spaces. We discuss problems with human-computer interaction schemes arising from the use of temporal metaphors. Direct-manipulation, on the one hand, can be too direct for controlling a number of autonomous processes such as cooperating agents. The complete delegation of tasks to agents, on the other hand, can leave users entirely in the role of passive observers. We propose a new approach, called the *participatory theater metaphor*, which combines the advantages of human computer interaction schemes based on direction manipulation and delegation and provides users with a continuous spectrum of control over their program behaviors.

KEYWORDS: Delegation interfaces, direct manipulation, participatory theater, problem solving, visual programming, opportunistic design, spatial metaphors, temporal metaphors, agents, human-computer interaction

1. Introduction

The initial perception of visual programming was that the visualization of syntactic structures would capitalize on innate human perceptual skills to make the programming task easier. Early visual languages such as BLOX Pascal [5] focused on providing visual representations of general purpose programming languages (Figure 1). In BLOX Pascal, the shapes of the building blocks help users make correct syntactic constructions but users still need to understand the Pascal language in order to write meaningful programs.

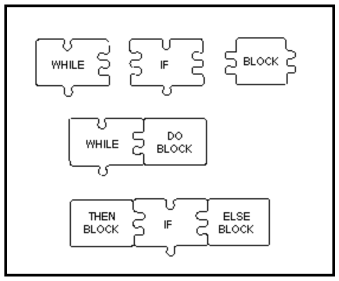


Figure 1. The shapes of the blocks facilitate making correct syntactic constructions.

Other researchers have long advocated that the role of visualization is to aid problem-solving. Simon states that solving a problem means representing it so as to make the solution transparent [20]. In this paper, we advocate that the role of visualization is not to serve programming per se, but to support problem-solving (i.e., problem representation) activities.

However, simply providing visual representations is insufficient to adequately support the problem-solving process. Schön has studied the complex problem-solving process of design [19]. He concludes that problems can only be understood through repeated attempts to solve them. Fischer [3] has described this aspect of design as "a dialectic between problem framing and problem solving." Green also adopts the stance that "design is redesign" and concludes that representations need to be easily modifiable to support opportunistic planning and redesign processes [6]. Thus, being able to modify representations is an essential property of a good visual problem-solving environment. However, many visual programming languages provide a single built-in representation of their underlying programming environment which users cannot change.

Instead of creating visual programming languages, we advocate the creation of visual problem-solving environments. Such environments combine the accepted wisdom of visual representations to support problem-solving with the added potential benefits of a programming language. Programming can aid problem-solving by transcending the limits of static visual representations and supporting users to explore different aspects of their problem. Many problem domains have dynamic aspects that cannot be represented statically since their importance is in understanding how these aspects change over time.

One challenge for these environments is to provide users with new human-computer interaction schemes that allow users to interact with and control the behaviors of these dynamic problem representations. Interacting with programs can be considered along a spectrum of control where one extreme provides the user with maximal control over behavior in the form of direct manipulation interfaces; the other end provides users with the least control in the form of complete delegation interfaces.

We explore this spectrum in terms of a theater metaphor. In this metaphor, direct manipulation interfaces correspond to providing users with hand puppets. Users have complete control over the actions of these puppets but are required to do everything for themselves. Users interacting with delegation interfaces correspond to audiences passively watching a staged production where they have no control over the course of events. Neither of these extreme views of human-computer interaction are sufficient for the visual problem-solving environments advocated in this paper. We suggest a new human-computer interaction metaphor called “participatory theater” where users interact with the actors in the production; i.e. the dynamic objects in the visual representation.

The two notions of programming as problem-solving and human-computer interaction as participatory theater are manifested in the Agentsheets system and visual problem-solving environments built using this system. In this paper, we enumerate five principles systems should embody to support this view of programming as problem-solving. Systems should:

1) Support the creation of spatial metaphors.

2) Support the creation of temporal metaphors that are intrinsically integrated with a spatial metaphor.

3) Support users in modifying both of these metaphors.

4) Provide users with a wide spectrum of control over the dynamic behaviors in their solution representation.

5) Provide symmetric communication facilities for both users and dynamic objects.

First we will discuss the theoretical background underlying the two notions of “programming as problem-solving” and “human-computer interaction as participatory theater” and we will derive the five principles. Next, the Agentsheets system will be described. The core of this paper will illustrate each of the five principles using three problem-solving environments built using the Agentsheets system – the Voice Dialog Design Environment, Eco Worlds, and Kitchen Planner.

