By Alexander Repenning
and Andri Ioannidou

The idea of using agents to create adaptable applications can help end users create complex interactive simulations and models.

AGENT-BASED END-USER DEVELOPMENT

THE GOAL OF AGENT-BASED END-USER DEVELOPMENT (EUD) is to empower end users with agents they can instruct directly. This process of instruction is completely transparent to the user; that is, it is not based on opaque adaptation mechanisms. Conceptually, the idea of instructing agents includes what is often called end-user programming [3] that addresses some of the major objections users have toward agents, such as the lack of trust and the need to train them. However, it poses the huge challenge of creating development tools suitable for end users who possess no programming background or interest in learning how to program.

Here, we describe how to facilitate EUD [4] by employing an agent-based approach as a means to create domain-oriented design environments (DODEs). We present two examples using the AgentSheets substrate:

Domain-oriented languages. Domain-oriented end-user programming languages are used to define the behavior of agents; for example, creating speech-based Web interfaces that pragmatically analyze and summarize information.

Domain-oriented agents. Agents can be used like building blocks in domain-oriented construction kits; for example, using agent-based design environments to allow people to learn about bridge design.

AgentSheets initially grew from the idea of building a new kind of computational media that allows casual computer users to

ILLUSTRATION BY HAL MAYFORTH

build highly parallel and interactive simulations. The simulations are used to communicate complex ideas or simply to serve as games. Adding a spreadsheet paradigm to agents enables the manipulation of large numbers of agents and their spatial organization in a grid. Partially influenced by the spreadsheet paradigm, agents live in an agentsheet—a grid-structured container of agents. In contrast to spreadsheet cells that hold text, numbers, or formulae, each agentsheet cell contains a stack of agents. The grid allows agents to employ implicit spatial relations (for example, adjacency) for communicating with other agents.

Agents in AgentSheets were designed to perceive and act by using a rich repertoire of multimodal communication capabilities. They can *perceive* mouse and keyboard input, sound and voice input (speech recognition), and Web page content; they can *act* by moving, changing appearance, creating new agents, playing MIDI music, speaking (speech synthesis), playing movies, evaluating formulae, and opening Web pages.

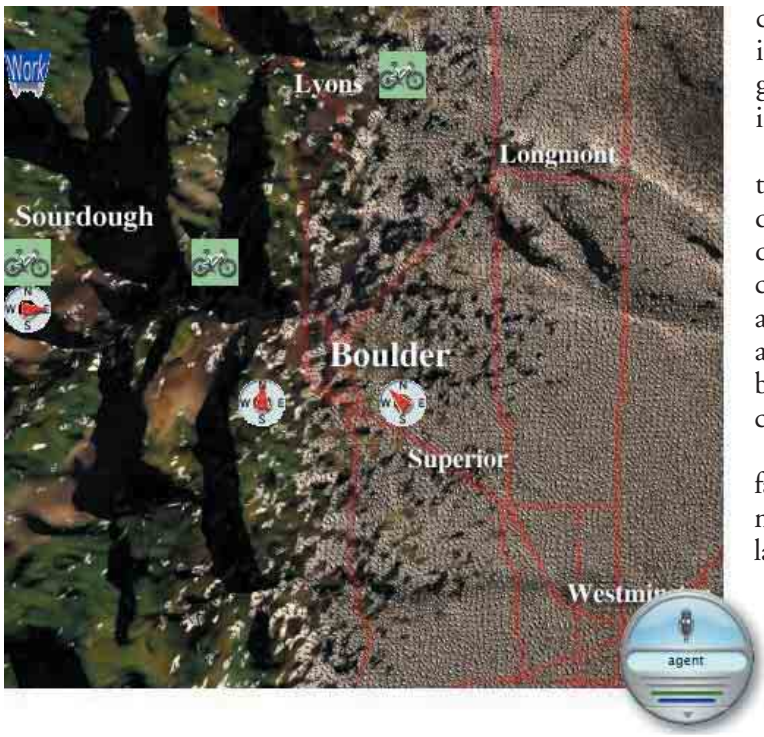

Figure 1. Agents arranged on a map use speech commands to retrieve, analyze, and synthesize Web information to make mountain biking recommendations.

