ABSTRACT

HTML5 is a new technology standard promising to empower browsers to become a suitable platform for developing rich Web applications. Whilst it is still considered an emerging technology, in this paper we attempt to capture and explore its impacts on mashup development. To do so, we start with a mashup case study showcasing new HTML5 features. We then move on to explore those aspects of mashup development that are affected and will possibly be enabled by HTML5 in the near future. These aspects are grouped into two categories: short-term impacts, that can be harnessed by mashup developers, and long-term impacts, that should be considered by service/content providers.

Categories and Subject Descriptors
D.2 [SOFTWARE ENGINEERING]: Design—Methodologies; H.3 [INFORMATION STORAGE AND RETRIEVAL]: Online Information Services—Web-based Services

General Terms
Standardization, Design

Keywords
Mashup Development, HTML5, Mashup

1. INTRODUCTION

With the trend towards the Web as a platform, browsers have turned into more than stand-alone applications for accessing the Web. As they are more and more used to run Rich Internet Applications (RIAs) like mashups, they have to ensure the efficient and reliable execution of client-side scripts. Moreover, with the rise of mashups as situational applications - applications that often (but not always) have a short life-span, and are created for a specific group of users with a unique set of needs- [16], browsers form the only bridge between a situational mashup and its users, through which the mashup can automatically access its user’s contextual information. Finally, the client-side of a RIA running on a browser usually communicates with a server, which, in case of mashups, can be either the mashup server or the content/service providers. Browsers as the point of departure for this communication and service/data delivery should initiate a faster and more secure connection.

HTML5 is an emerging technology standard1, geared towards addressing these challenges [35, 5, 10, 15]. For this reason, whereas HTML5 has not yet reached a formal completion of its standardization process, most recent browsers already provide support for many of its most innovative features. The purpose of this paper is, therefore, to discuss how mashup development is likely to be affected by HTML5 in the context of a concrete example case study. We present positive aspects and improved solutions as well as identify which challenges of mashup development still remain open. The rest of this paper is structured as follows. In the next section we put the current work into context by providing an overview of related work. In section 3, we describe a scenario, in which a mobile mashup is built using the new HTML5 features. Section 4 and 5 elaborate the contribution of this paper by highlighting, respectively, short-term and long-term impacts of HTML5 features on mashup development. We draw some conclusions and discuss remaining challenges to incorporate HTML5 for mashups development in section 6.

2. RELATED WORK

The majority of the research work towards fostering best practices for mashup development contributes to a better understanding of what a mashup is, and how it should be developed. This spans data integration [33, 19], process integration [32, 34], UI (User Interface) integration [4, 3, 37], context-awareness [2, 6], mashups in enterprise environments [27, 22, 36, 13], and end-user programming [14, 26]. These efforts provide a foundation for creating tools, technologies, and standards within the domain of mashup. Mashup tools are targeted towards facilitating, formalizing, and partially automating the mashup development process. Examples are Yahoo Pipes2, IBM Mashup Center3, and Intel Mash Maker [7]. A good example for the purpose of standardization is the Enterprise Mashup Markup Language

1 http://dev.w3.org/html5/spec/
2 http://pipes.yahoo.com/pipes/
3 http://www-01.ibm.com/software/info/mashup-center/
3. CASE EXAMPLE

In this section, we will outline the main features of HTML5 by describing a scenario in which a mobile mashup is developed. The attempt was made to keep the example as comprehensive and practical as possible so that insight can be obtained into how HTML5 will affect mashup development. Throughout this paper we shall refer to, or extend this case example for the purpose of exemplification.

3.1 Tourist Assistant Mashup

One of the best ways for tourists to entertain themselves is to attend the most popular events in the city they are currently visiting. This, however, requires prior knowledge about what kind of events will take place in the city. Having such information in hand is, therefore, beneficial for the tourists because they can make the most of their available time. In this scenario, a data mashup comes so handy as it can aggregate and effectively present required data from different sources on the web.

