Mashup Development with Web Liquid Streams

Andrea Gallidabino, Masiar Babazadeh, and Cesare Pautasso

Faculty of Informatics, University of Lugano (USI), Switzerland
{name.surname}@usi.ch

Abstract. Web services such as Twitter and Facebook provide direct access to their streaming APIs. The data generated by all of their users is forwarded in quasi-real-time to any external client requesting it: this continuous feed opens up new ways to create mashups that differ from existing data aggregation approaches, which focus on presenting with multiple widgets an integrated view of the data that is pulled from multiple sources. Streaming data flows directly into the mashup without the need to fetch it in advance, making it possible to exchange data between mashup components through streaming channels. In this challenge submission we show how streaming APIs can be integrated using a stream processing framework. Mashup components can be seen as stream operators, while the mashup can be defined by building a streaming topology. The mashup is built with Web Liquid Streams, a dynamic streaming framework that takes advantage of standard Web protocols to deploy stream topologies both on Web servers and Web browsers.

Keywords: Mashups, Streaming

1 Introduction

The mashup concept and the interest in the mashup tools started to appear when more and more Web services and Web Data sources were released [1]. While Mashups can be built using traditional Web development tools, languages and frameworks, specialized mashup composition tools have appeared focusing on raising the level of abstraction and thus enabling non-programmers to compose mashups [2]. Different tools can be characterized depending on the users they target, and the mashup development approach they implement [3]. A precise categorisation of the various mashup tools describes synthetically their expressive power and the type of solution they propose can be found in [4].

Mashups are in general data centric applications which gather data from many Web services or Web data sources and mix them together in a single integrated application. Data may be fetched from the Web in many different forms: static resources accessible through static URLs (e.g. JSON/XML files), resources accessible through REST APIs, or – in the case of this paper – streaming and feed APIs that forward new data to clients without the need of any new request after the initial subscription. Mashup tools make it easy to integrate one or more of those type of data.

This paper presents the rapid mashup challenge solution proposed by the Web Liquid Streams (WLS) framework [5], a stream processing runtime that helps developers deploy streaming topologies running on heterogeneous Web-enabled devices. WLS helps the users to develop logic mashups by creating JavaScript logic components. Components may interact with any of the data sources described above and components may be connected together in order to create a streaming topology representing the mashup.

2 Web Liquid Streams Framework

WLS helps Web developers to create streaming topologies running across heterogeneous Web-enabled devices. Any device on which a Web browser or a node.js Web server can run can be used to produce, process or consume a WLS stream. WLS targets programmers that are able to write JavaScript code that runs both on the server and on the client. Mashup components in WLS are called *operators* and may interact with any Web services API (both streaming, RESTful and RPC-based). An operator is the core building block of a streaming topology, it can receive data, process it and forward results downstream. By *binding* (connecting) two or more operators together it is possible to define a streaming topology.

Operators may run on Web server or on Web browsers. In both environment they can use the same WLS API to produce and consume the data stream. Operators running on a Web Browser have also access to an extended API for rendering the data stream and visualize it on web pages.

Web Liquid Streams abstracts away the deployment of the topology on the heterogeneous machines from the developers, it keeps the mashup alive in case of failures and if a mashup component overload is detected it will automatically allocate more resources [6].

WLS offers the following features, which will be demonstrated during the challenge:

Reusable mashup components Mashup components written as JavaScript operators can be reused in more than one topology. Component development is completely open for developers that can reuse JavaScript libraries and remotely access any Web service API.

Live mashup development Mashups can be changed while they run. WLS gives to the users direct access to the topology-creation tools through a command line interface. Users can *run*, *stop*, or *bind* mashup components, furthermore they can decide to *migrate* them on any peer connected to the application.

JSON mashup definition language Mashups can be defined by using our internal DSL based on the JSON syntax. The structure of mashups created interactively through the dynamic topology-creation tools can be viewed and automatically saved in JSON format, which can be manually edited and once again deployed to reconstruct the same mashup

Web of Things mashups Web Liquid Streams enables integration with smart devices and sensors which can create streams of data. WLS can run mashup components directly on those devices so that they can directly access hardware sensors and actuators.

Distributed user interface mashups Multiple operators to visualize the data stream can be instantiated and deployed on different client devices so that the same mashup results can be shared among multiple users.

