End-User Programming of Web-Native Interactive Applications

(Keynote Paper)

Mehdi Jazayeri, Navid Ahmadi
Faculty of Informatics
University of Lugano
Lugano, Switzerland

Abstract: *Web 2.0 has enabled Web users to create and share a variety of hyper-text based artifacts including embedded images, sound, and video on the Web. Creating Web-based interactive artifacts such as computer games, however, has remained a challenge: to end users due to the lack of end user programming tools; and to programmers due to the poor interactivity performance of the Web. With the emergence of HTML5 and improving performance of JavaScript engines, professional Web programmers have only just begun to develop Web-native interactive artifacts. Today's standard Web technologies make the Web a hospitable platform for efficient interactive applications both for professional programmers and end-users. With proper support, in tools and languages, end-user programming of interactive applications is feasible. In this paper, we review the current state of Web application development and the possibilities and potential benefits of end-user programming on the Web. We will use a case study, AgentWeb, a Web-based end-user development environment, as a representative of interactive Web applications. It is based completely on open Web technologies, rather than on any proprietary technologies. Given that 2D graphic interactive applications may be developed and efficiently executed on the Web, we discuss some of the potential applications in educational settings, including individual and collaborative learning.*

Key words: *World Wide Web, HTML5, end-user programming, Web applications, Web programming, open Web, Web native applications*

INTRODUCTION

The original view of the World Wide Web (W3) was to be "a pool of human knowledge, which would allow collaborators in remote sites to share their ideas and all aspects of a common project." [1] It was introduced in the early 1990s with the goal of making it possible to access information from any source in a consistent and simple way. Developed at CERN, in Geneva, Switzerland, it was aimed originally at physicists and other scientists that generate huge amounts of data and documents and need to share them with other scientists.

A number of basic technologies were developed to help implement these ideas. The HTML language was invented and adopted as a simple way to both describe the format of documents and to refer (link) to other documents. The HTTP protocol was designed to allow one computer—the *client* computer—to request data and documents from another computer—the *server* computer—so that it could make that document available to the users on the client computer. In this way, the World Wide Web was viewed as a vast repository of information that provided access to a large number of users. The original definition of W3 also provided the concept of a uniform resource identifier (URI) that could be used to identify uniquely each entity that is accessible on the Web. These entities could point to each other using the concept of link supported by HTML. This view of the Web was quite static, with producers of information creating pages and consumers of the information clicking on links to view the information.

With the introduction of the Internet to the commercial world in 1995, businesses and the general public started using the WWW and the static nature of the information

structure on the Web was no longer sufficient: businesses wanted their customers to be able to communicate back with them. They wanted the customers to be able to place orders for products and not only read their product catalogues. Fashion designers wanted their customers to be able to design or customize dresses by combining several pieces of clothing. Furniture stores wanted their customers to be able to virtually decorate a room with pieces of furniture. In general, there was a need for ordinary users of the Web to contribute information to this "pool of human knowledge." The so-called Web 2.0 incarnation of the Web was characterized, indeed, by increasing "user participation." [2] There was a wave of applications that allowed users to contribute and share ideas and opinions (blog posts), photos (e.g. Flickr), music (e.g. Last.fm), and video (e.g. YouTube) content, and comment on existing content (e.g. most newspapers). The Web became increasingly multi-media, and users were enabled to contribute content to it. Yet, user involvement remained static in that users could only add static entities, such as text, photo, or video.