In order to detail the requirements of the mashup, consider a scenario where a tourist (we call him John) wants to acquire information about the upcoming events in Lugano. John is a software engineer, and can manage to build a mashup to address his needs. While on vacation, he prefers to use his handheld device that can connect to the Internet, and therefore, the final mashup should be able to run on that device. The primary types of information to be provided are availability, location, time, and type of the upcoming events.

As for the required functionalities, since he is a newbie in the city, an integrated GPS navigator that can lead him step by step to the location of events seems essential. Figure 1 illustrates the architectural view of the mashup.

Using the new HTML5 GeoLocation API, the mashup can extract the current location of the user in realtime. This is mashed together with the Google Maps APIs\(^4\) to build up a simple navigator. The mashup accesses Eventful APIs\(^6\) in order to retrieve a list of future events filtered by the city name (Lugano) obtained from HTML5 GeoLocation APIs. The events, afterwards, will be geographically projected on a Google Map widget as a set of markers. Markers on the map are then filtered by selecting an event type (e.g. movie, music, etc.). Selecting each event marker from the table, first loads a list of relevant photos, obtained from Flickr APIs\(^7\), on the balloon tied to the selected event marker, and second shows directions from the current location of John to the location of the event.

3.1.1 Enhanced Tourist Assistant Mashup

After a nice and fruitful vacation using this mashup, John is now willing to share it with other people. Assuming that so many tourists might be interested in using this mashup, he decides to upgrade its functionality by adding an online chat feature, so that people in the same city can discuss the upcoming events. He has already developed an online chat mashup using the HTML5 WebSocket API, using which users, in the same city, can chat with each other in their own native language. The latter feature is delivered by the Google Language API\(^8\), that detects the language of the received message and translates it to the language selected by the user.

To be able to chat with other users, a user should first register into the WebSocket server using a unique ID. The ID is chosen by the user and will be checked by the server for uniqueness against all the existing IDs. If the user gets disconnected, his/her associated ID will be removed from the server. Once the registration is accepted by the server, the mashup should send the current location of its user to the

---

\(^4\)http://www.openmashup.org/
\(^5\)http://code.google.com/apis/maps/documentation/javascript/
\(^6\)http://api.evdb.com/
\(^7\)http://www.flickr.com/services/api/
\(^8\)http://code.google.com/apis/ajaxlanguage/documentation/
server. This is because the server determines the chat rooms based on the similarity of user’s locations. When embedding the widget within the mashup makes use of the HTML5 cross-window postMessage API. This feature enables the chatroom widget to send a notification to the parent mashup upon a new message is arrived. The notification contains the geographical coordinations of the message as well as its content. The geographical coordinations of the sender location is obtained from GeoLocation APIs, and will be attached to all outgoing messages. Thereby, each mashup may be shown as a marker on the parent mashup map widgets, with an info-window that displays the message itself.

4. SHORT-TERM IMPACTS

In this section, we will show how HTML5 features can be immediately employed by mashup developers, in a short-term manner, to improve their productivity in existing scenarios of mashup development. These scenarios include development of mobile mashups, collaborative mashups, client-side mashups, and server-side mashups.

4.1 Mobile Mashups

Within the last few years, new mobile devices such as smartphones, tablets, and Personal Digital Assistants (PDAs) have been growing ever more capable of handling larger and more complex tasks. In a number of countries, advances in communication technologies currently allow these devices to connect to the Internet in the same way as Personal Computers (PCs). Such dramatic improvements are expected to increase the use of internet-enabled mobile devices as powerful tools for surfing the Web to the extent of over 900% by 2014 [9]. From a usability perspective, however, the dynamic nature of the Web environment along with the dynamically changing geographical distribution of mobile users, result in the long tail [1] of dramatically changing user needs. As a consequence, mobile devices have become a demanding market for situational applications. One kind of these situational applications is called hybrid apps, that run natively on mobile devices but take advantage of Web APIs. Yet, hybrid apps cannot fully leverage this market for two important reasons. First, these apps are platform-dependent, meaning that each platform offers its own Software Development Kit (SDK). Accordingly, users of a particular platform, like Android, are limited to use Android-specific apps, and therefore, will be deducted from the benefits they can derive from interesting apps built for a different platform such as Windows. Second, building hybrid apps requires programming skills, even though it builds upon reusable Web APIs. As a result, the costs need to be balanced against the ultimate benefits one is supposed to gain from the intended app. This is, therefore, an important hurdle that can discourage creation of many costly hybrid apps, having short life-spans, but are useful for satisfying instant needs.