Agents communicate with each other in a number of ways. They may be using spatial references expressed as relative or absolute cell coordinates, broadcasting without a spatial reference to agents of a certain type, or sending messages wirelessly to agents hosted on other computers.

Drag-and-Drop Interfaces with End-User Programmiing

Agent behaviors are specified using a rule-based language called Visual AgenTalk [5] (see Figure 2, for example). Rules are composed from predefined conditions and actions organized as methods that include a trigger defining *when* and *how* a method will be executed. AgentSheets users, ranging from elementary school children to NASA scientists, create a large number of simulations and games in a variety of disciplines including computer science, environmental design, fine arts, robotics, music, and biology.

In Visual AgenTalk, language components such as conditions, actions, rules and triggers, are computational objects complete with user interfaces. This enables end-user programming using drag-and-drop interfaces in three different ways:

Program composition. The combination of visual programming and drag-and-drop with active feedback can support the design and implementation of syntactically correct programs. For instance, an animated cursor indicates to a user that an action cannot be dropped into a condition box. If the user is still confused, the system can provide further explanation.

Program modification. The user interface employed to wrap up language elements of an end-user programming language must provide users with proper guidance on how to successfully modify working programs. In many cases this can be achieved by embedding common user interface widgets, such as pop-up menus, radio buttons, and check boxes into language components. In other cases, Visual AgenTalk utilizes custom widgets. A direction widget, for instance, provides an effective mechanism for an agent to refer to a relative grid position.

Program function perception. Visual AgenTalk enables users to play with any piece of the language. Borrowing notions of tactile interfaces, Visual AgenTalk allows users to perceive function through manipulation, which results in audiovisual feedback. Any kind of language component can be dragged and dropped onto agents:

- Conditions: Test the condition in the context of a specific agent. If the condition is not satisfied, the condition will blink and the computer will say "false."

- Actions: Make the agent execute this action.
- Rules: Step-by-step, with visual feedback, test all the rules provided. If there is a rule that can fire, execute all its actions. Otherwise, indicate which conditions are preventing the rule from firing.

DODE Applications

Agent-based EUD is a versatile approach to create DODEs. Domain orientation aids the design process by providing specialized building blocks conceptually close to the problem domain. In AgentSheets, domain-oriented building blocks exist at two different levels:

Languages. Domain-oriented end-user programming languages avoid the need to assemble functionality from low-level, generic, programming primitives. Visual AgenTalk is domain-oriented through the introduction of conditions and actions tailored to a specific problem domain.

Agents. Ready-made agents can be arranged by an end user to represent and simulate a design in a particular domain.

As a generic tool, AgentSheets itself is not a DODE; however, it is used to design DODEs. The idea of domain orientability is a form of metadesign [1]. We illustrate the two levels of building blocks through examples that indicate how agent-based EUD in AgentSheets is used to create DODEs.

Domain-oriented languages—The Boulder Mountain Biking Advisor. The lowest level of domain orientation takes place at the end-user programming level. For our conceptual framework, called The Pragmatic Web [6], we have created a number of Visual AgenTalk extensions to allow end users to quickly create multimodal user interfaces to existing Web-based information.

The Boulder Mountain Bike Advisor application (Figure 1) connects real-time Web information with speech recognition. A user who desires to go mountain biking utters the word "biking" and several agents located on a map of Boulder County react to this voice command. These agents represent locations that are possible candidates for biking and feature real-time, Web-accessible weather information sensors.

Rules defined by the users (Figure 2) capture pragmatic interpretations relevant to the users who defined them. The first rule of the Sourdough biking advisor agent (one of the agents that looks like a bike in Figure 1) has a speech recognition condition that becomes "true" if the agent hears the word "biking." The agent now triggers a second method, called "check," that includes two conditions that access a weather station Web page to extract information such as the current temperature and wind speed. The Fahrenheit information is numerically converted into Celsius and announced to the user by speech synthesis: "Temperature at Sourdough is currently –3.2 degrees Celsius."

