Live Mashup Tools: Challenges and Opportunities

Saeed Aghaee, Cesare Pautasso
Faculty of Informatics, University of Lugano (USI), Switzerland
saeed.aghaee@usi.ch, c.pautasso@ieee.org

***Abstract*—*Live programming* is a programming style in which the repetitive task of compiling and running the software being programmed is managed automatically. This style can be a helpful practice in End-User Development (EUD) where the non-professional end-users are to be supported through techniques and tools that empower them to create or modify software artifacts. Mashups — a form of lightweight Web applications composing reusable content and functionalities available on the Web — are a popular target for EUD activities on the Web. EUD for mashups is enabled by intuitive composition environments, called mashup tools. In this paper, we introduce *live mashup tools*, a new class of mashup tools based on the live programming style. We give a comprehensive definition and classification of live mashup tools, giving examples of how well existing tools fit in this category and discuss open research challenges and opportunities.**

***Index Terms*—Live Programming, Liveness, Web Mashups, Mashup Tools, End-User Development.**

I. Introduction

End-User Development (EUD) [1] systems aims at facilitating the creation and modification of software artifacts by non-professional users. Mashup tools are a type of EUD system targeted for mashups [2] — lightweight compositions of Web services, Web widgets and Web data sources [3]. These tools [4] provide intuitive, highly abstract composition languages targeted for non-professional users on the Web.

Live programming is a highly interactive style of computer programming in which the target software is automatically and continuously rebuilt and executed while it is being coded [5]. EUD systems, like mashup tools, can benefit from incorporating this style, as it can effectively reduce the cognitive cost of programming for beginners.More specifically, live programming style provides major benefits as follows:

- **Bridging the gulf of evaluation.** The gulf of evaluation refers to the degree of difficulty of assessing and understanding the state of the system [6]. Live mashup tools are capable of providing immediate feedback about the effect of changes to the code on the output, as a result of which the gulf of evaluation can be significantly bridged. The immediate feedback contains either: (i) the new output that provides gratification and increases the self-efficacy of the user to better face further challenges, or (ii) errors that, thank to the immediateness of the feedback, corresponds to a small action and can be easily rectified (so it may cause no or very little anxiety in end-users and encourage their explorations).
- **Enabling efficient problem solving.** Live mashup tools allow end-users to more efficiently solve complex problems in a step-by-step fashion. The actions taken in each step are tiny and tested by users with the help of the immediate, automatic feedback provided by the tool. In practice, it is more time consuming and less efficient to first create the whole mashup and then execute and test it as a whole. Preventing anxiety in end-users caused by a challenging development process is an important goal in EUD, and creating mashups in a live fashion is one way to achieve it.
- **Leading to a more gentle learning curve.** Liveness can help end-users to gradually develop their skills in using the tool. Accomplishing and then immediately validating small steps one at a time towards solving a bigger problem lead the users towards a gradual and gentle learning process, which is an important requirement in the design of EUD systems [7]. We believe there is a direct relationship between the incrementality of the development process (enabled by live programming), and the graduality and gentleness of its learning curve.

In this paper, we propose *live mashup tools* as a new category of mashup tools featuring the live programming style. Also, we provide a classification of live mashup tools based on how closely they relate and connect the design perspective with the expected output and deployment perspectives in the tool environment. Live mashup tools pose a number of challenges that are subject to be discussed in this paper. For instance, considering that mashups are compositions of remote and distributed components, it is technically very challenging to comply with the requirements of liveness, as the changes made by the user to the mashup design must be reflected in the result of the mashup execution with minimal delay. To exemplify how these challenges can be addressed in a concrete mashup tool, we present the architecture of `NaturalMash`, a novel live mashup tool.

The rest of this paper is structured as follows. In the next section, we introduce our definition of live mashup tools. Section III presents `NaturalMash`. The challenges of live mashup tools are discussed in Section IV. Related work is discussed in Section V. Conclusions are drawn in Section VI.

II. Live Mashup Tools: Definition

We characterize live mashup tools by the notion of "liveness", which was first introduced by Tanimoto [8] in the context of visual programming languages. In his paper he defined four levels of liveness corresponding to the distance between the design and output perspectives of a visual environment.