On the other hand, mobile mashups—mashups that run on mobile browsers—are viable alternative products for this market [23]. Mobile mashups can be accessed from any browsers on any platform. This especially enables portability across different mobile platforms. Similar to a hybrid app, a mobile mashup combines reusable components from a large pool of Web APIs. However, it can be possibly developed using emerging mashup end-user programming tools such as Yahoo Pipes⁹, which requires much less effort as opposed to hybrid apps. Nevertheless, mobile computing possesses fundamental constraints that require adaptation by mobile situational applications [31, 12]. These constraints, however, have not been visible for mobile browsers, and consequently for the mashups running on these browsers. This was a considerable disadvantage of mobile mashups as opposed to native hybrid apps. HTML5, which is currently being supported by most of mobile browsers, is believed to make some of these constraints accessible by browsers. To be specific, there are three important mobile computing constraints that have been made accessible by HTML5: variable location, intermittent connectivity, and small size screen. In the following three subsections, we discuss why and how mobile mashups should be adapted to these constraints.

4.1.1 Location-based Mashups

It can be empirically ascertained that users expectations from many mobile mashups are greatly affected by geographical location. For instance, in the example previously described, users could expect the tourist assistant mashup to realize, and react according to their location, i.e. showing relevant events, and giving accurate directions. Thus, as the demand of mobile users to use and integrate geospatial data increases, the role of location awareness computing is becoming more important for the future of mobile mashups. HTML5 exposes access to all sources of information regarding the location of the device, using its Geolocation API. This new feature can fill the existing gap caused by lack of location-aware application support in order to develop location-based mashups. The comprehensiveness and effectiveness of Geolocation APIs is supported by the fact that a variety of available sources are automatically compared to check the accuracy of the location [21]. These sources are IP address, GPS, WI-FI with MAC address, GSM or GDMA, and cell phone IDs. Additionally, mobile mashups are provided with such information in realtime, thus enabling an immediate response. For instance, in the scenario example, a realtime location change notification feature could facilitate development of the GPS navigator.

4.1.2 Offline Mashups

The advent of mobility has affected Internet usage scenarios. In this context, challenges arise when users get dropped from the Internet due to their mobility. For instance, Internet connectivity may suffer while in places such as sub-
way, airplane, and train. When the internet connection becomes temporarily unavailable, most of current mobile Web applications will not be responsive even for tasks that do not require connection to server. This is, therefore, a major challenge promoting the continued use of offline hybrid apps. However, HTML5 offline caching API is out to address this challenge, by allowing users to keep interacting with the (cached) web application while in an offline mode [5].

In the context of mashup development, this feature presents many opportunities. It allows mobile mashups to cache the latest update received from the providers or mashup server. As a result, mashups can work in offline mode by operating on the latest data. In fact, this feature suits development of many mashups, since the common practice in mashup development is to fetch data once from the providers, integrate it, and then present the integrated data to the user. The rest of the user interaction mostly involves browsing the presented data which can be prefetched and cached locally, and only occasionally requires to further connect to the server or providers. For instance, in the case example, all the events are fetched and presented to users once the mashup runs or the current city of the user is changed. The rest of the user interaction concerns with filtering and browsing the events displayed on the map widget.

