


# Towards Liquid Web Applications

Tommi Mikkonen<sup>1</sup>, Kari Systä<sup>1</sup>, and Cesare Pautasso<sup>2</sup>

<sup>1</sup> Department of Pervasive Computing, Tampere University of Technology,  
Tampere, Finland

{tommi.mikkonen,kari.systa}@tut.fi

<sup>2</sup> Faculty of Informatics, University of Lugano (USI), Lugano, Switzerland  
c.pautasso@ieee.org

**Abstract.** As the complexity of rich Web applications grows together with the power and number of Web browsers, the next Web engineering challenge to be addressed is to design and deploy Web applications to make coherent use of all devices. As users nowadays operate multiple personal computers, smart phones, tablets, and computing devices embedded into home appliances or cars, the architecture of current Web applications needs to be redesigned to enable what we call Liquid Software. Liquid Web applications not only can take full advantage of the computing, storage and communication resources available on all devices owned by the end user, but also can seamlessly and dynamically migrate from one device to another continuously following the user attention and usage context. In this paper we address the Liquid Software concept in the context of Web applications and survey to which extent and how current Web technologies can support its novel requirements.

## 1 Introduction

Today, the average consumer in the U.S. or Europe has two primary computing devices – a personal computer, usually a laptop, and a smartphone, with more new mobile devices and tablets activated every day. We are on the move from a world in which each person has two or three devices to a world in which people will use dozens of computing devices – laptops, phones, tablets, game consoles, TVs, car displays, digital photo frames, home appliances, wearable computers, and so on. All these devices are connected to the Web, and their users are provided with computation that is constantly available, capable of delivering meaningful value even in few moments, without requiring active attention from the users part [1].

The architecture of current Web applications is not living up to these expectations. Content is increasingly made available on the Web and users have many ways to access the content published on the Web, but they are exposed to extra complexity caused by the large number of Web-enabled devices at their disposal. Additional complexity comes from the fact that user content is spread to several devices and Internet services. Managing all these as separate entities is currently a tedious task, while at the same time users expect casual experiences. Since all these devices are already connected to the Web, orchestrating their actions to simplify users' lives would be a natural extension of the Web platform.

To create software that fits with the multi-device paradigm, the architecture of current Web applications needs to be redesigned to enable what we call *Liquid Software* [2]. Such Liquid Web applications not only can take full advantage of the computing, storage and communication resources available on all devices owned by the end user, but also can seamlessly and dynamically migrate from one device to another continuously following the user attention and usage context.

In this position paper we apply the Liquid Software concept to the design of Web applications and survey to which extent and how current Web technologies can support its novel requirements. The rest of this paper is structured as follows. In Section 2, we provide background for the concept of Liquid Software. In Section 3, we discuss engineering Liquid Applications in the light of present Web technologies. In Section 4, we provide an extended discussions regarding our findings pointing out future research directions, summarized in Section 5.

## 2 Liquid Software: Background and Related Work

The notion of Liquid Software was originally proposed in [3]. In our interpretation, the essential feature of Liquid Software is support for hassle-free multi-device experiences, falling to the following categories [4]:

1. **Sequential Screening.** A single user runs an application on different devices at different times. The application adapts to the different devices capabilities while respecting the actual user needs in different usage contexts.

2. **Simultaneous Screening.** A single user uses the services from several devices at the same time, i.e., the session is open and running on multiple devices at same time. Different devices may show an adapted view of the same user interface or the system may have a distributed user interface where different devices play their own roles.

3. **Collaboration scenario.** Several users run the same application on their devices. This collaboration can be either sequential or simultaneous.

All the above scenarios share similar challenges in adapting the user interface to different devices and in synchronizing the data and state between devices. Synchronization of the data and state are important because the devices and users need to be aware of the results of their actions previously or simultaneously done in other devices. This is essential both for immediately transferring the work from one device to another, but also to enable seamless, real-time collaboration.