2. Programming as Problem Solving

The more traditional view of programming environments as implementation facilitating devices assumes that a problem is sufficiently well understood such that the process of programming is a mapping of this problem understanding onto a programming mechanism [7]. However, adopting the “programming as problem solving” perception leads to a programming substrate of a different nature that focuses on facilitating the problem solving process rather than an “implementation as mapping” process.

The process of problem solving includes creating and changing representations until the solution becomes visible [20]. Several studies [11] show the important role visually inspectable models play in problem-solving processes. In these studies, problem-solving proceeded by creating an external representation of the solution, visually inspecting this solution and noticing problematic aspects, and repairing the problematic areas by altering the proposed solution. In many cases, representations become transparent when they express information in ways analogous to familiar situations. In other words, representations can serve as physical manifestations of metaphors. Metaphors help us to learn about things we do not yet understand by creating analogies to things we do understand [9].

Principle 1: Problem-solving environments should incorporate spatial metaphors to support the creation of inspectable external representations.

However, many aspects of a problem cannot be represented in a static visual representation. In some domains, such as ecosystem simulation, the important aspects of the problem are how the system changes over time. Expressing such problems requires temporal metaphors in addition to spatial metaphors. Computers and programming languages enable the creation of temporal metaphors. We define a program to be a collection of actions or behaviors arranged in a temporal metaphor. Programming can enable users to express aspects of the problem that cannot be represented in static visual representations.

Principle 2: To fully support problem-solving and take advantage of human visual perceptual skills, temporal metaphors should be intrinsically tied to a corresponding spatial metaphors. Such a binding transforms dead paper representations into active media supporting problem-solving.

Opportunistic design processes are important for problem-solving [6] and can be viewed as a design strategy supporting a dialectic between problem framing and problem solving. The evolving solution serves as an inspectable model of the partial solution and guides the designers in the problem-solving process. In Nardi's study of spreadsheet users [11], she observed that the strength of the spreadsheet as a problems-solving environment was how it supported both the incremental evolution and radical restructuring of an inspectable model of the emerging solution.

Principle 3: Problem solving systems should support opportunistic design practices by providing metaphors that can be both incrementally modified and radically restructured.

3. HCI as Participatory Theater: The Art of Program Invocation

Programming as problem solving involves not only the problem of creating a program, but also the problem of knowing when and how to invoke the program. Program invocation takes on a new importance because in our model, programs are no longer monolithic entities but instead are collections of cooperating behaviors arranged in spatial and temporal metaphors.

Human-computer interaction, from a technical point of view, can be understood as the art of invoking the right program fragment at the right time in order to give users appropriate levels of *control* over their program behaviors. We will use a theatrical metaphor to illustrate the possible spectrum of user control and to compare several human-computer interaction schemes

Direct manipulation and *delegation* illustrate two extreme end points on the continuous spectrum of control. Control can be understood in terms of who takes the initiative – the user or the system. With direct manipulation, users have maximal control over the individual components of their system. In terms of the theatrical metaphor, direct manipulation interfaces are like hand puppets in the sense that users are completely in charge of the staged production (Figure 2).

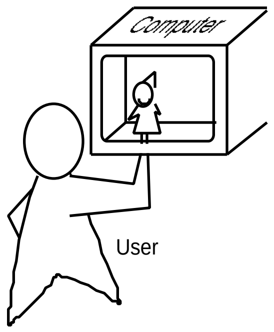


Figure 2. Direct Manipulation: Puppets are controlled by the user.

The extreme level of control provided by direct manipulation can be inappropriate for applications modeling large numbers of autonomous or cooperating entities [12]. Negroponte suggests the theatrical metaphor of actors in a production. In this approach, tasks are completely delegated to actors. This approach illustrates the opposite end of the control spectrum since once the actors' scripts have been created and the play has started, the audience is left with no control over the course of the production (Figure 3). However, the actors in the play can do more sophisticated tasks than the hand puppets since their activity is not bounded by the inherent sequentiality of single user actions found in direct manipulation schemes.

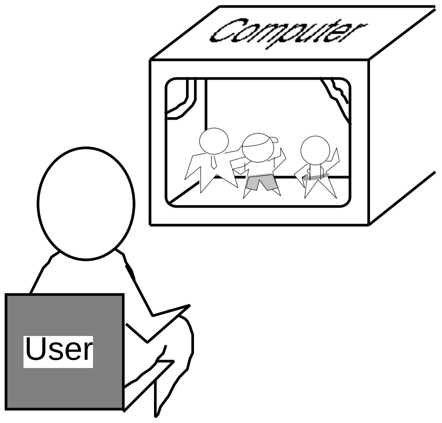


Figure 3. Complete Delegation: Passive audiences observe actors following a script.

