ICWE 2015 Rapid Mashup Challenge: Introduction

Florian Daniel[1] and Cesare Pautasso[2]

[1] University of Trento, Via Sommarive 9, 38123 Povo (TN), Italy
daniel@disi.unitn.it
[2] Faculty of Informatics, University of Lugano (USI), Switzerland
{name.surname}@usi.ch

Abstract. The ICWE 2015 Rapid Mashup Challenge is the first installment of a series of challenges that aim to engage researchers and practitioners in a competition for the best mashup approach. This paper introduces the reader to the general context of the Challenge, its objectives and motivation. It summarizes its structure into phases and the requirements contributions were asked to satisfy, so as to be eligible for participation. A brief summary of the contributions that were selected for presentation provides an overview of the content of the remainder of this volume.

Keywords: Mashups, Mashup tools, Challenge, Benchmarking

1 Context and Objective

By now, it's more or less a decade that the scientific community and industry use the term "mashup" in the context of Web engineering to refer to a type of Web application that heavily relies on the reuse of third-party constituent elements in their development. In fact, we usually define a mashup as "a composite application developed starting from reusable data, application logic and/or user interfaces typically, but not mandatorily, sourced from the Web" [1]. Mashups have been associated with their situational nature and the serendipitous reuse of components not necessarily originally intended for the purpose of the mashup. That is, the term mashup refers more to the way applications are developed and less to a specific type of application as perceived by its users – in the end we are always talking about a Web application.

The ICWE 2015 Rapid Mashup Challenge (the Challenge, `http://mashup.inf.usi.ch/challenge/2015`) acknowledges this peculiarity of mashups and puts its focus on the techniques, approaches, libraries, and tools that researchers and practitioners have come up with so far to aid the development of mashups. This perspective is different from the perspectives of similar challenges known from other contexts or communities. For instance, the Semantic Web Challenge (`http://challenge.semanticweb.org/`) focuses on the application of Semantic Web [2] technologies in the development of software with commercial potential, large user bases, or functionality that is useful and of societal value. The AI Mashup Challenge (`http://aimashup.org/`), instead, more specifically focuses on mashups that use AI (Artificial Intelligence) technology (e.g., machine learning and data mining, machine vision, natural language processing, reasoning, ontologies) and intelligence to mashup existing resources. The ICWE 2015 Rapid Mashup Challenge does not limit its focus to any specific technology and rather aims to compare how mashups are developed, independently of how their internals look like.

The maturity and sophistication of mashup tools/approaches has been growing over the past decade. Many research projects and industry tools have been dedicated to design and develop tools for the composition of Web services, Web data sources and Web widgets. Given their diversity, comparing and evaluating mashup approaches has been very challenging. Doing so from a practice-oriented point of view is the goal of the Challenge, of course, while still keeping also an eye on the quality and usefulness of the mashups developed during the Challenge. The only constraints we introduced concerned the required use of a set of representative Web APIs and the strict time limit of 10 minutes for the construction of the mashup itself.

In the following, we describe all the pre-challenge aspects, such as the Call for Participation, the requirements candidate approaches had to satisfy, the organization of the Challenge itself, as well as the set of selected competitors. The following articles in this volume describe each of the contributions individually and report on the mashups developed live during the Challenge using these tools. The last article in this volume then provides insight into the voting procedure and tool and the outcomes of the Challenge, including the winner.

2 Participation Requirements and Organization

2.1 Call for Participation and Requirements

In line with the above goals of the Challenge, this year's call for participation started as follows:

> *The ICWE 2015 Rapid Mashup Challenge launches a competition between mashup approaches/tools with special attention to their expressiveness and speed. We invite developers and researchers working on mashups, mashup tools and assisting technologies to compete in the creation of the most interesting and/or complex mashup they can develop within a given time boundary, using a given set of source components. The goal of the Challenge is to allow everybody working on mashups and composite Web applications to showcase their ideas and solutions and to establish an event that is both challenging and fun.*
>
> *We are interested in all kinds of mashup composition tools and approaches: from programming languages, domain-specific languages to natural language, from visual modeling tools to textual ones, etc. Submissions will be screened based on relevance, originality and maturity. Admitted contributions will be evaluated as follows: Points will be given by*

a jury for the complexity of the resulting mashup, the elegance of its construction and the features of the mashup tool/approach that have been used to build it. The public will also be able to give feedback and participate in the challenge evaluation process.

