Multi-Device Adaptation with Liquid Media Queries

Andrea Gallidabino[0000−0003−4191−7766]
and Cesare Pautasso[0000−0002−2748−9665]

Software Institute
Faculty of Informatics, Università della Svizzera italiana (USI)
Lugano, Switzerland
https://liquid.inf.usi.ch
andrea.gallidabino@usi.ch, c.pautasso@ieee.org

Abstract. The design of responsive Web applications is traditionally based on the assumption that they run on a single client at a time. Thanks to CSS3 media queries, developers can declaratively specify how the Web application UI adapts to each device capabilities. However, CSS3 media queries cannot detect whether Web applications are liquid (i.e., dynamically deployed across multiple devices). As users own more and more devices and they attempt to use them to run Web applications in parallel, we propose to extend CSS media queries so that they can be used to adapt the UI of liquid Web applications. In this paper we present our extension of CSS media queries with liquid-related types and features, allowing to detect the number of devices connected, the number of users running the application, or the role played by each device. The liquid media query types and features defined in this paper are designed and suitable for liquid component-based Web architectures, and they enable developers to control the deployment of individual Web components across multiple browsers. Furthermore we show the design of liquid media queries in the Liquid.js for Polymer framework and propose different adaptation algorithms. Finally we validate the expressiveness of the liquid media queries to support real-world examples and evaluate the complexity of our approach.

Keywords: Liquid Software · Media queries · Multi-device adaptation · Responsive user interface · Complementary view adaptation

1 Introduction

Liquid software [14] stands for a metaphor [20] that associates the behavior of a fluid with software: as a liquid is able to flow into and adapt its shape to any container, liquid software is able to flow across and adapt itself to fit on all the devices it is deployed on. Liquid software allows to seamlessly migrate at runtime parts of an application (e.g. individual components of the user interface) or the whole application from a device to another. Liquid applications are responsive (e.g. they are able to adapt to any device running it), but more importantly they

are also able to adapt to the *set* of devices simultaneously running the application. Finally liquid applications can share their state across multiple devices while keeping it synchronized [18].

Nowadays, due to the improvement of Web technologies with the release of new Web standards (e.g. supporting full-duplex, direct communication between clients), we are witnessing the shift towards more complex and decentralized Web architectures [3], which in turn enable developers to create Web applications featuring support for the liquid user experience.

In our previous works we showed how we designed liquid abstractions for the data and logic layers in liquid Web architectures [5]. In this paper we focus on the user interface layer as we introduce *liquid media queries*, an upgrade to standard CSS3 media queries [2] that allows the developers to create their own CSS style sheets that get activated when their Web applications are deployed across multiple devices. While as part of the liquid user experience, end users can control which user interface components are deployed on each device (e.g., by swiping or drag and drop), developers can use liquid media queries to declarative describe how their applications can automatically react to changes in their deployment environment.

The developers of liquid applications should be able to offer to the users an automatic rule-based deployment mechanism for populating all of the users' devices with pieces of the application they are running, because a misuse of the manual liquid user experience may lead to non-intuitive deployments which contradict with the developer expectations and intent. For example, in case of a picture sharing application, it should be possible to provide constraints for placing the components for taking and selecting pictures on the phones, while the picture viewer component gets deployed on a larger display. This way, users can select which picture to display from their personal smartphone photo library and take advantage of a public device to have a shared slideshow.

The rest of this paper is structured as follows. After reviewing related work in Section 2, we present the design of liquid media queries in Section 3 and show how they are encoded within the Liquid.js for Polymer [4] framework (Section 4). The queries drive the algorithms outlined in Section 5, which are used to automatically adapt a distributed user interface across multiple devices [13] – as shown in the example scenarios of Section 6 – making it possible to shift from the traditional *responsive* UI adaptation [12], to a *complementary* one [15] able to automatically migrate Web components across the set of heterogeneous devices running a liquid Web application.