Traditional simulation environments rely on the "actors in a play" model of interaction. The definition of simulation attributes is similar to writing the script for an actor. After the simulation environment has been prepared and the simulation has been started, the user of the system becomes a passive observer watching the progress of the simulation.

We suggest that the virtues of both approaches be combined in a *participatory theater metaphor* (Figure 4) by perceiving *control as a spectrum* rather than the discrete dichotomy of direct manipulation versus delegation. Actors in a participatory theater will act according to their script unless the audience tells them to do something differently.

Depending on the level of participation, the interaction can assume any point in the control spectrum. If the users choose not to participate, then they have no control over the production and become passive observers. Excessive participation will result in the user taking over the production completely to the point of direct manipulation.

Temporal metaphors provide a framework for organizing collections of cooperating actors into behavioral representations. Problem-solving with behavioral representations means "experiencing" behavior, interacting with it, and directing it to explore possibilities and what-if scenarios.

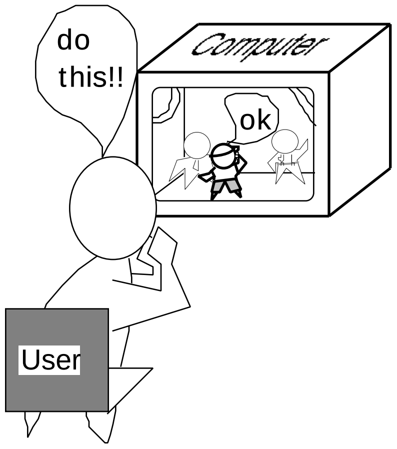


Figure 4. Participatory Theater: Both users and actors influence the course of the production.

Principle 4: Participatory theater interaction requires that users have a wide spectrum of control over their program behaviors. User should be able to interact with each individual actor in the collection (production) as well as the collection as a whole.

Actors in the production correspond to objects whose behavior is choreographed by a script representing the temporal metaphor. The user is just another participant in the production. Any participant in the production can guide or influence the course of action. As such, users and actors (objects) must be provided with the same communication facilities so that they can interchangeably communicate with each other.

Principle 5: Symmetrical communication allows actors to respond to sensory input the same way regardless of whether it originated from another actor or the user.

This symmetry allows users to interject an action into a production or shift the script's direction without having to stop the overall process. Thus in the participatory theater approach, there need not be an explicit run-interact-stop cycle of program execution.

4. Architecture of Agentsheets

Agentsheets is a substrate for constructing domain-oriented, dynamic, visual problem-solving environments. It supports the creation and animation of a variety of graphical representations. The environments created consist of cooperating agents organized in a grid. The grid structure is used to clarify spatial relationships between agents such as adjacency, relative and absolute position, distance, and orientation. Together, the notion of agents and grids can be used to create temporal and spatial metaphors. A comparison of Agentsheets to other systems can be found in [15]. Agentsheets is built on top of an object-oriented system and is written in Common Lisp.

In a typical application of Agentsheets, a system designer defines the look and behavior of domain-specific agents. These agents constitute the elements of a high-level visual programming language which can be used readily by end users. End users arrange these agents in a worksheet. The worksheet has an underlying grid structure analogous to the rows and columns in a spreadsheet. Relationships between agents can be explicitly specified by connecting them with links or implicitly specified simply by position within the grid structure. Each agent has an associated behavioral component which allows the agent to perform an action in response to some stimulus. The stimulus can come from the user or from another agent.

4.1. Agents and Agentsheets

The basic components of Agentsheets are agents [4, 10]. 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 worksheet. Every agent has a graphical depiction that is visible in the Agentsheet. Figure 5 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.

The *depictions* in Figure 5 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 look of the depictions plus rules guiding their arrangement in the agentsheet grid define the spatial metaphor.

Depictions of agents are stored in the gallery and defined by using the provided bitmap editor or by modifying and combining existing depictions. When depictions are placed in the agentsheet, each depiction gets bound to an agent. In the theater metaphor, agents correspond to actors in the production. These agents 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 being effected. Effectors can also be used to modify the agent's depiction or to play sounds.
- ***Behavior***: The built-in agent classes provide default *behaviors* defining reactions to all sensors. In order to refine this behavior, methods associated with sensors can be shadowed or extended making use of object-oriented mechanisms.

