Towards the Web of Applications: Incorporating End User Programming into the Web 2.0 Communities

Navid Ahmadi, Mehdi Jazayeri, Francesco Lelli
Faculty of Informatics
University of Lugano, Switzerland
firstname.lastname@usi.ch

Alexander Repenning
AgentSheets, Inc.
Boulder, CO 80301
U.S.A
alexander@agentsheets.com

ABSTRACT

The Web is evolving from the Web of documents to the Web of applications. Web 2.0 communities need end-user programming tools to create interactive applications according to their skills and domain of interest. However, due to the different domains of programming in different communities, providing a generic EUP tool to all communities is not possible. We aim at encapsulating development of domain-oriented EUP tools as a community-based effort-taking place at two layers: (i) among different communities, and (ii) inside a particular community. Accordingly, we suggest a domain-independent model and its respective Web-based infrastructure to supports different online communities to create and appropriate end user development tools for building interactive artifacts collaboratively.

Categories and Subject Descriptors: H.5.3 [Information Interfaces and Presentation]: Group and Ogranization Interfaces

General Terms: Human Factors, Languages

Keywords: Web 2.0, end user programming, community-based domain construction, interactive applications, Web-based applications

1. INTRODUCTION

Web 2.0[19] is evolving the Web from the Web of documents to the Web of applications[21], where typical Web users will build their own applications. Web 2.0 has promoted communities of Web users to develop different types of static artifacts such as images and movies collaboratively. However, development of interactive artifacts as a collaborative activity in Web 2.0 communities has remained challenging since Web 2.0 communities consist of mostly typical Web users with no background in programming. Therefore they need end user programming (EUP) tools to create interactive artifacts.

Tools and models developed in end user programming studies can be tailored to enable end users of Web 2.0 communities to create interactive artifacts. However, different Web 2.0 communities are associated with different domains each of which needs its own EUP tools to create interactive artifacts. Therefore, devising a generic EUP tool for all the communities is not possible. Rather, each community needs its own domain-specific tools to create such artifacts.

To address the "domain challenge", we aim at encapsulating the development of domain-oriented EUP tools as a community-based effort that happens inside Web 2.0 communities. We base our approach on leveraging the collaborative dimension of end user programming, which has been recognized as a persistent activity in several EUP contexts, to enable communities to build domain-specific tools collaboratively. Accordingly, we suggest a generic model to support the development of domain-oriented EUP tools in Web 2.0 communities. We consider embodying domain development process as a community-effort at two distinct layers: (i)**inter-domain tool construction,** where different communities share a generic infrastructure that supports development and sharing of EUP tools and components; (ii) **intra-domain tool appropriation,** where members of a particular community collaborate to adapt developed tools from the infrastructural layer and shape a domain-specific end user development environment according to the community's domain.

The rest of the paper is organized as follow: in Section 2 we provide the state of the art of programming by end users on the Web; in Section 3 we explore the collaborative dimension of end user programming on which we rely our work. In Section 4 we demonstrate our suggested model to support community-based development of EUP tools. Finally, in Section 5 we present our future work, and conclude.

2. EUP ON THE WEB: STATE OF THE ART

In this section we investigate the current situation for supporting typical Web users to write programs on the Web. Respectively, we do not consider other aspects of end user development such as designing dynamic HTML pages as programming and skip them here.

Web 2.0 has activated typical Web users to contribute to the data and information society. But the users' potential for contribution goes beyond just injecting the information to the Web. Web users are potentially able to customize, process and adapt the existing data according to their personal and organizational needs. Recommendation systems are the simplest examples of tools to enable users to customize the information by collecting and recommending useful information to other people on the Web. The demand to mix data

SoSEA'09, August 24, 2009, Amsterdam, The Netherlands.

and services on the Web according to the user's need has led to the development of Web Mashups[26], i.e. a combination of data and services from several sources on the Web. The data is usually provided as a feed or the output of other services, and the services can be Web and REST. Mashups are native Web applications written in JavaScript, and using AJAX, JSON and XML technologies. The fast growth of Web mashups proves the need for processing Web data and building applications on top of them. Another type of tools to enable Web users to extract data from pages and automate tasks over the data are called automators. Examples of such tools include Chickenfoot[5, 4] and coScripter[10, 3].