978-1-4673-6265-8/13 © 2013 IEEE LIVE 2013, San Francisco, CA, USA

Accepted for publication by IEEE. © 2013 IEEE. Personal use of this material is permitted. Permission from IEEE must be obtained for all other uses, in any current or future media, including reprinting/ republishing this material for advertising or promotional purposes, creating new collective works, for resale or redistribution to servers or lists, or reuse of any copyrighted component of this work in other works.

The design perspective is where the user creates or manipulates a software artifact (i.e., in our case the mashup). Likewise, the output perspective constitutes the result of the run-time execution of the software artifact being created or modified in the design perspective. At the lowest level of liveness, there exists only the design perspective (e.g., a blueprint), whereas at the highest level both the design and output perspectives are not only present, but also are automatically kept in sync. Live programming tools also feature the highest level of liveness. The same holds for live mashup tools, which enable mashup developers to take advantage of the live programming style and also feature the highest level of liveness.

There are three variations of live mashup tools with respect to the way the design and output perspectives are positioned in the mashup development environment (Figure 1).

- **Distinct.** This type of live mashup tools separates the design and output perspectives, in a way that the two perspectives are recognizable from each other. It should be noted that the expected output perspective might be integrated into the same user interface as the design perspective or not. The former case has the advantage of allowing the tool environment to be more self-contained. In the latter case, however, the output perspective resides in the actual target environment, in which the output mashups are to be deployed and executed. The output perspective is thus remotely connected to the design perspective that is the main part of the mashup tool environment. For instance, the output perspective may be integrated with a Web browser (e.g., FireFox) that gets connected to the mashup tool environment using a plug-in. The advantage of a realistic output perspective is that it shows not only the final execution of the mashup being created, but also its final deployment environment. The *distinct* variation is exemplified by MashupStudio [9], in which the environment interface only consists of a visual editor (design perspective) that is remotely connected to the output perspective (a mobile client) . In general, separating the design and output perspectives results in more freedom for mashup tool designers to choose among various interaction techniques for mashup programming [4] (e.g., visual wiring languages, textual languages, etc.). Still, this separation imposes more barriers on the users' side, as they need to be able to distinguish and switch between the two perspectives. This is specially the case when the output perspective is remotely connected to the tool environment.
- **Coincident.** This variation comes in effect when the design and output perspectives of the live mashup tool environment completely fold over each other, in a way that they are not distinguishable anymore. This is synonym to What You See Is What You Get (WYSIWYG), where the interface shown to the user allows to see and manipulate content (e.g., graphical object, text, etc.) that is very similar to the output mashup begin created. Examples are RoofTop [10], Lively Wiki [15], and Intel MashMaker [11]. The main advantage of this variation is the direct manipulation feedback governed by the WYSIWYG interface. From the mashup tool design perspective, Its major shortcoming is the inability of choosing other interaction techniques than WYSIWYG for mashup programming. WYSIWYG interfaces since they mainly target the user interface of the mashup are unable to provide adequate expressive power for manipulations of the business logic and data integration layers of the mashup being created.

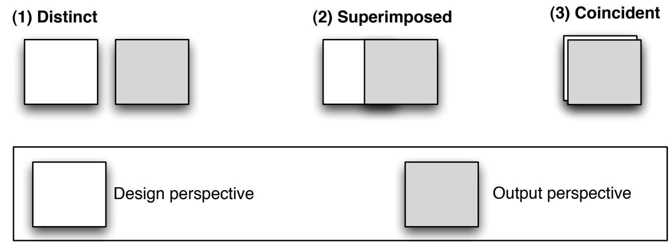

Fig. 1. A classification for live mashup tools.

- **Superimposed.** In this type, part of, or the whole of, the output perspective is superimposed with the design perspective, but does not cover it completely. The output perspective is basically based on WYSIWYG (similar to the *coincident* variation) that has the advantage of allowing direct manipulation. In contrast to the *coincident* variation, the design perspective is not limited to the WYSIWYG part (the output perspective). This allows to use other interaction techniques in the design perspective to further compensate the shortcoming of WYSIWYG in business logic creation. Analogous to the *distinct* live mashup tools, the output can be integrated or remotely connected. In other words, *superimposed* live mashup tools take the best of the other two variations (*coincident* and *distinct*) and apply them to their own shortcomings. As an example, `NaturalMash`, which will be described in this paper, has been designed following the *superimposed* variant.