To enable offline mobile mashups, the best practice is to frequently cache the data fetched from the providers, the latest integrated data, and the Javascript operating, and presenting these data. It should be added that the data should be stored in the local storage with the use of WebStorage API. WebStorage API is a new feature that has also appeared in HTML5. Currently, Google chrome offers 10MB of local space capacity for Web storage.

### 4.1.3 Screen Portability

The Web browser as a programmable platform empowers developers to build mashup applications that are portable on any Operating System (OS) platform. However, the unique constraints of mobile computing [31] might cause a portability problem between desktop mashups and mobile mashups, or in a broader sense between mobile Web apps and desktop Web applications. One of these constraints is correlated with the small screen and low resolution of mobile devices compared to desktop computers. When it comes to mobile browsers, this constraint gets highlighted, as the majority of mashups are not visually optimized for mobile browsing. Therefore, mashups currently either run well on a mobile browser or a desktop browser.

CSS3 Media Queries makes it easier for mashup developers to enable screen portability. They can easily create an alternative CSS file for existing desktop mashups to also suit mobile devices. In such a way, many desktop mashups can be easily ported to mobile browsers and vice versa. Also, new mashups can be built targeting both mobile and desktop browsers as they use the CSS3 features to adapt themselves. In reality, using screen portability for creating new mashups is of importance as many mashups will be likely to run on both mobile and desktop platforms. Considering the case example, even though the target platform is a handheld device, the mashup is also likely to run on desktop computers (e.g. in a hotel).

### 4.2 WebSocket-based Collaborative Mashups

An application that is deployed in a networked environment like the Web, can potentially mediate the interaction among it users [11]. The resulting application is called a collaborative application [17] as its users collaboratively create and manage content. Examples are many modern Web 2.0 applications ranging from collaborative editors to online chat rooms.

Mashups can also expose a collaborative front-end. The heterogeneous back-end of mashup when accessed through a collaborative front-end creates interesting use cases. For instance, in the case example, the chatroom mashup provides a collaborative front-end in which users, thanks to heterogeneous functionality aggregated within the mashup, can communicate with each other in any languages. Furthermore, there are a number of domains that are growing interests towards collaborative mashups. They include (but not limited to) Collaborative Decision Making (CDM) [29], emergency response [18], and e-learning [30].

From a technical point of view, the underlying model of a collaborative mashup is based on realtime communication among the involved participants, which is now significantly facilitated by the HTML5 WebSocket API. This new feature improves upon its predecessors in terms of development simplicity [35], and communication speed and efficiency [20]. The simplicity and power of the WebSocket APIs can induce the proliferation of collaborative mashups in the consumer market. Likewise, collaborative mashups used within industrial domains, can be largely enhanced by taking advantage of the high communication speed and efficiency that the WebSocket API provides.

### 4.3 Client-side Mashups

Mashups are designed with two different architectures, depending on where process and data integration takes place. If it takes place on the server-side, the mashup is called a server-side mashup. In this case, once the final result is produced on the server, it will be pushed to the client for the sake of visualization. Alternatively, both integration and visualization tasks can be performed in client browser, which results in a client-side mashup.

Despite the fact that client-side architecture has its disadvantages (less security, reliability, and performance), it can still provide faster user experience, less load on the server side, and easy development [25]. To do so, one important feature that until recently has been missed for browser-based clients is, indeed, multithreading. Multithreading is a technique that has been used by desktop and server-based applications to increase performance while performing concurrent long-running tasks.

This feature allows developers to take advantage of multithreaded JavaScript support in browsers. HTML5 introduces this feature as the WebWorker API. In the continuing discussion, we will more deeply discuss how this new feature can be utilized in efficient client-side mashup development. The focus is specifically on client-side execution of Process Integration (PI), Data Integration (DI), and data representation.