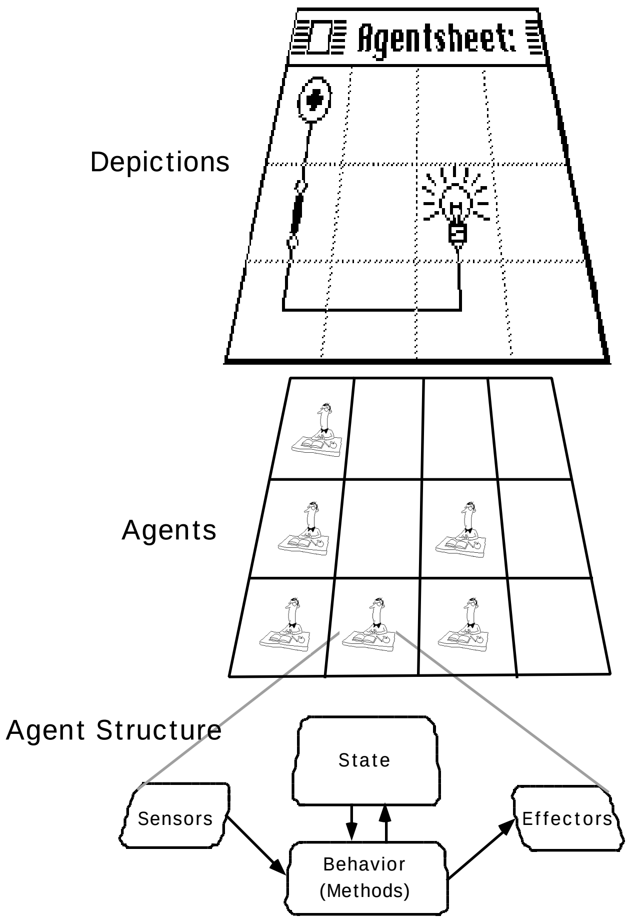


Figure 5. The Structure of an Agentsheet.

The sensors, effectors, and behavior of an agent determine how a particular agent reacts to its environment. The temporal metaphor is created by specifying how classes of agents communicate via sensors and effectors. The temporal metaphor should be closely tied to the spatial metaphor such that when depictions are arranged according to the spatial metaphor, the defined agent communication paths will be enabled simply by virtue of the agent's placement within the grid structure. For instance in the electric world (Figure 5), the spatial and temporal metaphors are integrated around the concept of "flow." Flow has a spatial manifestation based on adjacency in the worksheet. Adjacency, in turn, defines the temporal sequence guiding the flow of electricity through the circuit.

4.2. Using Agentsheets

Agentsheets anticipates two types of users: a system designer and an end user. End users build programs with the resulting system by selecting familiar objects (i.e., agents) from a gallery and placing them into a worksheet. The layout of objects in the worksheet defines the program's collective meaning.

In a typical application scenario, a system designer uses Agentsheets to create a high-level visual problem-solving environment. These environments are essentially construction kits 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 AgenTalk, which is an extension of 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.

5. Three Agentsheet Applications

In this section, we will present three applications that have been constructed using the Agentsheets substrate – the VDDE system (a design environment supporting the design and construction of phone-based interfaces), EcoWorlds (a family of ecosystem simulation applications), and Kitchen Planner (a kitchen floor plan construction system). For each system, we will: 1) briefly describe the domain and the problem the system is designed to address, 2) provide an overview of the specific system; and, 3) discuss how the application and the Agentsheets substrate fulfill our five principles. These three applications were chosen to illustrate the diversity of environments that can be constructed using the Agentsheets substrate. The first application, the VDDE system, shows how Agentsheets can be used to create active or behaving design representations. The second application, EcoWorlds, illustrates the participatory theater approach. The third application, Kitchen Planner, shows how taking the participatory theater approach to its extreme can turn design into a tactile experience.

5.1. The Voice Dialog Design Environment

Voice dialog applications are phone-based systems; prototypical applications are voice mail systems and voice information systems. These systems consists of a series of voice prompts requesting the user to perform certain actions; e.g. “to listen to your messages, press 1.” Traditionally, the interfaces of such systems have relied on audio recordings and the sounds emitted by the touch-tone buttons on a common telephone.

Designing in this domain means specifying the interface for a voice dialog application at a detailed level. Voice dialog interfaces consist of a cascading series of voice prompted menus which the user must navigate through. Objects such as voice menus and audio prompts are arranged in special visual representations similar to flow charts that convey the flow of control and range of allowable user actions within a particular phone-based interface. The design process of voice dialog applications is complicated by the "medium gap" between the visual design representations and the audio end product. In phone-based interfaces, information is presented to the user auditorally as spoken messages and signals. It is difficult for the designer working with a visually-oriented design representation to mentally bridge this medium gap and envision the auditory end product.