To lower the barrier of writing a Web mashup, tools have been created mostly by commercial companies but as free Web-based software. These atifacts usually provide a higher level of abstraction for the programming of mashups. For example Yahoo! Pipes[1] provide a GUI to pipe data to operators such as aggregators and services. Google Mashup Editor (GME)[2] provide a higher level API to use in JavaScript while building mashups. Additional mashup building tools besides Yahoo! Pipes and GME, includes Microsoft Popfly, and Intel Mashmaker[2]. From the end-user programming point of view EUP is not the target domain for these tools. They are mostly suitable for amateur and professional programmers. However, there exists some research on enabling end users to build mashups. Marmite[24, 25] and Vegemite[11] combine concepts from data-flow programming, scripting, and spreadsheets to enable end users to build mashups.

We can conclude that Web mashups and mashup building tools present the power of existing technology to build Web-based applications using data feeds, AJAX, JavaScript, and REST APIs. However, according to the described properties of Web 2.0 communities, mashup tools do not satisfy the requirements to support Web 2.0 communities to build programs, since:

- Mashup tools support mixing data and services. Building Web-based applications is left to JavaScript, which is not supposed to be used by end users of Web 2.0 communities.
- Mashup tools do not provide enough flexibility to build interactive applications which can react according to the user's behavior and environment's triggers.
- Mashup tools provide domain specific languages for mixing data and services. However, Web 2.0 communities have different domains, which demand specific models and languages.

3. COLLABORATIVE EUP

End-user programming provides languages, tools, and models to enable end users to write programs. End users are usually domain experts, such as a physician, biologist, or librarian, without background in programming. They also have computational needs and are not interested in programming per se. In the software context, users are using end-user programming approaches to customize a software system. They are use to perform these changes while they are using the artifact rather than at design time, and this behavior is known as "tailoring".

[1]http://pipes.yahoo.com
[2]http://editor.googlemashups.com

Nardi[17] argues that end users tend to gain computer knowledge and solve their computing problems by seeking help in their social environment, rather than laboring in isolation. Such pattern has been seen as a persistent activity and not the exception in several computing areas including sharing UNIX customization files[12], spreadsheets[18], tailorable office systems[13], and CAD settings [8]. The collaborative tailoring pattern have been demonstrated in other works as well [23][14][6].

Nardi[17] encounters a continuum of users with different skills ranging from end users to local developers to programmers in a community of end users and highlights the emergence of local developers in any settings where a computer program is widely used. Local developers are defined as: "domain experts who happen to have an intrinsic interest in computers and have more advanced knowledge of a particular program"[17]. Fischer et al.[7] extend this continuum to the consumer/designer spectrum ranging from passive consumer to well-informed consumer, end user, power user, domain designer, and meta designer. According to Pipek et al.[20], Oberquelle proposes a classification of groupware tailoring based on collaborating actors, who are individuals or a group, from people affected by a tailoring activity, who can be individuals or a group.

Maclean et al.[13] argue that tailoring has to become a community driven activity based on the expertise of involved users. Morch et al.[16] argue that the context in which a software is being developed is different from the context in which that software is being used, and to reach a shared context, tailoring has to be done as the indirect long-term collaboration between developers and user. They suggest "Multiple Representations" to capture and preserve a rich development context to be shared among developers and users, and "Application Units" which make multiple representations accessible to end users at use time. Fischer et al.[7] introduce meta-design, a conceptual framework aimed at providing social and technical infrastructures to support collaborative design, based on the assumption that future uses and problems can not be anticipated at design time. Pipek et al.[20] categorize the existing collaborative tailoring works, distinguishing between three levels of intensity of user involvement with tool usage, namely: "shared use," "shared context," and "shared tool" and introduce a fourth scenario called "shared infrastructure", and for each they discuss approaches to support collaborative scenarios. The goal in shared infrastructure tailoring is to support the appropriation beyond one tool or application. The challenge here is to support collaborative tailoring concretely due to the lack of a "targeted" design for one specific application. Similar to the "domain challenge" described in the introduction of this paper, the problem in here is a trade-off between the generality of the infrastructure respective to different domains and meeting domain's tailoring rationale.