Another classification of live mashup tools can be based on how they deal with *live data sources* (e.g., web feeds). Accordingly, live mashup tools either update the output perspective as the live data sources update (e.g., [12]) or not.

III. NaturalMash: A Live Mashup Tool

`NaturalMash` [13] (Figure 2) is an intuitive live mashup tool classified as *superimposed*. In terms of live data sources, It is also capable of updating the output mashup as the underlying live data sources update. In terms of interaction technique, `NaturalMash` is based on structured natural language programming and What You See Is What You Get (WYSIWYG). The advantage of using natural language lies in its understandability, which further shortens and simplifies the learning curve.

The intuitive environment of `NaturalMash` is composed of four main elements: (i) *visual field*, where the editable output perspective resides, (ii) *text field*, where the natural language description of the mashup is edited with the help of a strong autocomplete feature, (iii) *component dock*, where the list of components used in the mashup being edited is displayed, and (iv) *stack*, which contains the ingredients bar (component library), the mashups created by the users, and personalized information that can be used within mashups (e.g., user's location, user's tweets, etc.).

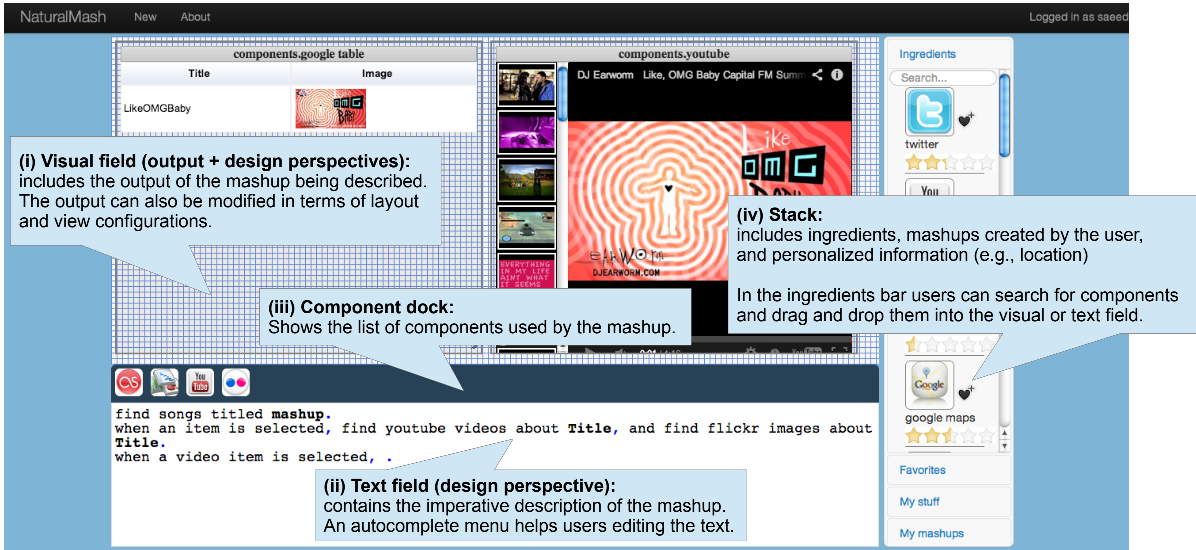

Fig. 2. `NaturalMash` interface incorporates both the design (text and visual fields) and output (visual field) perspectives.

`NaturalMash` also supports synchronized multi-perspective interaction. The three main interaction components of the environment (component dock, text field, and visual field) are all kept synchronized during every user interaction (e.g., hovering mouse over component icons in the dock highlights its corresponding widgets in the WYSIWYG output and descriptions in the text field). We believe synchronized multi-perspective modeling is a necessary feature to be integrated within live mashup tools that distinguish between the design and output perspectives (i.e., *distinct* and *superimposed* variations).

Another interesting feature of `NaturalMash` is the use of Programming by Demonstration [14] (PbD) in the visual field that results in getting suggestions in the text field. For instance, interacting with a widget in the visual field results in the text field showing suggestions for the corresponding user interface event (e.g., click event) triggered by the user's interaction.

The user studies we conducted with non-programmers revealed the benefits of liveness we outlined in the beginning of this paper. For instance, we observed the skills of users were gradually improving as they could successfully accomplish more challenging tasks each time. However, discussing the user studies in detail is beyond the scope of this paper.