### 2.1 Liquid User Experience Scenarios

Today, browsers are used to access many Cloud services and Web applications, and users can already use several devices to access those services. In some cases, it is sufficient for users to share a hyperlink (URL with optional parameters) to exactly reproduce and restore the state of the user interface of the application from one browser to another. For example, the hyperlink may be used to identify the video to be played and also the position within the video the playback should start from. In this case, the client navigation state can be reliably identified and

reconstructed elsewhere, assuming that the result of de-referencing the hyperlink is the same everywhere.

With more complex Web applications, so called Rich Internet Applications [5], it may not be possible to use a simple hyperlink to describe the large amounts of local state accumulated on the client during a user work session, and thus it may not be so simple to migrate the application across browsers. For example, Google Docs can be seen as a Liquid Application since one or multiple users can use it from multiple client devices either simultaneously or sequentially. While the state of the edited document (i.e., the data) gets synchronized in real-time, the state of the editor user interface is not. This means that when switching from one browser to another, even if the document remains the same, the editor configuration (e.g., the position of the caret within the document, or the configuration of the tool bar) will not be transferred across sessions.

Our ambition is not limited to centralized cloud services like Google Docs, or even to Rich Internet Applications. Many applications run locally in users devices and only sporadically contact to a server or cloud. At the moment mobile devices have many native mobile applications. Some of them are standard utilities like addressbook, some are mobile interfaces to common Internet services. We foresee that many of them will be implemented as Web applications, but still include a lot of local computation in the device. Local behavior, data and state are needed for better user experience and to cope with less than ideal network characteristics, and to access to local resources, like sensors and the file system. The fact that part of the application and its data resides in the client leads to different kinds of liquidity: liquidity between device and Cloud and liquidity between devices. If the applications or their components can move flexibly between devices and servers, the system may optimize factors like response times, networking costs or battery-life [6]. For instance, if the network latencies slow down the response times, the system may move the communicating components to the same host. Also, if we assume that a substantial part of the application logic and state is stored in the client device, liquid user experiences require liquidity of applications between devices using proximity-based communication.

So far, perhaps the most vivid example of a liquid behavior is the new Handoff capability in Apple iOS 8 [7]. One example scenario of Handoff is one where composing of an email may have started in a phone but finished with a PC that has bigger screen and a real keyboard. The participating devices need to be registered in iCloud with the same identity, and the devices must be able to communicate over Bluetooth. Handoff can be seen as a mechanism to implement a liquid user experience, since it enables sequential screening for a single user. The applications need to be rewritten to take advantage of the Handoff API and pre-installed across all devices; most of the common Apple applications are already compatible with Handoff. Each device runs a specific version of the application, hence the user interface is implicitly adapted to take advantage of the device capabilities (e.g., multi-touch, screen size, and resolution).

In comparison to Handoff, Liquid Software should support heterogeneous devices across software ecosystem boundaries. This way, developers would need

to implement only one application, which then can adapt itself to run everywhere. In our vision Liquid experiences are built using the Web – where applications already now are deployed on demand and through Responsive Web Design [8] are also adapted to fit on the local device display on the fly. Moreover, properties such as openness and freeness from predatory control make the Web a natural choice over native applications that are bound to a particular operating system, manufacturer, or ecosystem [9].

## 2.2 Liquid vs. Solid Software

In the multi-device world, the user experience of the applications may span across multiple devices [10]. Controlling the behavior of the applications as they migrate from one device to the next forms a new field of research, where proximity, gestures, and previous actions define the flow of user interaction. This has been well demonstrated by a video by the glass company Corning ([https://www.youtube.com/watch?v=6Cf7IL\\_eZ38](https://www.youtube.com/watch?v=6Cf7IL_eZ38)), where applications are used seamlessly with personal devices, shared screens, and embedded devices in a fashion that best fits the context as well as the particular use case at hand.