2 Related Work

In the literature we can find several research topics concerning adaptive multi-device user interfaces [19] such as Distributed User Interfaces (DUI) [11] or Cross-Device Interfaces [17]. All deal with distributed component-based user interfaces deployed across multiple devices [1]. User interface elements can be distributed across the devices either synchronously or asynchronously: when we

talk about asynchronous distribution the devices do not need to be connected in parallel when the UI elements are moved, while for synchronous distribution the devices need to be simultaneously connected [1].

In this paper we deal only with synchronous distribution, and design the *automatic complementary view adaptation* for the components of liquid web applications. In our scenario multiple devices are used together to accomplish a common task, however each device may play a different role and thus display different and complementary visual components. If the set of connected devices changes, then the distributed user interface should flow and adapt accordingly to the new configuration of the environment [10].

In the literature there are several attempts to use rules to describe cross-device user interfaces. Most of them rely on centralised architectures for computing the configuration and then the distribution of the components across multiple device. Zorrilla *et al.* [21] discuss a centralized custom rule-based approach that allows to assigns properties both to components and devices, it scores the best targets for distribution, and then shows and hides the corresponding components depending on the devices they are deployed on. Liquid media queries are also rule-based as they extend the CSS3 media query standard. However, the implementation of our algorithm is meant to be decentralized and involve every device on which the application is running.

Husmann *et al.* [7] implement cross-device user interfaces in a decentralized environment and define a similar rule-based approach. They do not associate the rules to CSS media queries, nor they support multiple CSS style sheets that need to be enabled or disabled on the target devices. Their approach instead deploys the whole application on all the connected devices and then hides the components that should not be displayed. Our approach is more fine-grained as it moves across the devices only the components that need to be deployed, migrating them directly from the device they are currently running on, instead of deploying the whole application from a centralized server.

3 Liquid Media Types and Features

CSS3 media types and features can be used to adapt the user interface of an application to multiple devices by associating a CSS style sheet with some expected device characteristics. Standard media features consider qualities of the Web browser and its environment (e.g., the screen size and resolution, the output media, the device orientation). If the media query matches what the device supports, the corresponding style is activated.

Standard CSS3 media queries are at the foundation for responsive user interfaces that adapt the Web application user interface layout to a single device at the time. However, they lack sufficient expressive power to describe the user interface adaptation in a multi-device environment. For this reason in this Section we introduce and describe new media types and features for liquid web applications (Table 1). Together they enable developers to perform cross-device user interface adaptation by declaratively constraining on which devices a component

Table 1. Proposed media types and features for liquid media queries.

Name	Description
Features	
liquid	The environment has at least two connected devices.
liquid-devices	The number of connected devices.
liquid-users	The number of connected users.
liquid-device-ownership	Whether the device is private, shared or public.
liquid-device-role	The application-specific role of a device.
Types	
liquid-device-type	The type of device(s) running the application.

should be deployed on and by controlling which style sheets should be applied depending on properties of the set of devices connected to their application.

liquid and liquid-devices - Liquid software is strongly tied to multi-device environments, especially when *parallel screening* scenarios are considered [5]. In these scenarios liquid applications must be deployed on multiple devices in parallel. Understanding when the liquid application is running on multiple devices is required for the adaptation. The *liquid* feature refers to any environment that include at least two connected devices, while the *liquid-devices* feature allows to tune this value for specific uses cases. Similarly to CSS3 media queries, it is also possible to define the minimum and maximum values for the *liquid-devices* feature by setting the values for *min-liquid-devices* and *max-liquid-devices* (e.g. it is possible to dynamically change the view of the liquid application when there are at least three connected devices instead of two, or create different views for specific number of connected devices).

liquid-users - In *multi-user parallel scenarios* [5] the liquid application is deployed across multiple devices and multiple users can interact with it at the same time. The *liquid-users* media feature allows to adapt a user interface depending on the number of users connected to the application. The features *min-liquid-users* and *max-liquid-users* can also be used for creating styles for single user applications (e.g. *max-liquid-users:1*) and multi-user application (e.g. *min-liquid-users: 2*).