At the programming language level few of end-user programming languages provide support for collaboration. Researchers have created a number of end-user programming models and languages since early 1960s in different domains and with different goals. Kelleher and Pausch[9] classify end-user programming languages according to their main purpose, where at the highest level they are divided into two branches of (i) languages with the purpose of teaching the programming concepts, and (ii) languages for empowering end users with programming to get their job done. Kelle-

her and Pausch identify three types of collaboration that are supported in different programming languages. (i) side-by-side-based collaboration in which two or more users were manipulating the same program on computers that were located in the same room, (ii) networked-shared manipulation in which users were in different locations but connected to a common network and could collaborate while building a program, and (iii) networked-shared results in which users were in different location but connected to a common network and could share completed programs or program fragments. Based on their survey of fifty eight end-user programming languages, there are only ten models supporting one of these three types, in which six of them support side-by-side collaboration, only one of them support networked-shared manipulation, and three of them support networked-shared results.

4. INCORPORATING EUP INTO WEB 2.0

In this section we propose a model to support development of domain-specific EUP tools across different Web 2.0 communities. To cope with the "domain challenge" we encapsulate development of domain-specific EUP tool as a process that takes place inside the community. However, it has been discussed that the appropriation of existing tools is often more efficient than construction of tools for a domain from scratch[15]. Accordingly, we divide the community-based development of EUP tools into two layers: (i)**inter-domain tool construction,** where different communities share a generic infrastructure that supports development and sharing of EUP tools and components; (ii) **intra-domain tool appropriation,** where members of a particular community collaborate to adapt developed tools from the infrastructural layer to shape a domain-specific end user development environment according to the community's domain.

4.1 Structure of a Community

We base our model on the presence of the same programming skill continuum among users in the Web 2.0 communities as those observed in other end user programming settings as explained in Section3. Such communities are composed of mostly end-users, however, the presence of few domain experts with programming knowledge and few professional programmers in such communities are expected. We rely on the presence of a continuum of these three main roles, namely, end users representing domain experts, domain programmers, and professional programmers, to conduct the development of EUP tools requested by the end users. Note that every individual can take more than one of these three roles. For example, a computational scientist can be at the same time, a domain expert and a professional programmer, which implies his role as a domain programmer. As we will see later, in our model we define each role according to the type of activity they are involving in.

Figure 1 demonstrates main roles and their primary assigned activity in our model. End users are supposed to deal with the EUP tools to build interactive applications. Domain programmers build domain specific instructions, to be used or requested by end users, over the provided programming language and simulation environment. Professional programmers develop generic EUP languages and simulation environments according to the domain experts' needs. The goal of this approach is to found a meta-design environment[7] with several involvement points in which users with different interests and programming skills collaborate on the

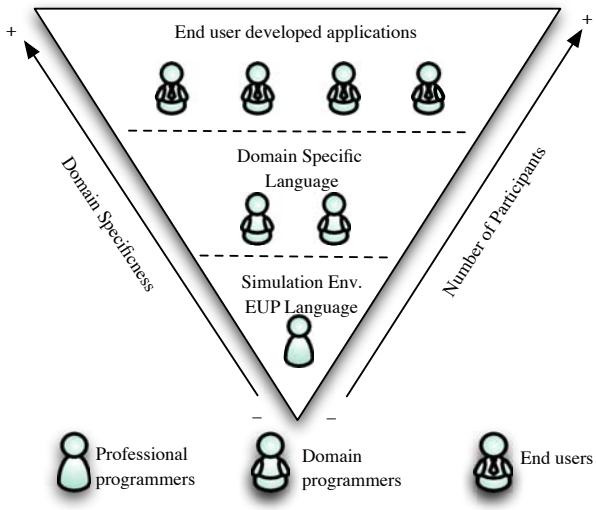

Figure 1: Intra-community collaboration for tool construction and appropriation.

design and implementation of components required for developing interactive artifacts(see next paragraph). As a result of providing domain-specific EUP tools for Web communities, we expect that a larger number of end users gain access to computational tools.