The initial design of Web technologies which was based on a "Web of documents" perspective, did not accommodate the requirements to execute interactive applications in the Web browser. The Web browser was expected to only interpret the HTML commands of the page and display the results on the client computer. Therefore, developers had to produce interactive Web applications using so-called rich Internet applications (RIAs). Such applications have become ubiquitous on the Web and exhibit both high performance and high interactivity [4]. RIAs are written by programmers, often using proprietary tools and languages. Nevertheless, the growth of user participation and the shift of Web pages to Web applications required native support for developing Web applications, which led to the development of HTML5 standard. The advent of HTML5 makes it even easier to develop Web applications that offer sophisticated user interfaces and run on nearly all browsers. HTML5 adds a number of particularly interesting features for high-performing interactive applications [9]:

- Canvas: A new tag "canvas" enables page developers to create animated graphics in HTML without needing a plug-in. This will make graphics both perform better and be more portable across browsers.
- Support for video/audio: audio and video will also be natively embedded in the Web page allowing browsers to play them without needing a plug-in. browsers will be (and are increasingly even now) shipped with co-decs that will be able to decode audio and video files for playing.
- Native support for drag and drop: HTML5 will support native drag and drop, so that objects may be dragged from one window and dropped into the Web page. Currently, this is done with specialized libraries.

This paper deals with removing the static restriction of artifacts and enabling Web users to create and contribute interactive artifacts to the Web. This is not possible using RIAs because they are proprietary environments designed for professionals. But now that HTML5 is emerging, open Web environments can support non-programmer Web users in developing interactive applications.

Interactive artifacts are, in fact, as software people know, nothing but programs. So, if users of the Web could program, they could also add interactive content to the Web. But most users are not programmers. How can we enable them to program on the Web? We can certainly rely on a long history of research in end-user programming on traditional computing platforms [5,6]. Fortunately, the Web offers an even better platform for end-user programming for two reasons. First, there are many potential users. Second, the success of Web 2.0 has shown that many users are ready to participate in the development of artifacts. We assume that at least a reasonable percentage of these users will be interested in learning how to program if it does not require years of technical study. In this paper, we show that by combining some of the lessons learned from end-user

programming research and appropriate use of current Web technologies we can enable end-user programming on the Web. We rely on a case study of a computer game design environment as an example.

AGENTWEB: A COMPUTER GAME DESIGN ENVIRONMENT

AgentWeb is a Web-based computer game design environment targeted for non-programmers. It is written completely in open Web technologies with no reliance on proprietary software or applications. That is, it is written in JavaScript, HTML, and CSS, which are all industry standards, and available in all Web browsers. It uses a few open technologies for its user-interface implementation, namely, canvas for 2D graphics and Dojo toolkit for supporting drag-and-drop operations. These standard technologies are sufficient to give AgentWeb the familiar look and feel of desktop applications.

The reader may want to try AgentWeb at http://agentweb.inf.usi.ch/. It offers a visual game development environment in which users can design and program games. The programming model for games is agent-based, rule-based, and visual. For a user to create a game, they have to create agents and program the behavior of agents by writing rules for the agents to follow. Rather than introducing a new interface to the users, AgentWeb's user-interface reproduces that of AgentSheets [7,8], which runs on a desktop. AgentSheets has a rule-based, visual language and has been shown to be convenient for novices and end-users. Indeed, AgentSheets has been deployed successfully in education all the way from elementary schools through university classes. Users find its interface easy to learn and use. Therefore, the goal of AgentWeb is to demonstrate that AgentSheets-like applications can be hosted entirely and natively on the Web.

AgentWeb offers the user a development environment (IDE) with panes for different activities:

- Agent Gallery lets users create agents and icons to represent them.
- Image Editor lets users draw customized icons for the agents.
- Programming Environment lets users program the agents using a visual, rule-based programming language.
- Scene Editor, known as worksheet in the IDE, lets users create the game scene by instantiating the agents and executing them. The worksheet is organized as a rectangular grid of cells. Each cell contains zero or more agent instances.

Figure 1 shows the IDE of AgentWeb being used to create a version of the familiar Frogger game. As the IDE shows, it is a highly graphical, interactive, environment. It offers familiar and friendly features such as drag-and-drop for the convenience of the user.

Once a game scene has been created, the user can start (i.e. execute) the game. In the case of the Frogger game, this means that the user (player) can try to move a frog from one side to the other trying to avoid the oncoming traffic and also from falling into the river.