The voice dialog design environment (VDDE) [16] provides an on-screen gallery of voice dialog design units, such as menus and prompts, and a worksheet for design construction and simulation. The design of a small restaurant pizza-ordering system constructed in this environment is illustrated in Figure 6. Using the voice dialog design environment, designers work with meaningful domain abstractions such as prompts and menu design units. “Designing” involves placing design units into the worksheet in accordance with three design unit placement rules. Together, the look of the domain-oriented depictions and the three placement rules define a spatial metaphor based on traditional voice dialog design representations. The three placement rules are:

- ***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.

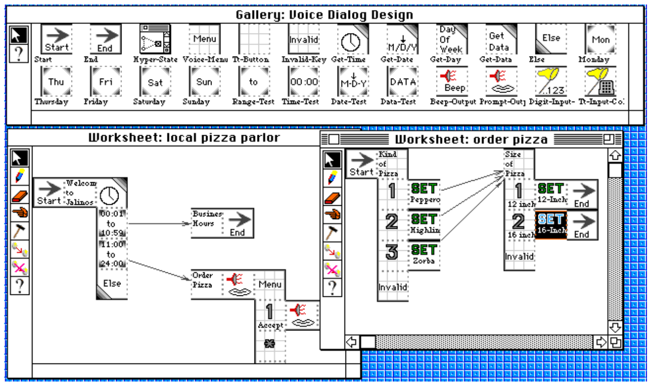


Figure 6. The Voice Dialog Design Environment

The design shown is an interface for a delivery service in a pizza parlor. If customers call outside of the restaurant's open business hours, they hear a standard message. If customers call during business hours, they can navigate through a series of voice menus to specify their pizza order. The design shown consists of two programs.

At any time, the behavior of the visual design representation 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. This simulation helps the designer to bridge the medium gap between the visual representation and the audio artifact. 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. The temporal metaphor, as well as the spatial metaphor, is defined by the three placement rules. These rules integrate the spatial and the temporal metaphors common to voice dialog application design. The temporal metaphor is realized at the agent level by a small set of sensors and effectors specifically defined to support simulation requirements in this domain.

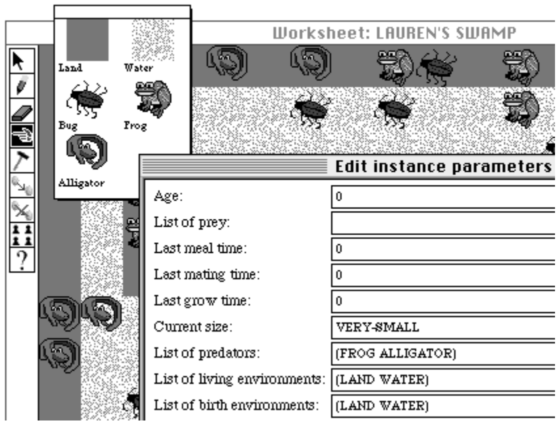
In [21], we describe how we successfully applied iterative prototyping and participatory design in the creation of the VDDE system. In less than four months, a mixed team of professional voice dialog designers and academic researchers were able to design and build a substantial core design environment. During the course of this collaboration, the design representation employed by the VDDE system has undergone many evolution cycles in response to user evaluations [17]. This evolutionary design approach was facilitated by Agentsheets support for the incremental refinement of both spatial and temporal metaphors.

The VDDE system provides the user with both end points in the spectrum of control. Users interact with individual design units using direct manipulation techniques to instantiate, rearrange, and modify design units. However, users interact with the collection of design units using delegation techniques. The user can initiate the simulation process by double-clicking on a “start” design unit (see Figure 6). Once initiated, the design simulation progresses as long as the necessary touch-tone button inputs are provided. In the VDDE system, symmetrical communication means that some of the sensors used for supporting direct manipulation by the user (e.g., sensors to detect mouse and keyboard input) are also used by the simulation process and thus support delegation techniques. The VDDE system illustrates how symmetrical communication allows a system to support both end points in the control spectrum: the remaining two applications will show how symmetrical communication can support the full range of participatory interaction.

5.2. EcoWorlds

A different type of Agentsheets applications are microworlds [13] in which agents are the inhabitants of simplified worlds. Microworlds are specially designed to highlight particular concepts and ways to think about these concepts [18]. The microworlds created in Agentsheets are similar in nature to SimCity-like applications excepts that radically new characters can be introduced by users and their behavior can be defined by the user rather than the creator of the microworld. New behaviors can either be defined by programming or by modifying parameters associated with generic characters.