Temperature and wind speed are further interpreted by calling a third method. This is the pragmatic part of the interpretation, which uses temperature and wind thresholds most relevant to the user who has expressed these rules. Unlike the objective part of the rule, which merely communicates the numerical value of the temperature, the pragmatic

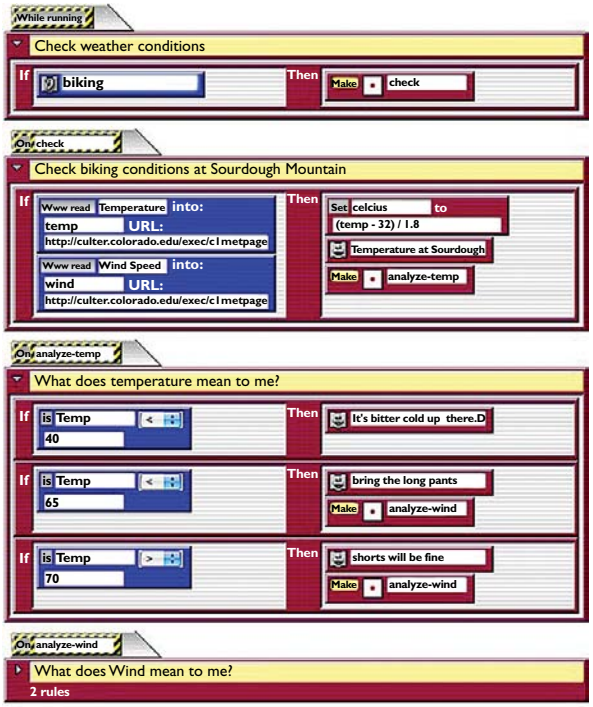

Figure 2. Visual AgenTalk rules for the Sourdough mountain biking agent.

part is directly employed to reach a decision. In our case, because the temperature is less than 40 degrees (Fahrenheit), the agent advises against a bike ride at Sourdough: "It's bitter cold up there. Don't come to Sourdough!" At more moderate temperatures, the agent would have recommended bringing additional clothing such as windbreakers in case the wind speed exceeds a different threshold. Because the entire dialog takes place using speech, there is no need for a visualization of the agents. This kind of application can be used to interface devices such as cell phones to existing Web pages.

The Pragmatic Web language extensions (such as the WWW Read condition) allow agents to parse Web pages. Agents can extract information from HTML or XML pages.

Domain-oriented agents—The Bridge Builder. The Bridge Builder is a bridge design environment

for education. The objective of the Bridge Builder activity is to develop an understanding for static and dynamic forces on a bridge. Users are instructed to build a bridge with the minimal number of bricks. Users modify the bridge design by adding, removing, and rearranging bridge components. Figure 3 shows evolving bridge designs.

The Bridge Builder is a DODE that provides domain-oriented agents created for end users by a high-level language designer. For the Bridge Builder, EUD is the process of composing agents representing bricks, tunnels, and soil to create a working bridge design. To evaluate the stability of a bridge, agents must solve complex diffusion equations and exchange information with adjacent agents.