#### 4.3.1 Process Integration

In mashup development, PI takes place in the application level where the logic of the mashup resides. PI concerns putting together external Web APIs and services in order to perform a function. With the use of the WebWorker API,
PI can be executed in the background without interfering with the UI. However, PI is more than a single background process. In fact, an important component of PI is a composition model telling how the external services are integrated to form a mashup logic. The true value of the WebWorker API is thus revealed as it can enable lightweight, client-side execution of process integration models. We identify and explain the following characteristics of a composition model for PI that benefit from the WebWorker API.

Orchestration. The WebWorker API enables two different styles of orchestration. The first one is a sequential style, in which PI worker runs from beginning to end in a sequential manner. It means that the UI merely invokes the PI worker, and may also wait for its final response. The second one uses an event-based style in which the PI worker is driven by UI intervention. These interventions are in form of UI events triggered by the user.

Initiation: Another characteristic concerns the way PI worker is initiated. It can be invoked directly from the UI. Alternatively, the initiation of a PI worker can be scheduled after a given delay or repeated with a certain frequency.

Termination: The WebWorker API allows setting a time out for a PI worker. In this sense, if the PI worker does not finish its job in the set period of time, it will be automatically terminated. This is important because it allows to deal with unresponsive remote data sources and services without blocking the mashup execution.

Subprocess. A composition model may also involve execution of sub processes. In this case, The WebWorker API allows a subworker to perform a task on behalf of another worker.

Dataflow. Data and message passing between the UI and PI workers as well as among PI workers can be performed by one of the following mechanisms:

- **Flow-based:** In this style the communication is performed directly between UI, PI worker, and its subworker through the use of postMessage API. This is, in fact, the only official communication technique offered by WebWorker API. A subworker can only use postMessage API to communicate with its parent worker. This restriction has been imposed according to concurrent design principles. As for the data, the postMessage API can be used to send most JavaScript variable types. Moreover, various type of formats such as JSON and XML can be wrapped into string objects.

- **Shared memory:** Shared memory is another communication style, in which data to be communicated is shared on a storage to where all the participates have access. This style is not supported by WebWorker APIs. Instead, we enable this through another new HTML5 feature, which is called Web storage API. It offers realtime notification of data manipulation to all the browsing contexts that could access it (e.g. UI thread, workers, and subworkers). Thereby, in contrast to postMessage, Webstorage can mediate the communication between a subworker and the UI.

4.3.2 Data Integration

Another usage of the WebWorker API in mashup development is for performing more efficient client-side DI. Data can be retrieved through Web APIs, RSS feeds, or Web scraping. DI deals with operating on these data such as conversion, transformation, filtering, and combining [24]. Depending on the volume of data, these tasks might consume considerable time and resources. Therefore, the best place to efficiently perform DI on the client is through a spawned worker.

4.3.3 Data Representation and Visualization

After the data is integrated, the final step is to present the resulting data to the user. The data is usually in a standard format such as XML, JSON, or a custom defined format. In neither case the data is readable by a human user. Therefore, the data should be transformed to HTML so that it can be rendered by the browsers to construct an informative visual representation. This task is usually performed by a front-end JavaScript. If the volume of the data is large, it becomes a long-running task that can possibly freeze the UI. Therefore, using a worker for this purpose can potentially improve the user experience of the mashup. The outcome of the worker then can be easily placed inside a DIV element.

4.4 Server-side Mashups

Data transmission between the mashup client and server is normally done through client-side XMLHttpRequest (XHR) calls. However, when dealing with realtime data, a better communication way between the mashup client and server could be over HTML5 WebSockets. The Ericsson Lab report indicates that WebSockets produce significantly less overhead than current XHR to the extent of 35% reduction in the data (in bytes) communicated between the client and server. Replacing XHR with WebSockets

https://labs.ericsson.com/developer-community/blog/what-websocket-difference
for pulling realtime data from server, therefore, results in mashups that run faster and consume less bandwidth.