liquid-device-ownership - The types of access granted to devices can be either *private*, *shared*, or *public*. A *private* device is owned and used exclusively by one user. *Shared* devices are owned by one user, but they can be used by another. *Public* devices (e.g. public displays [16]) can be used by both registered and authenticated users or anonymous guests.

liquid-device-role - The *device role* is an application-specific feature. In Liquid.js for Polymer is possible to configure the connected devices and assign *roles* to them (e.g. *controller*, *console*, *multimedia*). When the role of a device in a liquid application is not tightly bound to the type of device, the device-role can be used by the developers to assign specific styles to the user interface. In order to use the *liquid-device-role* feature, any time a new device connects to the application, it must be configured with the chosen role. The meta data associated with the device can then be changed at any time.

Listing 1.1. Liquid-style element

```
<liquid-style
  liquid         // Default: "true"
  devices="" min-devices="" max-devices="" // Default: ""
  users="" min-users=""  max-users=""      // Default: ""
  device-ownership="" device-role=""       // Default: ""
  device-type=""                           // Default: ""
  priority=""    // Default: "1"
  clone=""       // Default: ""
  css-media=""   // Default: ""
> <!-- CSS Stylesheet --> </liquid-style>
```

liquid-device-type - The latest standard media types only distinguish between *screen*, *print*, or *speech* devices. Depending on the context of the application, it can be useful to distinguish the types of screen devices connected so that they can be assigned to perform certain kind of tasks (e.g. desktop computers are used more for working in an office) [9] , while other devices are more convenient in certain social situations (e.g., smartphones as opposed to laptops are more convenient during meals) [8]. In our current implementation *liquid-device-type* can be set to *Desktop*, *Laptop*, *Tablet*, *Phone*.

4 Liquid Style Element

CSS3 media queries do not allow to define new query types or features, nor they allow to customize existing ones[1]. The solution we design in this paper for extending the standard media queries is to create a new Web component labeled as *liquid-style* inside the Liquid.js for Polymer framework [4].

The *liquid-style* element shown in Listing 1.1 allows the developer to write their own liquid media queries and encapsulate a standard CSS style sheet that is automatically activated when the media query expression is accepted by the device. The attributes of the *liquid-style* component allow developers to assign values to attributes (e.g., `device-role`) that are mapped to the previously defined liquid media types and features by adding the `liquid-` prefix (e.g., `liquid-device-role`). Developers define new liquid media queries by assigning values to the corresponding attributes, as shown in Listing 1.2 and 1.3.

Listing 1.2. Liquid media query expression mapped to liquid-style component attributes

```
@media liquid and (liquid-device-type:phone) {
    body { flex-direction: row; }
}
<!-- Maps to --->
<liquid-style device-type="phone">
    body { flex-direction: row; }
</liquid-style>
```

[1] https://drafts.csswg.org/mediaqueries-4

Listing 1.3. Liquid media query expression including standard CSS media features mapped to liquid-style component attributes

```
@media liquid and
  (liquid-device-role:controller) and
  (min-liquid-users:3) and
  (min-height:900px) {
    :root { background-color: red; }
}
<!-- Maps to --->
<liquid-style device-role="controller" min-users="3"
    css-media="min-height:900px">
    :root { background-color: red; }
</liquid-style>
```

In the first example, the liquid media query expression contains both the *liquid* feature and the *liquid-device-type* type. Inside the *liquid-style* component is not necessary to explicitly set the *liquid* feature to *true*, since it is the default value for the element, while *liquid-device-type* maps to the attribute *device-type*.

The second media query expression contains the liquid media features *liquid-device-role* and *min-liquid-users*, which map directly to the attributes *device-role* and *min-users*. Furthermore the expression also contains the standard media feature *min-height*, any non-liquid part of the query expression is set into the *css-media* attribute.