In order to enable appropriation of tools developed by other communities, we focus on structuring and modularizing the basic tools that can be built and shared at the infrastructural level. As a result, the appropriation phase will be simplified to choosing required modules and bundling them into a domain-specific EUP environment. To modularize the basic tools, we consider the generic components required to build interactive artifacts. To enable end users to build interactive artifacts, interactive EUP environments, such as AgentSheets, consist of (i) a programming paradigm suitable for end users; (ii) related domain-specific language (DSL) that captures the domain expression; and (iii) an execution environment that simulates the world in which objects interact. We consider the infrastructure to support development of these three types of components together with their underlying compiler that turns the end-user developed program into the machine-executable program. In the rest of the paper we consider these components of interactive domains as the *interactivity components*.

4.2 Supporting Inter-Domain Development

In our model, tool development is not an isolated task that happens solely inside a particular community. Rather, different communities with common interest in a particular tool have to be able to contribute to the development of that tool. The essence of this approach is:

Inter-community collaboration: by enabling sharing among different communities, we enable different communities with the common interest in a tool to get to know and collaborate with each other. A domain-independent infrastructure is required to provide the ground for shaping and representing online communities and their members around EUP tool development activities.

Reuse: once an interactivity component was built by a

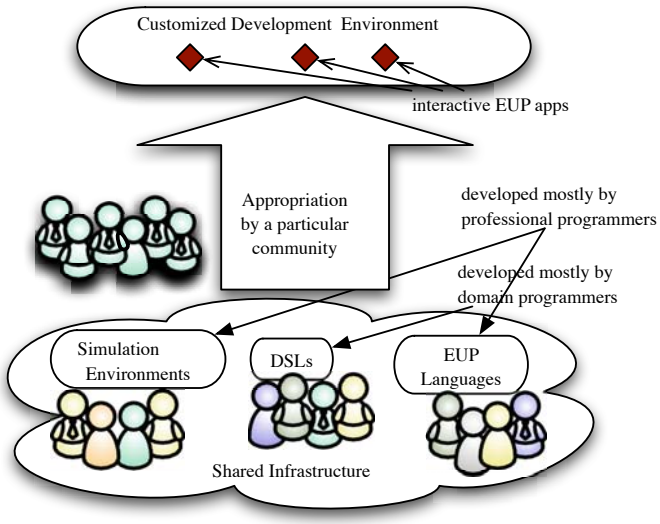

Figure 2: Shared infrastructure to support inter-domain tool construction. Different colors represent different communities.

community, it can be globally re-used by all other communities. The infrastructure has to provide hosting and search services for developed interactivity components.

Simplifying tool appropriation: as a result of collaborative development and reuse across communities we can minimize the overall development, i.e, a community will need to build only non-existing components. Especially, by modularizing required components into shared interactivity components the infrastructure can support appropriation.

Figure 2 depicts our infrastructural model. In this model the members of different communities collaborate according to their interests to build interactivity components.

4.3 Web as the Target Platform

Conceptually, in the proposed model it is the community who decides on the development tools and languages to build the interactivity components, and defines the target platform to which the end user developed programs will be compiled. However, our main goal is to support Web 2.0 communities to develop Web-based interactive artifacts to support the evolution of the Web towards the Web of applications. Therefore, we focus on the Web as the target platform, i.e., the compiler transforms the end user developed programs to their Web-compliant equivalent executables that runs inside a Web browser.

We base our developments on exploiting native Web standards, and mostly JavaScript as the assembly language of the Web platform, both for the target language to which developed programs will be compiled, and for creating the EUP development environment.

By exploiting JavaScript as the target language, we avoid using third-party drivers such as Java applets or Flash, and browser plugins. As a result, the running overhead applied by the third-party drivers will be removed, and the developed applications will be platform and browser independent. Furthermore, we consider the community to develop the compiler in the client-side Web-based languages, i.e., JavaScript and AJAX, to provide a responsive development environment to the end user to build artifacts.