Our experiments have shown that users, such as middle school students, find AgentWeb easy to learn and use. In general, they take the same amount of time to create a game with AgentWeb as they do with AgentSheets. The execution speed of the game is also acceptable from the users' point of view. We have measured the performance of the games in various browsers. While there are variations in the performance of different browsers, they are all getting increasingly faster and we observe less and less difference between the execution speed of AgentWeb and its desktop counterpart.

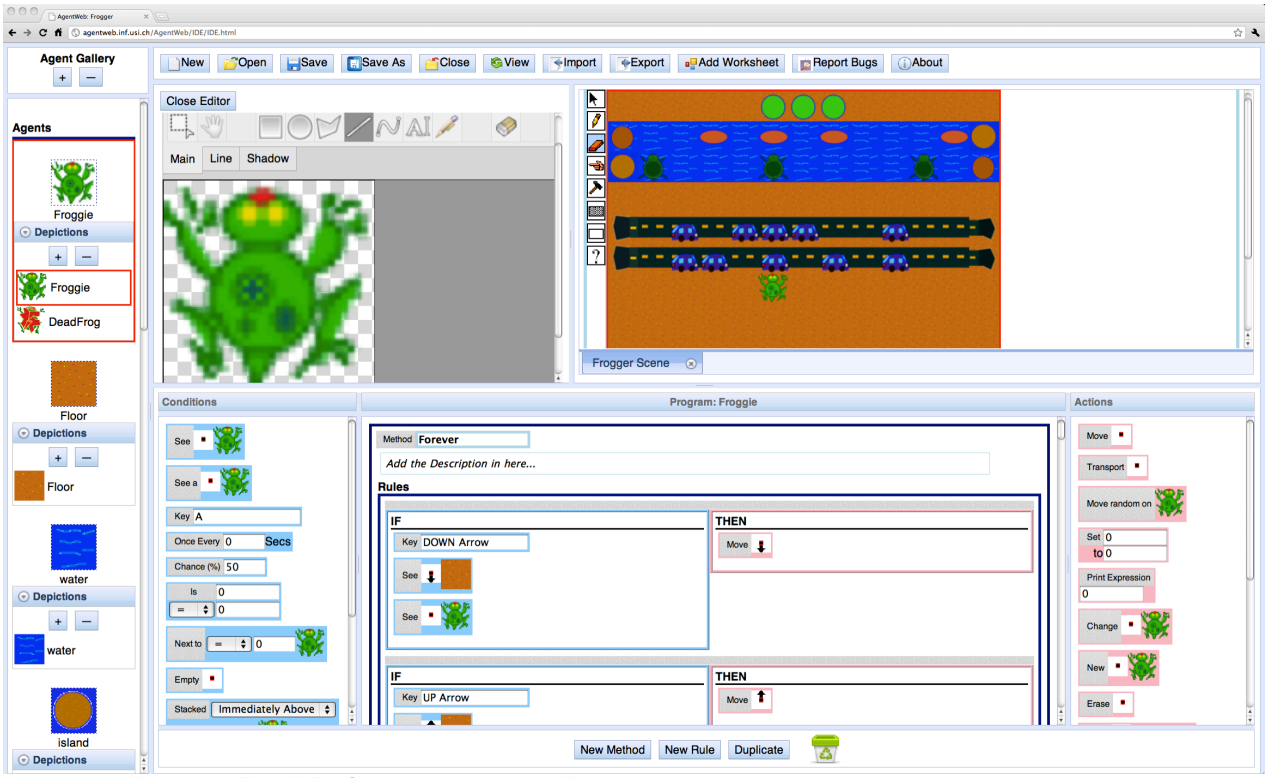

Figure 1. The IDE of AgentWeb with a Frogger game being designed in the Web browser.