Several ecosystem microworlds have been created using Agentsheets including EcoOcean, EcoSwamp, and EcoAlaska. Using these ecosystem simulations, the complex relationships between heterogeneous collections of creatures and their environment can be studied. The EcoSwamp application (Figure 8) is tailored to the microworld of swamps and swamp inhabitants. The dialog box shown provides a simple way to change the behavior of creatures by modifying parameters such as age, types of prey, size, and information about mating and sleeping.



Age:	0
List of prey:	
Last meal time:	0
Last mating time:	0
Last grow time:	0
Current size:	VERY-SMALL
List of predators:	(FROG ALLIGATOR)
List of living environments:	(LAND WATER)
List of birth environments:	(LAND WATER)

Figure 8. Parameter Modifying Dialog in EcoSwamp

Microworlds such as the EcoWorld simulations are problem solving devices in the sense that they can help users to experience how the seemingly simple behavior of individuals can have complex, sometimes hard to predict consequences for the collective system.

Programming is used to define the behaviors of individual creatures such as the alligators or frogs. These programmed behaviors represent hypotheses concerning how creatures operate in their world. Users determine what perceptions will lead to what reactions. Hypotheses are then tested by arranging the creatures in the microworld and observing their behavior. Typically, the behavior exhibited is much more complex than the anticipated behavior because the heterogeneity of the assembled creatures can lead to unexpected situations.

The EcoWorlds provide users with access to the simulation along the full spectrum of user control. Direct manipulation methods are used to create, configure, and arrange creatures in the microworlds. Delegation is used to initiate the simulation. The user can engage in participatory theater by adding new creatures or modifying existing creatures while the simulation is an progress. For instance, if the user notices that all the frogs in a certain area are dying out due to lack of food, instead of waiting for them to die and restarting the simulation, the user can add more food (insects) into the area in real time. This level of participation is facilitated by symmetrical communication. When a new insect appears, it is indistinguishable whether the user just added it or whether it is the product of two existing insects mating. In both cases, the same sensors and effectors are used.

5.3. Kitchen Planner

In the domain of kitchen design, appliances such as stoves, sinks, and refrigerators are arranged in a floor plan representation according to the owner’s requirements and the principles of good kitchen design. In this domain, principles of good design can be expressed as desirable spatial relationships between the various kitchen components. For instance, an important heuristic concerning an efficient “work triangle” states that the total distance between the stove, the refrigerator and the sink should be less than 23 feet. The goal of the Kitchen Planner (Figure 9) is to find new approaches for conveying such spatial design knowledge.

One approach for capturing design principles is to represent them as explicit rules in a computational critiquing system [2]. Critiquing systems observe and evaluate user actions and notify users when potentially problematic situations are detected. In the Janus system [1], a designer creates a kitchen floor plan by dragging appliances from a palette into a work area. A critic checking the work triangle rule would notify the designer if the components involved in the work triangle are too far apart. As such, critic rules only explicitly represent *evaluative knowledge*; they notify designers that something is wrong but provide no constructive knowledge about how to remedy the situation. Other knowledge-based components in the Janus system provide designers with this constructive knowledge.

In contrast, the Kitchen Planner conveys constructive knowledge through the use of a spatio-temporal hill climbing metaphor. In Janus, kitchen appliances are passive objects and the designer and the critiquing mechanism are the active agents in the design process. In the Kitchen Planner, these passive appliances are replaced with active appliances that make use of hill climbing agents. Hill climbing agents try to improve their situation by climbing up a conceptual hill representing their "happiness." Each agent will probe all of its immediate neighbor locations, determine the potential happiness at each location, and compare this with its current happiness. If any of the neighboring locations promise an increase in happiness, then the agent will move there.

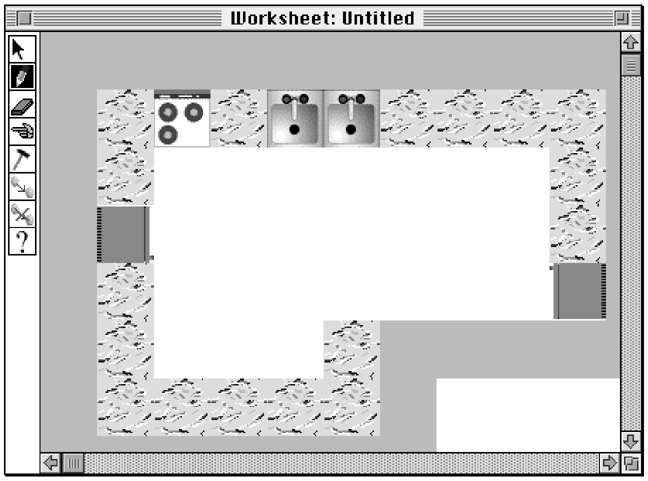


Figure 9. A kitchen floor plan.