5. LONG-TERM IMPACTS

Mashup developer and service/content provider are two important stakeholders in mashup development. In conjunction with mashup developers, we discussed different aspects of mashup development that are affected by HTML5 features. However, there still remains a number of enhancements that are to affect service and content providers in the future. Therefore, these aspects are counted as long-term impacts of HTML5 on mashup development.

5.1 Point-to-Point UI Integration

The UI is an important component in most mashups, by which users and the mashup interact with each other. Mashup UI constitutes the front-end, by means of which users access and exchange information with the back-end logic and data. The back-end concerns with integration of data and functionality rendered from third-part contents and services. One interesting fact about mashup is that its front-end can also be built by combining visual representation of contents. Contents that provide a visual representation are called UI components, chiefly exemplified by widgets. The act of incorporating UI components into a new front-end for mashup is thus called UI integration.

One of the dimensions in the UI integration realm is correlated with how UI components communicate with each other [4]. From an architectural perspective, the style of communication can be either centralized or point to point (Figure 5). In a centralized communication style, the interaction of components is mediated by the parent mashup. In doing so, the mashup offers an eventbus, that uses a publish/subscribe mechanism to handle events fired by the components. When a point to point communication style is applied, the interaction occurs directly between the components. This style is especially useful when the interaction between the components is not complex.

However, the direct communication between UI components has not been technically feasible due to lack of support. To be specific, current UI components are not built for point-to-point communication. Even so, the direct communication could be banned by Same Origin Policy (SOP), given the fact that various UI components are generally from different domains. Presently, the new HTML5 postMessage API has changed.

This defines a binding mechanism between UI components. One of the dimensions in the UI integration realm is correlated with how UI components communicate with each other [4]. From an architectural perspective, the style of communication can be either centralized or point to point (Figure 5). In a centralized communication style, the interaction of components is mediated by the parent mashup. In doing so, the mashup offers an eventbus, that uses a publish/subscribe mechanism to handle events fired by the components. When a point to point communication style is applied, the interaction occurs directly between the components. This style is especially useful when the interaction between the components is not complex.

However, the direct communication between UI components has not been technically feasible due to lack of support. To be specific, current UI components are not built for point-to-point communication. Even so, the direct communication could be banned by Same Origin Policy (SOP), given the fact that various UI components are generally from different domains. Presently, the new HTML5 postMessage API has changed.

In the above JSON configuration, the event foo is associated with method bar. The index parameter points to the target iframe, which holds the corresponding UI component, and is indexed by the parent mashup. The parameters of the event are also mapped to the input parameters of the methods. This defines a binding mechanism between UI components.

5.2 WebSocket-based Content Providers

In many cases a mashup that is built by composing RESTful services needs to update itself as soon as the resource state is changed by other clients. For instance, a mashup that draws upon twitter REST APIs\(^\text{11}\) may need to get notified as soon as a new tweet is posted. To this end, mashup need to constantly send GET requests and check if the resource state has been changed. It results in heavy traffic, hitting the provider server, as well as reduction of mashup performance due to wasted JavaScript execution for sending unnecessary GET requests. This is especially an issue when RESTful service providers limit the number of requests per client.

Here we see another potential use for the WebSocket API as a mechanism for enabling realtime communication with content providers. The WebSocket API provides a generic communication mechanism that makes it possible to transform various representations of resources on the Web including XML, JSON, and Atom. Therefore, WebSocket API can be used to sent data to clients only when the resource state has changed.

5.3 Pure Client-side Mashup Architecture

Current client-side mashup architecture usually includes a mashup server through which all client requests to fetch contents or services are routed. In fact, incorporation of a server is required by the fact that the client can not directly make XHR calls to content/service providers that reside in a different domain. This is due SOP constraint imposed by browsers.

The use of a server in client-side architecture can possibly increase security vulnerability, and decrease performance.