The call highlights the three key aspects of the evaluation:

- *Complexity of mashup*: The key criterion of any development environment is of course the quality of the output it is able to generate. In the case of mashup tools/approaches, this output are the mashups. During the Challenge, participants were therefore asked to showcase live the development of a mashup using their own tool/approach, whose complexity and quality as application was assessed.
- *Elegance of construction*: Talking about aiding the development of software, it is important to look at how this aid is implemented. The elegance of construction, in this respect, refers to how easy the proposed mashup tool/approach is perceived, how efficient the jury and audience think the tool is compared to the state of the art, and which benefits it provides to its users.
- *Features of mashup approach*: Finally, since a single mashup may not be able to showcase all the features of a proposed tool/approach, it is also good to have a look at which exact development features it provides. For instance, a tool that is oriented toward professional programmers is fundamentally different from one that instead targets end-users.

As for the kind of mashup tool/approach that was considered eligible to participate, the Challenge was very open, and all kinds of mashup composition tools and approaches were allowed: from programming languages, domain-specific languages to natural language and visual modeling tools. The limitation was only the imagination and creativity of the participants. The complexity of mashups and the features of the mashup tool/approach were self-declared by the authors using a dedicated feature checklist (see Section 3) and assessed by the jury and the audience during the Challenge.

2.2 Structure of Challenge

The Challenge was organized into four phases:

1. *Admission*: Submission of application. The application should include a brief description of the proposed tool/approach and a filled feature checklist.
2. *Preparation*: If a proposal was accepted to the challenge, the authors received a list of Web APIs that are allowed to be used to compose the demo mashup during the competition. This preparation phase gave the authors about one month to prepare for the event.
3. *Competition*: During the ICWE conference, participants had to give a live demonstration of how you build their own mashup within at most 10 minutes of time, preceded by a 10-15 minutes presentation of their approach and preparation for the Challenge. The time limits made the challenge more challenging.

4. *Post-challenge*: Preparation of post-challenge paper explaining the proposed solution and giving technical details about the approach and how it was used to rapidly build the mashup.

The goal of this structure was to have authors focus more on the practical aspects before the Challenge (the preparation of their demonstration), while asking them to concentrate on the conceptual and scientific aspects afterwards (with the writing of the paper to be included in the proceedings). Submitted applications for participation were evaluated by the organizers of the Challenge based on the relevance and maturity of the proposed approach.

3 Feature Checklist

In order to facilitate the comparison of approaches, authors were required to accompany their submission with a filled feature checklist that describes the two key parts of the evaluation, i.e., the nature of the mashups that their tool/approach allows one to develop and the development features of the proposed tool. Figure 1 graphically summarizes the features identified as relevant for the Challenge, while the following subsections describe the features in more detail.

3.1 Mashup Features

In order to be able to compare the mashups produced by the different approaches during the Challenge, the mashup features proposed by Daniel and Matera [1] were taken as reference:

– **Mashup type:** The mashup type expresses the positioning of the mashup at one or more of the three layers of the typical application stack (data, logic, presentation), depending on where the mashup's integration logic is positioned. *Data mashups* operate at the data layer, integrate data sources, and are typically published again as data sources (e.g., RSS feeds or RESTful Web services). *Logic mashups* integrate components at the application logic layer, reuse data and application logic (e.g., Web services), and are typically published as Web services. *User Interface (UI) mashups* are located at the presentation layer, integrate UI components/widgets, and are published as Web applications that users can interact with via the Web browser. Finally, *hybrid mashups* span multiple layers of the application stack.
– **Component types:** The types of mashups introduced above strongly relate to the types of the components they integrate. *Data components* comprise RSS and Atom feeds, XML JSON, CSV and similar data resources, web data extractions, micro-formats, but also SOAP or RESTful web services that are used as data services only. *Logic components* comprise SOAP and RESTful web services, JavaScript APIs and libraries, device APIs, and API extractions. *UI components* comprise code snippets and JavaScript UI libraries, Java portlets, widgets and gadgets, web clips and extracted UI components.

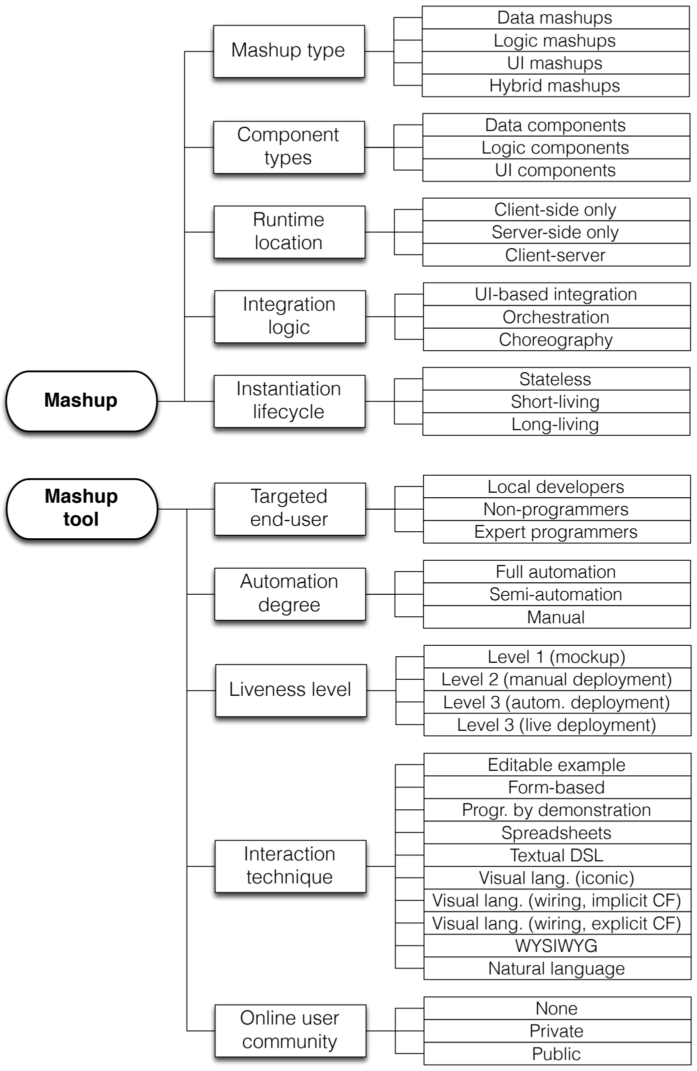

Fig. 1: The feature checklist used to compare and position mashup approaches.

- **Runtime location:** There are generally a variety of possible architectural configurations that may be adopted for the development of mashups, compatibly with the requirements of the chosen components. *Client-side* mashups are executed in the client browser. *Server-side* mashups are executed in the server. *Client-server* mashups are distributed over client and server, and both parts interact the one with the other at runtime.
- **Integration logic:** The integration logic tells how integration happens, that is, how components are used to form a composite application and how they

are enabled to communicate with each other (if at all). *UI-based integration* applies exclusively to UI components and uses the graphical layout of the mashup's user interface to render UI components in parallel next to each other inside one or more web pages. *Orchestrated integration* applies to all kinds of components and consists in a centralized composition logic. *Choreographed integration* is for all those types of components that are able to comply with a given convention (oftentimes also called a contract or protocol), so as to manage integration without a central coordinator.
– **Instantiation lifecycle:** The last aspect of mashups considered is how long an instantiated mashup is running. *Stateless mashups* do not require keeping any internal state for their execution and end after processing. *Short-living* mashups are mashups that last the time of a user session, i.e., as long as a user is interacting with the mashup in the client browser, and terminate with the closing of the client browser. *Long-living* mashups may last longer than a user session, that is, they survive even after the user closes the browser with the rendered mashup or after the first invocation of the mashup.