AGENTWEB: IMPLEMENTATION

AgentWeb's design and implementation are, of course, completely different from those of AgentSheets because it runs on the Web and in a browser. Perhaps surprisingly, its good performance is not due to its use of proprietary software. It is written completely with Web native technologies, primarily in JavaScript. Its most interesting feature is that it automatically translates the visual rules created for agents by the user into JavaScript code. So, when a game is executing, it is native JavaScript that is running. As JavaScript engines and browser implementations have improved, they impose increasingly less overhead on the execution of AgentWeb and therefore we observe better performance.

The primary reason for the high performance of AgentWeb is that both its IDE and the execution environment run entirely in the browser with no communication back to the server (once the environment is downloaded to the browser).

The user interface of AgentWeb has been built using CSS, HTML and Dojo Toolkit. Dojo provides layout management and drag-and-drop libraries out-of-the-box, which fits the requirements for building an IDE. Such open-source toolkits facilitate the programming of friendly Web applications.

The IDE panes, e.g., visual programming environment, have been built following a model-view-controller (MVC) architecture [10], which separates model (Web-native executable game) from the view (the game being designed by the user) and connect them through the controller (compiler and runtime system). Such architecture, when deployed completely on the client-side, turns the IDE to a direct-manipulation environment that updates the running game instantly according to the modifications that the user makes in the IDE.

PERFORMANCE OF WEB-NATIVE WEB APPLICATIONS

Since the Web was not designed for running interactive applications, a natural question is whether Web-native applications can meet the performance requirements of interactive applications. We explored this issue by developing 'Ristretto^{Mobile}', a compiler

and execution engine for transforming interactive applications built in AgentSheets to Web-native applications running on JavaScript and HTML5 Canvas [11]. As mentioned earlier, AgentSheets is an authoring tool used by school children, to build games and science simulations. By optimizing the execution and rendering engines, we were able to reach execution speeds high enough for running interactive games in the browser. For instance, Figure 2 shows a simulation of E. coli bacteria [12], an instance of CPU- and graphic-intensive application, running more than 2000 agents at 35 fps (frames per second) on a personal computer's Web browser. Thirty fps is considered adequate for interactive response of such applications.

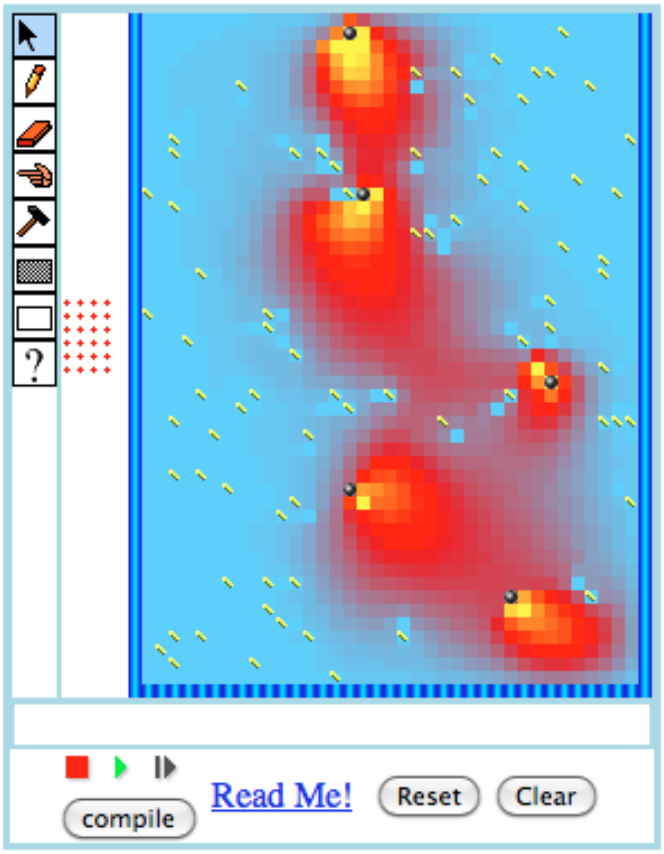

Figure 2. Simulation of E. coli bacteria as a Web-native interactive application. More than 2000 agents run at 35fps in the Web browser.