Based on the above, Liquid Software is by no means a single technology, but rather a mindset for developing applications to be executed on multiple, heterogeneous computing devices [2]. Liquid Applications then satisfy the SAFE qualities [11] – they are Scalable, Adaptive with respect to their environment, Flexible thus supporting heterogeneity, and Elastic with respect to their workload. The essential elements that make such software different from traditional, "solid", single-device applications are the following:

- **Code mobility:** dynamic relocation of executable code. One categorization of code mobility has been given by Fuggetta et al. in [12]. In that work *strong mobility* means that the code can move together with its state while in *weak mobility* the execution is re-initialized in the new location.
- **State synchronization:** ensuring that the state of the software (and in particular, its user interface) is preserved after the relocation. Fugetta et al. differentiate *migration* from *cloning* [12]. Liquid Applications take advantage of both of these mechanisms. The use case where the user wants to move the ongoing work from a smart phone to a PC (sequential screening), requires strong mobility with migration since the applications move with their states. In use cases where several users collaborate simultaneously, cloning of the runtime state of the software is needed.
- **Adaptation** The applications as a whole and their components need to adapt to different contexts, runtime environments and hardware capabilities.

Conversely, solid software is characterized by the inability to change its execution environment, without going through a complex and expensive redeployment, reconfiguration, and re-initialization process – or, in general, by associating software with a single computer.

**Data and State.** In this work we distinguish the application data from the application state. With data we refer to the content that users store persistently across usage session, while state is the information that the application needs in

order to continue its own execution after being relocated. This division is similar to the division given in [12] where the data was called *Data Space Management*. In the current Internet some of the resources are local to the device, while some data is available through the network. The user expectation of Liquid Applications is that all relevant data is available regardless of the device, and that modifications done in one location need be synchronized and made visible to all hosts. Companies like Apple, Google or Microsoft are solving this issue by providing data synchronization to the cloud. However, these solutions remain isolated from each other and are difficult to integrate with Web applications due to their file-based abstraction.

### 3 Engineering Liquid Web Applications

Creating Liquid Applications requires revising the role of traditional layers used to design Web applications as well as using new implementation mechanisms.

#### 3.1 Layers Revisited

As explained earlier, Liquid Applications and their architecture vary in three different dimensions 1) simultaneous vs. sequential multi-device usage, 2) single vs. multiple users collaboration, and 3) thin vs. thick client. These dimensions are not necessary binary choices, and there may be a spectrum of intermediate solutions. For example, the "thickness" of Liquid Web applications may vary depending on the available resources and needs at hand. Fig. 1 shows a possible architecture where the application logic is split between server and client and data is persisted by the server-side. The client part of the logic with related state and user interface (UI) are about to migrate to another device. Below we discuss how the liquid quality impacts each architectural layer: User Interface, Logic, Data and State in the light of these scenarios.

**User Interface.** The first challenge is to support the adaptability to different device characteristics and usage contexts, but new challenges emerge when interaction includes collaboration between different devices or users. Simultaneous usage will require that the data model shown through the user interfaces

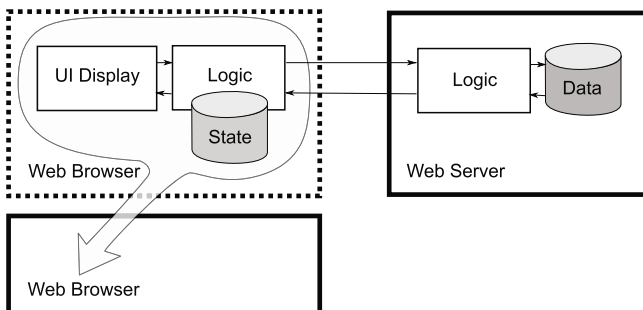


Fig. 1. Liquid Web Application Architecture

of different devices is kept synchronized. Furthermore, the user interface should support interactions that are based on coordinated input and output involving multiple devices. For example, the touch screen of a mobile device could act as an input device for driving the larger display of a laptop. These scenarios could also be combined so that different users look at the same shared display but use their own devices for input. The thicker the client is, the more there are possibilities to implement adaptation in the device, but on the other hand synchronization and coordination between devices become easier in centralized and server-based approaches.

**Logic.** Similarly to the user interface, also the logic that implements Liquid Applications needs be runnable in different client devices and also adaptable different devices and device combinations. If multiple screens or multiple users work on the same application simultaneously, the desired behaviour may be the same for all devices, or different users and devices may play different roles in the overall logic. In the dimension of thin and thick client, the logic may be split between server-side (e.g., in the Cloud) and client-side (e.g., on the mobile device) in various ways depending, e.g., on the available computational power, quality of the network connection, and the battery charge level of the device. The logic may also change its location dynamically. To support these scenarios, components need to be designed to be portable, for example across devices and between device and server, which is already supported by existing Web technologies (e.g., JavaScript, HTML5, Web components).