Constructive knowledge can drive hill climbing. Evaluative knowledge typically consists of predicates returning a true or false value based on the current situation. For instance, the work triangle critic returns the same notification message to the designer regardless of whether the total distance is 24 feet or 2400 feet. However, hill-climbing agents provide a different knowledge representation of this same work triangle situation that returns a degree of fit. This result is used to constructively drive the hill climbing process.

The hill climbing process is complex because the landscape of the problem space implicitly defined by the evaluation function may lead the agent into local maxima leaving the agent unaware of even better places to be. Furthermore, typical Agentsheets hill climbing applications employ multiple hill climbing agents at the same time. Since most agents define their happiness in terms of the location of other agents, "happiness" literally becomes a moving target. The participatory theater approach allows users to participate in the hill climbing process and overcome some of these algorithmic limitations:

- *The hill climbing process has an intrinsic spatial manifestation*. The hill climbing agents are visible in the worksheet. Hence, the situation the agents are in can be inspected by the user.
- *Dynamic properties can be observed.* Sometimes there is no static solution and the agents never settle down, but instead keep moving. Dynamic properties of the solution, such as the participation of agents in cyclic behaviors, may be directly observable.
- *Participatory theater can turn the exploration of the problem space into a tactile experience*. Hill climbing is the implicit script handed out to the actors in the production. Users can be a passive audience or they can take the initiative and steer the hill climbing process. For instance, users can: 1) help agents to move out of local maxima, 2) change the environment of agents to remove a local maxima, or 3) modify parameters so that agents measure their happiness differently. Participatory theater can become a tactile experience. The synergistic combination of the forces of direct manipulation and hill climbing result in a tactile experience where the landscape of the problem space is perceived by touching its dynamic components. In the Kitchen Planner, the work triangle is not visibly represented in the worksheet. However, it can be indirectly experienced by trying to move the sink too far away from the refrigerator and oven and observing how these appliances rearrange themselves in response to the user's action.

Tactile interaction makes use of principles intrinsic to the participatory theater metaphor. On the one hand, user's can express design intentions through direct manipulation by simply moving components. On the other hand, components are autonomous in trying to optimize their happiness according to the guidelines attached to them. This can lead to conflicts between the intentions of the user and the attached design guidelines. Users can address these conflicts by modifying the strengths of individual guidelines or by freezing the positions of components.

5.4. A Comparison of these Applications

Table 1 summarizes the discussions of the three applications. All three applications relied extensively on Agentsheets facilities for creating spatial and temporal metaphors. In the VDDE system, the ability to modify these metaphors supported system designers to engage in an iterative, participatory design process with professional voice dialog application designers. In the EcoWorlds and Kitchen Planner systems, the ability to change representations allows users to substantially modify the

Table 1. Summary of the Three Applications

	VDDE	EcoWorlds	Kitchen Planner
1) Spatial Metaphor	Sequential control flow defined by adjacency	Relationships between creatures and their environment	Spatial relationships between appliances in the floor plan
2) Temporal Metaphor	Sequential control flow defined by adjacency	Creatures satisfy their life cycle demands; i.e., birth, eating, death	Appliances satisfy their urge for happiness; i.e., hill climbing
3) Change of Representation	Incremental evolution of metaphors supported iterative participatory design process	New creatures can be added dynamically	New appliances can be added dynamically Spatial design guidelines can be added dynamically
4) Control	Both end points in the spectrum: direct manipulation and full delegation	Full spectrum of control: direct-manipulation to full delegation	Full spectrum of control: direct-manipulation to full delegation Tactile experience: implicit design space experienced by "touching" components
5) Symmetry of Communication	In all three applications, behavior can be invoked by both users and agents.		

simulation microworlds. The VDDE system provides users with interaction schemes at both ends of the control spectrum. EcoWorlds and Kitchen Planner support the full spectrum of user control by allowing users to dynamically interact with and modify simulations as they are in progress.

6. Conclusions

In summary, the two notions of programming as problem-solving and human-computer interaction as participatory theater dictate five principles that systems should embody:

1) Systems should support the creation of spatial metaphors.