IV. Challenges

Despite the benefits of live mashup tools, their design and development remain challenging:

- **Minimizing the live response time.** End-users may be detecting latency in the execution response of a live mashup tool because of a variety of factors. Some of these factors are concerned with the real performance of the tool. It means how fast the tool can compile and execute the mashup being created. There are also factors that deal with the way the live execution response time information is managed and provided at the user interface level. These refers to the perceived performance of the tool. Failing to minimize the live response time through increasing both real and perceived performance of live mashup tools may cause anxiety in their end-users.

`NaturalMash` incorporates a high performance architecture that considerably decreases the live response time of `NaturalMash`. The main components of this architecture include: (i) *incremental change compiler*, that incrementally compile the new changes made to the mashup being edited, as opposed to recompiling the whole mashup, (ii) *compiler cache*, that further boosts up the incremental compiler through the use of a `cache` storing all the previous target models and their corresponding generated code, and (iii) *run-time cache*, that enhances the mashup execution life-cycle.

Also, to maximize the perceived performance of `NaturalMash`, we adopted the following guidelines: (i) enable waiting control mechanism after a reasonable time, (ii) show mashup user interface after the complete load, (iii) do not prevent users from interacting with the mashup while it is being compiled, and (iv) make user aware of internet connection errors.

- **Coping with service call rate limits.** The majority of free mashup components introduce strict call rate limits. For instance, Twitter API (https://dev.twitter.com/), which is one of the most popular mashup components, allows 150 calls per hour. Moreover, many existing components still issues developer keys to identify the source of calls. Within a mashup tool platform (e.g., `NaturalMash`), a component is usually developed and shared by one person (there is one developer key), whereas it might be used by many users at the same time. These issues may pose serious challenges on the feasibility of building mashups in a live fashion. This is because the live mashup development process requires continuous execution of the same mashup, and thus there is a risk of exceeding the call rate limit of the constituents components.

Run-time and compiler caching adopted by NaturalMash architecture allows partial execution of the mashup, which further addresses service call rate limit problem. Also, the NaturalMash engine is aware of components with call rate limit from the meta-data already provided in the NaturalMash component model. Therefore, if a component is about to reach its limit, NaturalMash force-freezes its reexecution. Also, we advocate component providers to use OAuth (http://oauth.net/), as opposed to developer keys, to identify the source calls. The advantage is that, the call limit is uniquely allocated to each user of the mashup tool and is not shared by all the users.

- **Making the most of the provided screen space.** The advantage of live programming is even more pronounced when both the output and design perspectives are visible at the same time. Therefore, a tab-based environment might not be a proper design option. Depending on the interaction techniques utilized by the live mashup tools, it might become challenging to integrate both the design and output perspectives into the same user interface without compromising the usability of the tool. For instance, a visual wiring language alone (design perspective) requires a relatively large space for diagrams to be understandable. To cope with the limited screen space problem, in NaturalMash we took advantage of the compactness and monodimensionality of text. Also the output perspective can be maximized at any time to give further space to work with the WYSIWYG output.

V. Related Work

Supporting live programming in general EUD was first encouraged by [7]. In the mashup area, the trend towards live mashup tools was first predicted in our previous work [4]. Intel MashMaker [11] was among the first mashup tools that went "live". However, non of the previous work completely studied "live mashup tools" as a research topic. We believe this work provides a roadmap towards further research and development in this area.

Yahoo! Pipes (http://pipes.yahoo.com/) and Microsoft Popfly (shut down in 2009) are/were among the first-generation mashup tools based on the wiring interaction technique. NaturalMash uses a hybrid interaction technique combining natural language programming, WYSIWYG, and PbD. Related to NaturalMash are also WYSIWYG spreadsheets tailored for mashup development [16]. These can also be classified as *superimposed*. Instead of expressions and formulas, NaturalMash uses natural language to give a description of the composition logic.

VI. Conclusion

EUD tools, such as mashup tools, can largely benefit from the live programming style. To this end, we introduced live mashup tools as a class of mashup tools adopting the live programming style. We provided a classification of them, enumerating how some existing mashup tools fit into the proposed categories (*distinct*, *coincident*, and *superimposed*). We outlined the challenges and opportunities for further research in the area of live mashup tool and exemplified them in the context of NaturalMash, a live mashup tool we have been building in the past few years with promising feedback from its user community. We expect to see more and more mashup tools adopt the live programming style in the future.