CONCLUSIONS AND FUTURE WORK

We have observed that the current open Web standards are sufficient to support sophisticated end-user programming of graphical interactive applications. This is an exciting opportunity because it indicates that the power of user participation demonstrated by the success of Web 2.0 can be harnessed to enable the collaborative development of applications and simulations of interest to diverse groups of Web users. The number of users of the Web is huge and rising ever rapidly. Their interests are also wide and varied. Commercial software producers cannot satisfy the interests of all users. Web-enabled end-user programming tools can fill this vacuum.

End-user programming and simulation have been applied with success in educational environments both to engage students at early stages and to help non-computer specialists to apply computing to areas of their specialty such as in social or physical sciences. By bringing end-user programming to the Web, these and other applications can be explored by a vast number of people. We have shown only one paradigm for end-user programming, that of agent-oriented, rule-based, visual programming. More work is

needed to explore other paradigms and different domains of application. The current Web and browser technologies are up to the challenge.

The use of open standard Web technologies means that the developed tools are portable across platforms, existing ones and those yet to come. This is particularly beneficial for running interactive applications on mobile devices, which are highly diverse in hardware and execution platforms. For example, browsers that run on new tablet and other mobile devices can run such applications without modification. It is likely that such devices, rather than traditional computers, will become the dominant access points to the Web. Using open Web technologies to build Web native applications can ensure that users of such devices will enjoy the benefits of the new class of interactive applications.

REFERENCES

[1] T. Berners-Lee, R. Cailliau, A. Luotonen, H. F. Nielsen, A. Secret. The World-Wide Web. Comm. ACM 37(8): 76–82, August 1994.

[2] T. O'Reilly. What is Web 2.0 - design patterns and business models for the next generation of software. 2005.

[3] M. Jazayeri. Some trends in web application development. 2007 Future of Software Engineering (FOSE '07), pages 199–213, 2007, held at International Conference on Software Engineering (ICSE 2007).

[4] P. Fraternali. Tools and approaches for developing data- intensive web applications: a survey. *ACM Computing Surveys*, 31(3):227–263, 1999.

[5] M. Burnett, C. Cook, O. Pendse, G. Rothermel, J. Summet, and C. Wallace, End-user software engineering with assertions in the spreadsheet paradigm. Proc. *ICSE*, 2003:93-103.

[6] C. Kelleher and R. Pausch. Lowering the barriers to programming: a taxonomy of programming environments and languages for novice programmers. ACM Computing Surveys, 37(2):83–137, 2005.

[7] A. Repenning, AgentSheets: an interactive simulation environment with end-user programmable agents, Interaction, 2000.

[8] A. Repenning, A. Ioannidou, J. Zola, AgentSheets: End-User Programmable Simulations. J. Artificial Societies and Social Simulation 3(3): (2000)

[9] B. Johnson, The Web is reborn. Technology Review, November/December, 2010:46-53.

[10] T. Reenskaug, "Models - views - controllers," Technical note, Xerox PARC, 1979.

[11] N. Ahmadi, A. Repenning, and A. Ioannidou, "Collaborative end-user development on handheld devices," IEEE Symposium on Visual Languages and Human-Centric Computing(VL/HCC), pp. 237-241, 2008.

[12] D. Klaus. Microgravity and its implications for fermentation biotechnology. Trends in Biotechnology, 16(9):369–373, 1998.

ABOUT THE AUTHORS

Mehdi Jazayeri is professor of informatics in the Faculty of Informatics at the University of Lugano. He was the founding dean of the faculty which started in October 2004. Before that he was at the Technical University of Vienna as professor and head of the Distributed Systems Group. He is a Fellow of the IEEE.

Navid Ahmadi is a PhD candidate in the Faculty of Informatics at the University of Lugano. He has been researching end-user programming on the Web. He is the developer of AgentWeb, a Web-based game design environment for end users, built using open Web technologies.