**Data.** In the case of Liquid Software we assume that the data managed by a Web application is available and accessible from all devices and contexts. Furthermore simultaneous screening requires that changes on data become visible on all screens immediately. If multiple users work with the same data, the system needs to handle ownership and access control of the data as well as conflict detection and resolution. Solving of these problems has a direct dependency on the design decision concerning the thin vs. thick dimension. The desired functionality can be achieved through centralization of the data layer, or by replicating and synchronizing a copy of the shared data across all devices.

**State.** The state of the application encapsulates enough information for migrating or cloning an application to a new location and to continue the execution there. The easiest way to move the state is to serialize the entire process or its whole virtual machine. From the user's point of view it is only important that the user can smoothly move the application to an another device and continue working with minimal disruption. This does not mean that all details on the state need to be preserved, but only the parts that are relevant for the user and for running the application in the new context. It should also be noted, that many applications are client-server systems and part of the state remains on the server. In these cases, the server-part and client-part can move independently from each other. If the client part – usually the user interface – moves, the server part does not need to move. In some systems, all relevant state may be on the server, and changing from one client to another does not require the transfer of any state beyond a URL identifying the server-side state.

### 3.2 Mechanisms

In the context of the Web, liquidity builds on numerous already existing techniques and mechanisms that are already commonly applied, or at very least, they have been experimented with in research projects. Next, we address the building blocks for achieving liquidity in existing Web browsers.

**Responsive Web design.** The primary weapon for tackling user interface issues in different devices used for accessing Web services is Responsive Web Design [8]. In its purest form, a key issue in responsive design is how to design the layout and the elements so that they can easily be customised for different screen sizes and device types, with focus usually only on eventual visual appearance on the screen. In general, due to the differences in browsers used in desktops and mobile devices, the development of Responsive Web Designs calls for architecture where some of the functionalities are always executed on the server side. While this simplifies the development, as these functionalities need not be tested with every possible client device, there are also complicating factors, since the state of the user interface must still be transferred from one device to another. Concrete technologies for composing Responsive Web Designs include using proportion-based grids and flexible images, where element sizing takes place using relative units instead of absolute ones, and CSS3 media queries, where different styles can be used for different devices. For the developer, the above facilities are usually visible through a library that supports Responsive Web Design, such as Bootstrap (<http://getbootstrap.com>) or Foundation (<http://foundation.zurb.com>).

**Liquid Middleware.** A key component in designing Liquid Applications is a middleware system that provides support for code and data mobility. Several such systems have been proposed in the field of agent systems [13–15]. In connection with Web applications, there are three different levels to perform a migration from one device to another:

- **Web application.** In our own work, we have shown how to serialize, migrate and deserialize Web applications in two contexts: 1) *Agent framework* enables HTML5 mobile agents that demonstrate that both application code and runtime state can be moved together with the application code [15], and 2) *Lively Kernel extensions* allow Web applications written using the Lively Kernel Web framework to migrate from one computer to another [16]. Both approaches require that the developer follows certain conventions to record state variables to be transferred as well as to define initialization and termination procedures.
- **JavaScript virtual machine.** In the context of browsers, migration can be realized through the underlying JavaScript virtual machine. Unlike the Web application level migration, migrating the JavaScript VM and the corresponding DOM tree state implies that any Web application can be moved automatically, without requiring developers to explicitly depend on a specific mobility framework, apart from what is necessary for adapting the user interface to different contexts with Responsive Web Design techniques.
- **Operating system level migration.** The most extensive approach to code migration is to transfer the whole virtual machine or operating system image

from one host to another. Obviously, such operation requires extensive virtualization, details of which fall beyond Web technologies. Moreover, with such migration, also considerations regarding the time it takes to transfer the image arise.