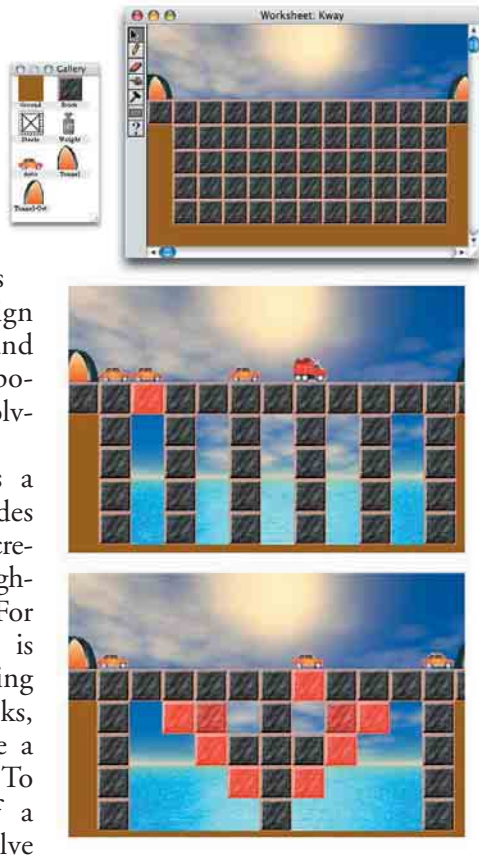

Figure 3. A bridge is designed by selecting, and composing agents from a palette (top); An ancient Greek-style, straight-column design works but still uses too many bricks (middle); A Roman-style involving arches reduces the number of bricks (bottom).

This design environment is live in the sense it is a running simulation that provides feedback to any design modification. Bricks under a great deal of stress will change their color, providing users with valuable design information. If, in an effort to build a bridge with the least number of bricks, a user removes too many bricks, the bridge will collapse and the cars will crash.

The AgentSheets metaphor is powerful because, in addition to its generality, it supports emergence [2]. For instance, the Bridge Builder brick agents, although not computationally trivial, have no notion of column-based or arch-based bridge design. The solidity of a bridge design is an emergent property.

Another AgentSheets example of a DODE featuring agents is the Kitchen Design, in which agents represent refrigerators, kitchen sinks, and cabinets that are arranged into a kitchen design. An example of more abstract domain-oriented agents is the Voice Dialog Design Environment [7], which lets designers of phone-based services quickly build working prototypes of nested dialogue structures.

Conclusion

Agent-based EUD explores the use of agents to create adaptable applications. Specifically, this research investigates how end users can control agents by explicitly instructing them through domain-oriented building blocks. This process of instruction may unfold at different levels. The lowest level features domain-oriented end-user programming languages. Program manipulation mechanisms such as drag-and-drop with active feedback help users to compose, modify, and perceive the functions of programs. At a higher level, domain-oriented agents are used as building blocks of design environments. We are currently working on an even higher level that uses wirelessly connected handheld devices to run distributed simulations and to control a central simulation that can become the domain-oriented building blocks of collaborative design activities. In the Mr. Vetro application, end users experience human physiology through collaborative role-playing. Individual organs such as the heart and lung are simulated on wirelessly connected PocketPCs. C

References

1. Fischer, G. Meta-design: Beyond user-centered and participatory design. In *Proceedings of HCI International 2003* (Crete, Greece, 2003), 88–92.
2. Johnson, S. *Emergence: The Connected Lives of Ants, Brains, Cities, and Software*. Touchstone, New York, 2002.
3. Nardi, B. *A Small Matter of Programming*. MIT Press, Cambridge, MA, 1993.
4. Paterno F. D1.2 research agenda: End-user development: Empowering people to flexibly employ advanced information and communication technology. *End-User Development Network of Excellence* (2003), 17.
5. Repenning, A. and Ambach, J. Tactile programming: A unified manipulation paradigm supporting program comprehension, composition and sharing. In *Proceedings of the 1996 IEEE Symposium of Visual Languages* (Boulder, CO, 1996), IEEE-CS, 102–109.
6. Repenning, A. and Sullivan, J. The Pragmatic Web: Agent-based multimodal Web interaction with no browser in sight. In *Proceedings of the Ninth IFIP TC13 International Conference on Human-Computer Interaction* (Zurich, Switzerland, 2003).
7. Repenning, A. and Sumner, T. Agentsheets: A medium for creating domain-oriented visual languages. *IEEE Computer 28,* 3; 17–25.

Alexander Repenning (ralex@cs.colorado.edu) is the CTO of AgentSheets, Inc., and a professor of computer science at the University of Colorado at Boulder. He is the creator of the AgentSheets simulation and game-authoring tool.
Andri Ioannidou (andri@agentsheets.com) is the senior project manager of AgentSheets, Inc., Boulder, CO.