2) Systems should support the creation of temporal metaphors that are intrinsically integrated with the spatial metaphor.

3) It is essential that both of these metaphors can undergo continual modification.

4) Users must be provided with a wide spectrum of control over the dynamic behaviors in their solution representation.

5) Systems should provide symmetric communication facilities for both users and dynamic objects.

We have illustrated these principles with three problem-solving environments created using the Agentsheets substrate. In the last 4 years, the Agentsheets substrate has been used to create over 40 applications in a variety of domains such as a river basin management system, a children's storybook tool, a front-end to a power station's expert system [14], and a computer network design environment.

A substrate providing flexible representation mechanisms allows users to explore radically different approaches to solving the same kinds of problem. For instance, the Kitchen Planner uses hill-climbing to implicitly represent principles of good design whereas the VDDE system uses a critiquing approach [8]. The advantage of a substrate such as Agentsheets is that it lets us experiment with different types of representations including the notions of space, time, communication, and interaction.

Acknowledgments

We thank the HCC group at the University of Colorado and Clayton Lewis, who contributed to the conceptual framework and the systems discussed in this paper. We wish to particularly thank Jim Sullivan, the creator of the EcoWorlds applications. We also wish to thank Jonathan Ostwald, Kumiyo Nakakoji, and Gerry Stahl for their comments on early drafts of this paper. This research was supported by: the National Science Foundation under grant No. RED-9253425, Apple Computer Inc., and US West Advanced Technologies.

References

1. Fischer, G., A. Lemke, T. Mastaglio and A. Morch, "Using Critics to Empower Users," *CHI '90*, Seattle, WA, 1990, pp. 337-347.

2. Fischer, G., A. C. Lemke, T. Mastaglio and A. Morch, "The Role of Critiquing in Cooperative Problem Solving," *ACM Transactions on Information Systems*, Vol. 9, pp. 123-151, 1991.

3. Fischer, G. and K. Nakakoji, "Empowering Designers with Integrated Design Environments," *First International Conference on AI in Design*, Edinburgh, UK, 1991, pp. 191-209.

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

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

6. Green, T. R. G., "Cognitive Dimensions of Notations," *Proceedings of the Fifth Conference of the British Computer Society*, Nottingham, 1989, pp. 443-460.

7. Green, T. R. G., "Programming Languages as Information Structures," in *Psychology of Programming*, J. M. Hoc, T. R. G. Green, R. Samurcay and D. J. Gilmore, Ed., Academic Press, San Diego, 1990, pp. 117-137.

8. Harstad, B., "New Approaches for Critiquing Systems: Pluralistic Critiquing, Consistency Critiquing, and Multiple Intervention Strategies," University of Colorado at Boulder, M.S.. thesis, Dept. of Computer Science, 93 Pages, 1993.

9. Lakeoff, G. and M. Johnson, *Metaphors We Live By*, The University of Chicago Press, Chicago and London, 1980.

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

11. Nardi, B. and C. Zarmer, "Beyond Models and Metaphors: Visual Formalisms in User Interface Design," *Journal of Visual Languages and Computing*, ,pp. 5-33, 1993.

12. Negroponte, N., "Beyond the Desktop Metaphor," in *Research Directions in Computer Science: An MIT Perspective*, A. Meyer, J. Guttag, R. L. Rivest and P. Szolovits, Ed., MIT Press, Cambridge, MA, 1991, pp. 183-190.

13. Papert, S., *The Children's Machine*, Basic Books, New York, 1993.

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

15. Repenning, A., "Agentsheets: A Tool for Building Domain-Oriented Dynamic, Visual Environments," University of Colorado at Boulder, Ph.D. dissertation, Dept. of Computer Science, 171 Pages, 1993.

16. Repenning, A. 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. Repenning, A. and T. Sumner, "Creating Domain-Oriented Visual Languages: A Real-Time Collaborative Approach," *Submitted to: IEEE Computer (Special Issue on Visual Programming)*, , pp. 1994.

18. Resnik, M., "Beyond the Centralized Mindset: Explorations in Massively-Parallel Microworld," Massachusetts Institute of Technology, Ph.D. dissertation, Dept. of Computer Science, 176 Pages, 1992.

19. Schön, D. A., *The Reflective Practitioner: How Professionals Think in Action*, Basic Books, New York, 1983.

20. Simon, H. A., *The Sciences of the Artificial*, The MIT Press, Cambridge, MA, 1981.

21. Sumner, T., 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.