3 Checklist

3.1 Mashup Feature Checklist

Mashup Type	Logic mashups
Component Types	Logic components
Runtime Location	Both Client and Server
Integration Logic	Choreographed integration
Data passing logic	Direct data passing
Instantiation Lifecycle	Short-living

3.2 Mashup Tool Feature Checklist

Targeted End-User	Programmers
Automation Degree	Semi-automation or manual
Liveness Level	Level 4
Interaction Technique	Textual DSL and other (console)
Online User Community	None

4 Demo

In the challenge demo we will present a nonlinear topology deployed both on server and clients. We will use three different APIs: Google Maps[1], GeoNames[2], and the Twitter REST[3] and streaming[4] APIs. The presentation will show how operators and bindings behave in a topology and how it can change dynamically at runtime by adding, removing, or migrating operators. Figure 1 shows the structure of the topology that will be implemented during the mashup challenge. Operators will be predefined in advance. At the end of the demo we will involve the audience so that they can deploy the mashup UI components on their Web browsers to see for themselves the results of the stream processing.

[1] https://developers.google.com/maps/
[2] http://www.geonames.org/
[3] https://dev.twitter.com/rest/public
[4] https://dev.twitter.com/streaming/overview

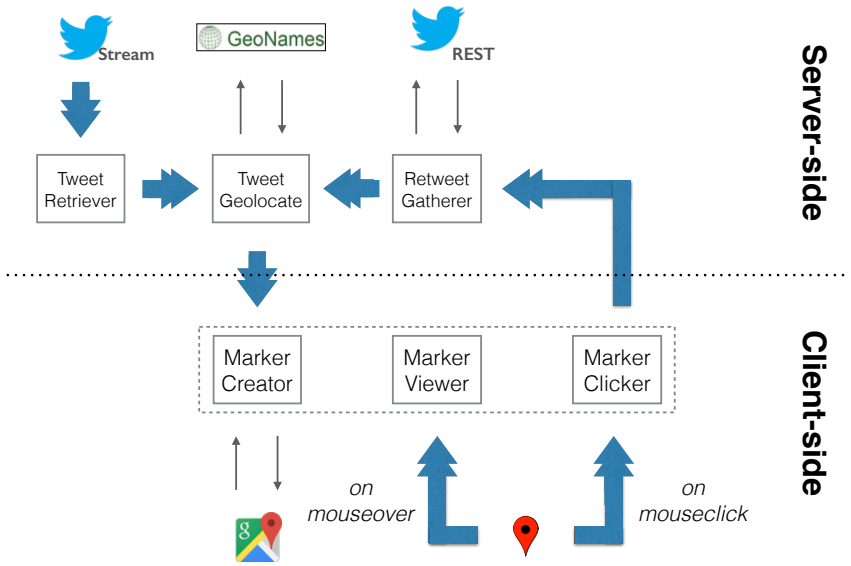

Fig. 1: Mashup Challenge Topology

Acknowledgment The work is supported by the Hasler Foundation with the Liquid Software Architecture (LiSA) project.

References

1. Zang, N., Rosson, M.B., Nasser, V.: Mashups: who? what? why? In: CHI'08 extended abstracts on Human factors in computing systems, ACM (2008) 3171–3176
2. Liu, Y., Liang, X., Xu, L., Staples, M., Zhu, L.: Composing enterprise mashup components and services using architecture integration patterns. Journal of Systems and Software **84**(9) (2011) 1436–1446
3. Aghaee, S., Nowak, M., Pautasso, C.: Reusable decision space for mashup tool design. In: Proceedings of the 4th ACM SIGCHI symposium on Engineering interactive computing systems, ACM (2012) 211–220
4. Daniel, F., Matera, M.: Mashups: Concepts, Models and Architectures. Springer (2014)
5. Babazadeh, M., Gallidabino, A., Pautasso, C.: Decentralized stream processing over web-enabled devices. In: Proc. of 4th European Conference on Service-Oriented and Cloud Computing (ESOCC 2015), Taormina, Italy, Springer (September 2015)
6. Babazadeh, M., Gallidabino, A., Pautasso, C.: Liquid stream processing across web browsers and web servers. In: Proc. of ICWE. Springer (2015) 24–33