These five features allow one to easily classify mashups and to assess their internal complexity. Of course, this is not an exhaustive list of characteristics and many other distinguishing features could be examined [1]. Yet, for the sake of assessing the suitability and interestingness of approaches in this first version of the Challenge we considered these five features as enough.

3.2 Mashup Tool Features

The comparison of the features of the mashup tools/approaches was instead based on the work by Aghaee et al. [3].

– **Targeted end-user:** Determining which group of users is targeted by a mashup tool/approach is a strategic design issue decided on by the developers. *Non-programmers* do not have programming skills. Yet, they may be interested in creating mashups as long as it does not require them to learn and use a programming language. *Local developers* are those non-programmers who usually have advanced knowledge in computer tools. *Expert programmers* have adequate programming skills and experience to develop mashups using programming and scripting languages (e.g., JavaScript and PHP).
– **Automation degree:** The automation degree of a mashup tool refers to how much of the development process can be undertaken by the tool on behalf of its users. *Full automation* of mashup development eliminates the need for direct involvement of users in the development process. *Semi-automatic* tools partially automate mashup development by providing guidance and assistance. *Manual* approaches do not provide any automated support during development; typically, these approaches come in the form of programming libraries or runtime middlewares.
– **Liveness level:** Tanimoto proposed the concept of liveness [4], according to which four levels of liveness can be distinguished. At *Level 1* (non-executable

prototype mockup), a tool is just used to create prototype mashups that are not directly connected to any kind of run-time system. *Level 2* (explicit compilation and deployment steps) of liveness is characterized by mashup design blueprints that carry sufficient details to give them an executable semantics. *Level 3* (automatic compilation and deployment) tools support rapid deployment into operation, e.g., triggered by each edit-change or by an explicit action executed by the developer. *Level 4* (dynamic modification of running mashup) of mashup liveness is obtained by the tools that support live modification of the mashup code, while it is being executed.

– **Interaction technique:** There have been a number of interaction techniques through the use of which the barriers of programming can be lifted to its developers [5]. *Editable examples* let users modify and change the behavior of existing examples, instead of programming from scratch. In *form-based* interaction, users are asked to fill out a form to create a new or change the behavior of an existing object. *Programming by demonstration* suggests to teach a computer by example how to accomplish a particular task. *Spreadsheets* are one of the most popular and widely used end-user programming approaches to store, manipulate, and display complex data. *Textual DSLs* are languages targeted to address specific problems in a particular domain; they have a textual syntax that may or may not resemble an existing general-purpose programming language. A *visual language* (iconic), as opposed to a textual programming language, is any programming language that uses visual symbols, syntax, and semantics. Some visual languages support *wiring with implicit control flow*, where the control flow of the mashup is derived from its data flow graph. Other visual languages support *wiring with explicit control flow*, where the control flow is explicitly defined, for instance, by adding directed arrows connecting the boxes, or putting the boxes in a specific order (e.g., from left to right). *WYSIWYG* (What You See Is What You Get) enables users to create and modify a mashup on a graphical user interface that is similar to the one that will appear when the mashup runs. *Natural language* allows developers to express their mashup via a restricted, controlled set of natural language constructs (e.g., a subset of English) that can be interpreted unequivocally by a runtime environment.
– **Online user community:** Online communities are an important resource in assisting developers, especially end-users, to program [6]. If a tool does not support any online community (*none*), it is harder to leverage on the experience of others. In *public* communities, the content is accessible to any user on the Web who wishes to join the community (with or without registration). In *private* communities, the authority to join the community is granted on the basis of compliance with some operator-specified criteria.