All in all, we believe liveness is an important feature that should be integrated in any kind of programming environment, from end-user programming to professional programming [17]. Most of the concepts presented in this paper can also be generalized to general purpose programming.

Acknowledgements

The work presented in this paper has been supported by the Swiss National Science Foundation with the SOSOA project (SINERGIA grant nr. CRSI22_127386).

References

[1] H. Lieberman, F. Paternò, M. Klann, and V. Wulf, "End-User Development: An Emerging Paradigm," in *End User Development*. Springer Netherlands, 2006.

[2] F. Casati, "How End-User Development Will Save Composition Technologies from Their Continuing Failures," in *Proc. of IS-EUD 2011*, 2011.

[3] D. Benslimane, S. Dustdar, and A. Sheth, "Services Mashups: The New Generation of Web Applications," *IEEE Internet Computing*, vol. 12, pp. 13–15, 2008.

[4] S. Aghaee, M. Nowak, and C. Pautasso, "Reusable Decision Space for Mashup Tool Design," in *Proc. of EICS 2012*, 2012.

[5] N. Collins, A. McLean, J. Rohrhuber, and A. Ward, "Live Coding in Laptop Performance," *Org. Sound*, vol. 8, pp. 321–330, 2003.

[6] D. A. Norman and S. W. Draper, *User Centered System Design; New Perspectives on Human-Computer Interaction*. L. Erlbaum Associates Inc., 1986.

[7] A. Repenning and A. Ioannidou, *What Makes End-User Development Tick? 13 Design Guidelines*. Springer Verlag, 2006.

[8] S. L. Tanimoto, "VIVA: A visual language for image processing," *J. Vis. Lang. Comput.*, vol. 1, pp. 127–139, 1990. [Online]. Available: http://dx.doi.org/10.1016/S1045-926X(05)80012-6

[9] J. Yang, J. Han, X. Wang, and H. Sun, "MashStudio: An On-the-fly Environment for Rapid Mashup Development," in *Internet and Distributed Computing Systems*. Springer Berlin Heidelberg, 2012.

[10] V. Hoyer, F. Gilles, T. Janner, and K. Stanoevska-Slabeva, "SAP Research RoofTop Marketplace: Putting a Face on Service-Oriented Architectures," in *Proc. of SERVICES 2009*, 2009. [Online]. Available: http://dl.acm.org/citation.cfm?id=1590963.1591531

[11] R. Ennals, E. Brewer, M. Garofalakis, M. Shadle, and P. Gandhi, "Intel mash maker: join the web," *SIGMOD Rec.*, vol. 36, pp. 27–33, 2007.

[12] G. Tummarello, R. Cyganiak, M. Catasta, S. Danielczyk, R. Delbru, and S. Decker, "Sig.ma: Live views on the Web of Data," *Web Semantics: Science, Services and Agents on the World Wide Web*, vol. 8, pp. 355 – 364, 2010.

[13] S. Aghaee and C. Pautasso, "EnglishMash: usability design for a natural mashup composition environment," in *Proc. of ComposableWeb 2012*, 2012.

[14] A. Cypher, D. C. Halbert, D. Kurlander, H. Lieberman, D. Maulsby, B. A. Myers, and A. Turransky, Eds., *Watch what I do: programming by demonstration*. MIT Press, 1993.

[15] R. Krahn, D. Ingalls, R. Hirschfeld, J. Lincke, and K. Palacz, "Lively Wiki A Development Environment for Creating and Sharing Active Web Content," in *Proc. of WikiSym 2009*, 2009. [Online]. Available: http://doi.acm.org/10.1145/1641309.1641324

[16] D. D. Hoang, H.-y. Paik, and B. Benatallah, "An Analysis of Spreadsheet-based Services Mashup," in *Proc. of ADC 2010*, 2010.

[17] A. J. Ko, R. Abraham, L. Beckwith, A. Blackwell, M. Burnett, M. Erwig, C. Scaffidi, J. Lawrance, H. Lieberman, B. Myers, M. B. Rosson, G. Rothermel, M. Shaw, and S. Wiedenbeck, "The State of the Art in End-user Software Engineering," *ACM Comput. Surv.*, vol. 43, pp. 21:1–21:44, 2011.