By using JavaScript to create and EUP IDE for developing interactive applications: (i) we will be able to exploit the Web as the distributed collaborative infrastructure to extend the EUP development environment to a collaborative development environment, leveraging the collaborative aspects of end user programming inside communities; and (ii) we provide client-side compilation, rather than server-side compilation, of end users' programs, which brings up several advantages including 1) providing an interactive and responsive compiler capable of immediate compilation of applications without need to send the modified programs to the server for compilation; 2) transferring the compilation process from the server to the clients, distributing the compilation load among clients, avoiding the server to become the bottle-neck; and thus preserving the scalability of the system; 3) mobile users with no network connection can continue their development without need for a server.

4.4 Development Process

To realize the suggested model, we plan to develop an experimental end user programming infrastructure for online communities to develop tools for interactive domains. We divide our development process to small prototypes that we build and connect each other in a bottom-up fashion according to the following phases: (i) to investigate the feasibility of carrying EUP tools and environments for interactive applications to the Web, we have developed a Web-based engine called Ristretto$^{\text{Mobile}}$ which compiles AgentSheets applications, a well-known EUP environment, into native Web applications[1]; Ristretto$^{\text{Mobile}}$ consists of a Web-based compiler for programs written originally in AgentSheets using Visual AgenTalk[22] which is a graphical rule-base language, the equivalent JavaScript programs for AgentSheets instruction set, and a Web-based simulator that simulates the worksheets. According to our experiments, current technologies, i.e., native Web standards and existing Web browsers, allow rendering a reasonable amount of interactivity efficient enough to execute EUP applications on the Web. (ii) Ristretto$^{\text{Mobile}}$ only provides the compiler, with no authoring tool. To shape an EUP IDE on the Web we plan to develop e Web-based an extensible programming environment first to support developing applications in Visual AgenTalk for Ristretto$^{\text{Mobile}}$ and gradually move our prototype towards a generic environment which can contain languages developed by communities; (iii) Based on our experiences with developed prototypes, we develop a repository with the capability of inserting new interactivity components, including our developed prototypes. (iv) We plan to develop facilities for domain appropriation, where community members could choose from already existing interactivity components inside the infrastructure, put them together and shape EUP tools requested by domain experts. The infrastructure helps communities in bundling chosen components into a customized development environment for end users, and takes care of dependencies. (v) We plan to build facilities into the infrastructure to support collaboration among community members to develop new interactivity components and embed them inside the infrastructure.

Upon developing the experimental infrastructure, we plan to evaluate two different aspects of the system: 1) testing the suitability of the infrastructure to support different domains. We plan to develop few interactivity components and

embed them into the infrastructure as the proof of concept. 2) User/community studies on carrying domain construction/appropriation using the infrastructure.

5. CONCLUSIONS

In this research we aim at support communities of typical Web users to program interactive applications collaboratively in the Web 2.0 framework. We suggest a community-based approach for developing domain-specific EUP tools since: (i) Web 2.0 communities consist of mostly end users, demanding EUP tools for developing interactive applications, (ii) the targeted domain of programming is not defined in such a large communities of interest, therefore every community needs tools suitable for programming in their own domain; We suggest a two-layer model to support community-based development of EUP tools, in which the bottom layer provides an infrastructure for inter-domain collaboration for developing tools and languages for building interactive applications, and the top layer supports appropriation of tools developed inside the infrastructure to shape a domain-oriented development environment for end users of a particular community. We aim at using the Web, as the target platform on which the end user developed applications will be executed.

6. REFERENCES

[1] N. Ahmadi, A. Repenning, and A. Ioannidou. Collaborative end-user development on handheld devices. *Visual Languages and Human-Centric Computing*, Jan 2008.

[2] O. Beletski. End user mashup programming environments. 2008.

[3] C. Bogart, M. Burnett, A. Cypher, and C. Scaffidi. End-user programming in the wild: A field study of coscripter scripts. *IEEE Symposium on Visual Languages and Human-Centric Computing, 2008. VL/HCC 2008*, pages 39–46, 2008.