**Data Synchronization.** As discussed earlier, there is a need to synchronize updates on data shared by several devices. This can be done in the following ways: 1) keep all data in a centralized server and access data always from that server; 2) on the other extreme, keep all data replicated across all user devices and employ a decentralized, peer-to-peer synchronization tool (such as BitTorrent Sync, AeroFS, and the like); 3) build a system of several primary providers that all other devices use through a unified interface - for example VisualREST [17]. The preferred option depends on the need for simultaneous, concurrent access from multiple devices and thus the need for conflict avoidance/resolution. If a thin client approach is taken, all data is stored in server and addressed from the client. Possible caching would then be implemented with appropriate protocols - like in HTTP. Various Backend as a Service (BaaS) systems allow linking Web applications to hosted backend cloud storage systems and associated services such as push notifications, user management and social network integration. BaaS systems offer well-documented APIs that can be used from Web applications simply by adding a few lines of boilerplate code. For instance, there are systems such as Firebase that focus on making cloud side data storage as simple and effortless as possible. In our designs, this dimension is best considered in the context of Cloudberry [18].

## 4 Discussion

We claim that Web is a suitable platform for Liquid Software. This claim is backed by several technical experiments and prototypes we have been working on [6, 15, 16, 18]. In the following we propose the next step - design and implementation of a general software framework for Liquid Web applications. The framework is still based on Web technologies and standards, but we propose some changes to existing designs.

The users should be provided with an interface that allows smooth control of liquidity. This means that the top-level user interfaces of devices should complemented with new user interface gestures. One example of such gesture is *hyper-dragging* [19] that users can use for pushing applications away from the device they are currently running on. Alternatively, a pull approach is also possible, where users indicate the target device on which the application should be moved on. Another requirement for user interface is adaptability of the user interface. Design and implementation of adaptive user interfaces usually takes extra effort, and we should develop frameworks and tools that make implementation of adaptive user interfaces easy.

A further important component of the software framework is the middleware that supports mobility of the code and execution state. The mechanisms should be as automated as possible and extra effort for the developer should be as small



as possible. For example, serialization and de-serialization of the state should be automated but the developer may need to declare what is the relevant part of the state. Likewise, it should be possible to develop applications that can observe and react to specific events happening during the migration. The ideas presented in HTML5 Mobile Agents [15] can act as a starting point, but the primary use case should be achieving a liquid user experience of Web applications. In addition, the middleware should support application architectures where same component can run both on server and client. The ability to make optimal use of all available resources of the execution environment has been investigated as part of the Liquid Web Services architectural style [11].

Finally, the architecture should include a *Liquid Storage* abstraction that makes relevant data available and synchronized to users and applications who need it. This can be achieved using different centralized or decentralized replication mechanisms that are abstracted away by the storage abstraction, which should work in a way similar to HTML5 local storage. The data API of CloudBerry and EDB [20] include several ideas our design could reuse.

Ultimately, our research goal is to develop a new *liquid.js* framework that utilizes the above framework components and make creation of Liquid Applications easy. The liquid.js toolkit will simplify the migration of stateful components and their adaptation to heterogeneous execution environments.

## 5 Conclusions

From the end-user perspective, liquid software essentially means that the software has a built-in ability to perform adaptive, live migration. The role of the Web is then to act as a platform-independent execution environment, which provides suitable abstractions for relocation, serialization, migration, and adaptation that are needed to develop such applications. All the facilities already exist today, but require use of special frameworks or even application specific code. All these should be either standardized by bodies like W3C, or be simply included to mainstream Web frameworks that today largely define solid application models in the first place.

There are plenty of future research to be carried out in the field of liquid applications. First and foremost, security challenges associated with using multiple devices, with some of them being used by multiple people, are many, and we have largely overlooked them in this phase. Secondly, Internet-of-Things (IoT), as well as its Web-oriented counterpart, Web-of-Things (WoT), introduce challenges that resemble liquid software – managing a large number of computing units, storage and sensors/actuators that are not pre-allocated or pre-configured simply falls beyond what existing application models can deliver. We believe that liquid software provides a suitable metaphor to envision the behavior of software systems as they are deployed in such complex and heterogeneous environments.