\(^\text{11}\)http://apiwiki.twitter.com/
The essence of the data received from the server cannot be determined until it is consumed by the client. This is a potential vulnerability as the data may contain harmful scripts that can give a malicious server full control of the client system. Moreover, the use of a server as a proxy to transform request/response on behalf of the client and content/service providers may reduce speed and performance. This is because all the requests and responses should first reach a server and then be forwarded by the server to either the client or provider. Routing requests and responses still requires programming which uses server-side languages and scripts such as Java and PHP. Therefore, in the best case, a mashup can be developed by three different languages (one for server-side, and Javascript and HTML for client-side programming).

Put differently, excluding the server from the client-side mashup architecture, in a way that client and providers directly communicate with each other, results in more security, better performance, and less development effort. The cross-site XHR in HTML5 can potentially enable a pure client-side architecture. The prerequisite is that provider must be able to handle cross-site HTTP requests, which, in turn, requires Cross-Origin Resource Sharing (CORS) enabled in the provider server. This not only enables the potential use of a pure client-side architecture, but also allows providers to build an access control mechanism to check the origin of the incoming requests. It can possibly replace the use of long developer keys to check the validity of the requests.

6. CONCLUSION
As summarized in Table 1, we outlined both the short-term and long-term impacts of HTML5 features on the way mashups are developed. The short-term impacts can be harnessed immediately by mashup developers to enhance the development of mashups. Contrariwise, the long-term impacts yet remain to be realized by service/content providers.

<table>
<thead>
<tr>
<th>HTML5 feature</th>
<th>Short-term impact</th>
<th>Long-term Impact</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Geolocation API</td>
<td>Location-based (mobile) Mashups</td>
<td></td>
<td>Track the current geographic location of the client and use it as input to location-based mashups</td>
</tr>
<tr>
<td>WebSocket API</td>
<td>Collaborative Mashups &amp; server-side mashups</td>
<td>WebSocket-based content providers</td>
<td>The WebSocket API provides simple development of faster and more efficient collaborative and server-side mashups. It can also be used as a real-time subscription method for the data requiring periodic update.</td>
</tr>
<tr>
<td>postMessage API</td>
<td>Multithreaded client-side Mashups</td>
<td>Point-to-point UI integration</td>
<td>postMessage API can be used to enable a direct communication between UI components</td>
</tr>
<tr>
<td>WebWorker API</td>
<td>Multithreaded Javascript support</td>
<td>Multithreaded Javascript support can speed up client-side data integration and process integration operations</td>
<td></td>
</tr>
<tr>
<td>Cross-site XHR</td>
<td>Pure client-side mashup</td>
<td></td>
<td>The use of cross-origin resource sharing by providers can result in creation of pure client-side mashup (client-side mashups without a server)</td>
</tr>
<tr>
<td>Offline Caching API</td>
<td>Offline mashups</td>
<td></td>
<td>Empower users to work with the mashup when the connection to its data source is dropped</td>
</tr>
<tr>
<td>CSS3 Media Queries</td>
<td>Screen portability</td>
<td></td>
<td>The mashup UI can adapt itself according to whether it runs on desktop or mobile browsers</td>
</tr>
</tbody>
</table>

Table 1: HTML5 impacts on mashup development

As a whole, HTML5 will provide a strong foundation for development of various types of mashups including client-side mashups, server-side mashups, mobile mashups, and collaborative mashups. However, mashup development still requires HTML5 to continue following the path towards letting browsers know more about the contextual information of the users, and further providing mashups with such information in a streamlined manner. We already discussed three forms of contextual information (location, connection status, and screen-size) that are currently accessible using HTML5. However, the context of the user can be characterized by more parameters such as the client computational power that are not yet formally accessible by HTML5. Therefore, the more browser can make such information visible to a mashup, the better the mashup can respond and adopt. Moreover, the main advantage of native hybrid apps over mobile mashups is the privilege of having access to mobile features like Bluetooth, and camera. Therefore, in order to empower mobile mashup to compete with native hybrid apps, HTML5 should allow browsers to have access to such mobile features.

7. REFERENCES