Like for the mashup features, also in the case of the mashup tools/approaches many other characteristics could be considered (e.g., collaboration). The features selected for the Challenge, however, already provide good insight into the philosophy behind each approach, and we preferred to keep the list concise.

4 Participants

The purpose of the above feature checklist with its 10 features is threefold: firstly, it allows interested participants to understand what kind of contributions the Challenge is interested in; secondly, it allows the organizers of the Challenge to pre-screen contributions and select submissions for inclusion in the Challenge and proceedings; and, thirdly, it allows the participants to better position their contributions and the jury and audience to better compare the contributions. The first two steps led to the following list of participants to the ICWE 2015 Rapid Mashup Challenge (we postpone the discussion of the jury/audience assessment to the concluding article of this volume):

- **FlexMash**: Extended Techniques for Flexible Modeling and Execution of Data Mashups, by Pascal Hirmer and Bernhard Mitschang. FlexMash is a visual mashup tool for the development of data mashups that targets non-programmers. The tools pays particular attention to flexibility and extensibility to enable the integration of heterogeneous data sources as well as the dynamic (un-)tethering of data sources. The authors participate with a prototypical implementation of their tool.
- **UI-Oriented Computing**: Interactive, Live Mashup Development through UI-Oriented Computing, by Anis Nouri and Florian Daniel. UI-oriented computing is less an individual mashup tool and more a novel idea of programming paradigm that looks at the Surface Web as at a programming environment and aims to support interactive and live mashup development inside the Web browser, without requiring users to program any line of code. The authors participate in the Challenge with a prototype implementation of a Web browser extensions that extends the browser with UI-oriented computing capabilities.
- **SmartComposition**: Extending Web Applications to Multi-Screen Mashups, by Michael Krug, Fabian Wiedemann and Martin Gaedke. SmartComposition takes mashups to a different level by proposing an environment based on Web components that supports the development of multi-screen mashups, that is, mashups that are naturally distributed over multiple devices. Web sockets allow the environment to synchronize components across screens. The authors participate with a prototype environment with support for dynamic runtime modifications.
- **EFESTO**: A platform for the End-User Development of Interactive Workspaces for Data Exploration, by Giuseppe Desolda, Carmelo Ardito and Maristella Matera. EFESTO is a platform for the creation of interactive workspaces supporting end-users in the exploration and seamless composition of heterogeneous data sources. Internally, it makes use of Linked Open Data, so as to provide its users with advanced data integration features almost for free. The authors showcase their current implementation of development environment in the form of a workspace for integrating UI components and data sources.

– **WebMakeup**: Empowering Users to Mod Websites, by Oscar Diaz, Iñigo Aldalur, Cristobal Arellano, Haritz Medina and Sergio Firmenich. Also WebMakeup proposes an original perspective on the problem of mashup development: instead of proposing an own, new development environment, it leverages on the Web browser and the applications running therein as natural environment for the modding (client-side extension) of existing applications (e.g., by adding widgets that fetch data from other applications). The authors participate with their publicly available Chrome extension.
– **WLS**: Mashup Development with Web Liquid Streams, by Masiar Babazadeh, Andrea Gallidabino and Cesare Pautasso. Finally, Web Liquid Streams (WLS) delves into one peculiar aspects of modern mashups, i.e., streaming data. The approach enables the development of mashups with streaming operators that support the live, runtime integration of data streams. The authors showcase the use of their dynamic streaming framework that takes advantage of standard Web protocols and targets expert programmers.