**Acknowledgments.** The work is partially supported by the Hasler Foundation (Switzerland) with the Liquid Software Architecture (LiSA) project.

## References

1. Weiser, M.: The computer for the 21st century. *Scientific American* **265**(3), 94–104 (1991)
2. Taivalsaari, A., Mikkonen, T., Systä, K.: Liquid software manifesto: the era of multiple device ownership and its implications for software architecture. In: *Proc. of the 38th IEEE Computer Software and Applications Conference (COMPSAC)*, pp. 338–343 (2014)
3. Hartman, J.H., Bigot, P.A., Bridges, P.G., Montz, A.B., Piltz, R., Spatscheck, O., Proebsting, T.A., Peterson, L.L., Bavier, A.C.: Joust: A platform for liquid software. *IEEE Computer* **32**(4), 50–56 (1999)
4. Google: The new multi-screen world: Understanding cross-platform consumer behavior (2012). [http://services.google.com/fh/files/misc/multiscreenworld\\_final.pdf](http://services.google.com/fh/files/misc/multiscreenworld_final.pdf)
5. Casteleyn, S., Garrigós, I., Mazón, J.N.: Ten years of Rich Internet Applications: A systematic mapping study, and beyond. *ACM Trans. Web* **8**(3), 18:1–18:46 (2014)
6. Babazadeh, M., Gallidabino, A., Pautasso, C.: Liquid stream processing across web browsers and web servers. In: *Proc. of the 15th International Conference on Web Engineering (ICWE 2015)*. Springer, Rotterdam, NL, June 2015
7. Gruman, G.: Apple’s Handoff: What works, and what doesn’t. *InfoWorld*, October 7, 2014
8. Marcotte, E.: *Responsive Web Design*. Editions Eyrolles (2011)
9. Mikkonen, T., Taivalsaari, A.: Cloud computing and its impact on mobile software development: Two roads diverged. *Journal of Systems and Software* **86**(9), 2318–2320 (2013)
10. Levin, M.: *Designing Multi-device Experiences: An Ecosystem Approach to User Experiences Across Devices*. O’Reilly (2014)
11. Bonetta, D., Pautasso, C.: An architectural style for liquid web services. In: *Proc. of the 9th Working IEEE/IFIP Conference on Software Architecture (WICSA)*, pp. 232–241 (2011)
12. Fuggetta, A., Picco, G.P., Vigna, G.: Understanding code mobility. *IEEE Trans. Softw. Eng.* **24**(5), 342–361 (1998)
13. Dömel, P.: Mobile telescript agents and the web. In: *Proc. of the 41st IEEE International Computer Conference. COMPCON 1996*, p. 52 (1996)
14. Feldmann, M.: An approach for using the web as a mobile agent infrastructure. In: *Proc. of the International Multiconference on Computer Science and Information Technology*, vol. 2, pp. 39–45. PTI (2007)
15. Systä, K., Mikkonen, T., Järvenpää, L.: HTML5 agents: mobile agents for the web. In: Krempels, K.-H., Stocker, A. (eds.) *WEBIST 2013. LNBIP*, vol. 189, pp. 53–67. Springer, Heidelberg (2014)
16. Kuuskeri, J., Lautamäki, J., Mikkonen, T.: Peer-to-peer collaboration in the lively kernel. In: *Proc. ACM Symposium on Applied Computing*, pp. 812–817 (2010)
17. Mäkitalo, N., Peltola, H., Salo, J., Turto, T.: VisualREST: a content management system for cloud computing environment. In: *Euromicro Conference on Software Engineering and Advanced Applications*, pp. 183–187. IEEE (2011)
18. Taivalsaari, A., Systä, K.: Cloudberry: An HTML5 cloud phone platform for mobile devices. *IEEE Software* **29**(4), 40–45 (2012)
19. Rekimoto, J., Saitoh, M.: Augmented surfaces: a spatially continuous work space for hybrid computing environments. In: *Proc. CHI*, pp. 378–385. ACM (1999)
20. Koskimies, O., Mikola, T., Taivalsaari, A., Wikman, J.: EDB: a multi-master database for liquid multi-device software. In: *Proc. MobileSoft*. ACM (2015)