[4] M. Bolin. End-user programming for the web. 2005.

[5] M. Bolin, M. Webber, P. Rha, T. Wilson, and R. Miller. Automation and customization of rendered web pages. *UIST '05: Proceedings of the 18th annual ACM symposium on User interface software and technology*, Oct 2005.

[6] A. Clement. Cooperative support for computer work: a social perspective on the empowering of end users. *Proceedings of the 1990 ACM conference on Computer-supported cooperative work*, pages 223–236, 1990.

[7] G. Fischer and E. Giaccardi. Meta-design: A framework for the future of end user development. *End User Development*, Jan 2006.

[8] M. Gantt and B. Nardi. Gardeners and gurus: patterns of cooperation among cad users. *CHI '92: Proceedings of the SIGCHI conference on Human factors in computing systems*, Jun 1992.

[9] C. Kelleher and R. Pausch. Lowering the barriers to programming: A taxonomy of programming environments and languages for novice programmers. *Computing Surveys (CSUR*, 37(2), Jun 2005.

[10] G. Leshed, E. Haber, T. Matthews, and T. Lau. Coscripter: automating & sharing how-to knowledge in the enterprise. *CHI '08: Proceeding of the twenty-sixth annual SIGCHI conference on Human factors in computing systems*, Apr 2008.

[11] J. Lin, J. Wong, J. Nichols, A. Cypher, and T. Lau. End-user programming of mashups with vegemite. *IUI '09: Proceedingsc of the 13th international conference on Intelligent user interfaces*, Feb 2009.

[12] W. Mackay. Patterns of sharing customizable software. *CSCW '90: Proceedings of the 1990 ACM conference on Computer-supported cooperative work*, Sep 1990.

[13] A. MacLean, K. Carter, L. Lövstrand, and T. Moran. User-tailorable systems: pressing the issues with buttons. *Proceedings of the SIGCHI conference on Human factors in computing systems: Empowering people*, Jan 1990. Cited for "Gentle Slope".

[14] T. Malone, K. Grant, K. Lai, R. Rao, and D. Rosenblitt. Semistructured messages are surprisingly useful for computer-supported coordination. *ACM Transactions on Office Information Systems*, Jan 1987.

[15] M. Mernik, J. Heering, and A. Sloane. When and how to develop domain-specific languages. *ACM Computing Surveys*, 37(4):316–344, 2005.

[16] A. Mørch and N. Mehandjiev. Tailoring as collaboration: The mediating role of multiple representations and application units. *Computer Supported Cooperative Work (CSCW)*, Jan 2000.

[17] B. Nardi. A small matter of programming: Perspectives on end user computing. *MIT Press*, Jan 1993.

[18] B. Nardi and J. Miller. Twinkling lights and nested loops: distributed problem solving and spreadsheet development. *International Journal of Man-Machine Studies*, Jan 1991.

[19] T. O'reilly. What is web 2.0. *Design patterns and business models for the next generation of software*, 30:2005, 2005.

[20] V. Pipek and H. Kahler. Supporting collaborative tailoring. *End-User Development*, Jan 2005.

[21] T. Raman. Toward 2w, beyond web 2.0. *Communications of the ACM*, 52(2):52–59, 2009.

[22] A. Repenning and J. Ambach. Tactile programming: a unified manipulation paradigm supporting program comprehension, composition and sharing. *Visual Languages, 1996. Proceedings., IEEE Symposium on*, pages 102 – 109, Aug 1996.

[23] R. Stallman. Emacs the extensible, customizable self-documenting display editor. *ACM SigPlan Notices*, Jan 1981.

[24] J. Wong and J. Hong. Marmite: end-user programming for the web. *CHI '06: CHI '06 extended abstracts on Human factors in computing systems*, Apr 2006.

[25] J. Wong and J. Hong. Making mashups with marmite: towards end-user programming for the web. *CHI '07: Proceedings of the SIGCHI conference on Human factors in computing systems*, Apr 2007.

[26] R. Yee. Pro web 2.0 mashups: Remixing data and web services. *Apress*, 2008.