Table 1 summarizes the characteristics of the selected approaches as declared by the authors. Compared to the emergence of mashups, the approaches represent well the recent focus of the mashup community on the user interface side of mashups. In fact, UI mashups are widely considered most suitable for end-users without programming skills, and end-users have been in the mind of mashup tool developers from the very beginning on. Thanks to the availability of stable JavaScript communication technologies, such as AJAX, it is also evident, that more and more approaches enable the development of full-fledged, client-server mashups whose resource consumption can strategically be distributed over client (e.g., for UI synchronization) and server (e.g., for data integration). Interestingly, most of the proposed approaches feature mashups with a short-lived lifecycle, that is, mashups that run inside the Web browser as long as the browser is open.

On the development support side, a preference for dynamic, live development approaches (level 4) is evident – again, in line with the latest trends in end-user development. The paradigms proposed to approach development (the interaction techniques) are, instead, very heterogeneous and led to a very varied and diversified live demo session during the Challenge. The degree of automation is mostly that of semi-automation, while only one tool (WebMakeup) already has an own online user community. This last result is strictly related with the early stage of development (prototypes) of most of the proposed approaches.

We believe the selected mashup approaches represent a vivid and cutting-edge picture of the state of the art in research on mashups development and are confident the reader will enjoy discovering how each tool was able to compete in the challenge, as described in the next chapters.

References

1. Daniel, F., Matera, M.: Mashups: Concepts, Models and Architectures. Springer (2014)

Table 1: Overview of the mashup and mashup tool features declared by the approaches that participated in the ICWE 2015 Rapid Mashup Challenge.

			FlexMash	UI-Oriented Computing	Smart-Composition	EFESTO	WebMakeup	WLS
Mashup	**Mashup type**	Data mashups	✓					
		Logic mashups						✓
		UI mashups		✓			✓	
		Hybrid mashups		✓	✓	✓		
	Component types	Data components	✓		✓	✓		
		Logic components			✓	✓		✓
		UI components		✓	✓	✓	✓	
	Runtime location	Client-side only		✓		✓	✓	
		Server-side only						
		Client-server	✓		✓	✓		✓
	Integration logic	UI-based integr.		✓		✓	✓	
		Orchestration	✓			✓		
		Choreography			✓			✓
	Instantiation lifecycle	Stateless	✓					
		Short-living		✓	✓	✓	✓	✓
		Long-living				✓		
Mashup tool	**Target end-user**	Local developers			✓			
		Non-programmers	✓	✓		✓	✓	
		Expert programmers		✓				✓
	Automation degree	Full automation						
		Semi-automation	✓	✓		✓	✓	✓
		Manual		✓	✓			
	Liveness level	Level 1 (mockup)						
		Level 2 (manual)						
		Level 3 (automatic)	✓				✓	
		Level 4 (dynamic)		✓	✓	✓		✓
	Interaction technique	Editable examples			✓			
		Form-based						
		Progr. by demonstration		✓				
		Spreadsheets						
		Textual DSL		✓				✓
		Visual (iconic)	✓					
		Visual (wiring, implicit)				✓		
		Visual (wiring, explicit)						
		WYSIWYG		✓			✓	
		Natural language						
	Online user community	None	✓	✓	✓	✓		✓
		Private					✓	
		Public						

2. Berners-Lee, T., Hendler, J., Lassila, O.: The semantic web. Scientific American (May 2001) 34–43
3. Aghaee, S., Nowak, M., Pautasso, C.: Reusable decision space for mashup tool design. In Barbosa, S.D.J., Campos, J.C., Kazman, R., Palanque, P.A., Harrison, M.D., Reeves, S., eds.: EICS, ACM (2012) 211–220
4. Tanimoto, S.L.: Viva: A visual language for image processing. Journal of Visual Languages & Computing **1**(2) (1990) 127–139
5. Myers, B.A., Ko, A.J., Burnett, M.M.: Invited research overview: end-user programming. In: CHI'06 extended abstracts on Human factors in computing systems, ACM (2006) 75–80
6. Nardi, B.A.: A small matter of programming: perspectives on end user computing. MIT press (1993)