4.1 Automatic Component Migration and Cloning

Automatic complementary view adaptation is achieved through the liquid media query expressions that both define when styles should be enabled on a device and constrain where the components should be migrated if any device with the appropriate features connects to the application. The *liquid-style* component is designed to be attached directly to a Liquid.js *liquid-component* [4] and bundled with a standard Polymer component. The framework extracts the liquid media query expressions from within every instantiated component and shares them with all other connected devices so that they can check whether they would satisfy the liquid media queries. Whenever a query is accepted on a device, that device becomes a possible target for the migration of the corresponding component. Since it is possible to define multiple *liquid-style* elements inside a component, each can have a different *priority* (see Listing 1.1). The *priority* attribute helps the developers to define multiple styles for different environments, while still being able to influence the migration process, as described in Section 5.

In addition to the migration, the *liquid-style* component provides another liquid user experience primitive for deploying components across multiple devices [5] called cloning, in which components are copied and kept synchronized across two or more devices. Migration enables to redistribute pre-existing user interface components across multiple devices, however it does not allow developers to create adaptive user interfaces with rules for instantiating new components like "*component X needs to be instantiated in all public displays*" or "*component Y*

needs to be instantiated on phones devices, but only once per user". While the migration of a component is obtained by simply adding a *liquid-style* element, cloning components requires additional configuration.

The attribute labeled *clone* in Listing 1.1 is used to enable multiple instances of the same source component to be cloned across multiple devices instead of just migrating it on one of them. The *clone* attribute accepts values in the form of $N-feature$, where N is a positive non-zero integer or the symbol $*$, and $feature \in \{user, device, phone, tablet, desktop, laptop, shared, public, private, role = X\}$.

The value N specifies the maximum number of instances of the source component which should be cloned across the set of available devices which match the liquid media query constraints in relation to the chosen $feature$. Their combination allows to write cloning rules such as:
1-user, clone the component once per user, picking one of their available devices;
1-device, the component is cloned at most once per device type;
2-tablet, up to two component instances are cloned among all available tablets;
***-public**, the component is cloned on all available *public* devices.
***-role=dashboard**, the component is cloned on all devices playing the dashboard role;

The *clone* attribute works in conjunction with the other attributes of the *liquid-style* component, so that the liquid media query expression mapped from the attributes must be accepted on the device so that it is considered a valid target.

5 Liquid UI Adaptation Algorithm

The UI adaptation algorithm operates on three distinct phases: constraint-checking and priority computation, migration and cloning, and local component adaptation. First it decides which devices are suitable for displaying a component encapsulating the liquid media query, then it migrates or clones the component on the highest priority device and activates the corresponding style sheet as soon as the component is loaded on the target device.

5.1 Phase 1: Constraint-Checking and Priority Computation

The constraint-checking phase decides if there is a suitable device in the pool of connected devices that satisfies the liquid media query expressions encapsulated inside the components.

Algorithm 1 computes the matrix of valid target devices in which at least one liquid media expression is accepted. The matrix has size $\#components \times \#devices$. Each element represents with a positive integer the highest *priority* value of all the accepted liquid media queries encapsulated in the component, or *zero* if there are no accepted queries.

The matrix shown in (1) is the *priorityMatrix* produced by Algorithm 1 during the example scenario shown in Figure 2, when both $UserA$ and $UserB$ are connected. There are four instantiated components and seven devices connected

Algorithm 1: Incremental Constraint-checking and Priority Computation

```
Data: Input: priorityMatrix, cloneMatrix
Data: Shared global state: components, devices, users, deviceConfigurations
Data: Event
if Event == component c created then
    Add a new row in the priorityMatrix;
    forall d ∈ devices do
        forall liquid-style in the created component do
            Check if the device accepts the liquid-style and save the highest
              priority in priorityMatrix[c][d] and in cloneMatrix[c][d];
else if Event == component deleted then
    Remove the corresponding component row from the priorityMatrix;
else if Event == device d configuration changed then
    forall c ∈ components do
        forall liquid-style in the component do
            Check if the device accepts the liquid-style and save the highest
              priority in priorityMatrix[c][d] and in cloneMatrix[c][d];
else if Event == device connected || Event == device disconnected || Event ==
  user connected || Event == user disconnected then
    forall c ∈ components do
        forall d ∈ devices do
            forall liquid-style in the component do
                Check if the device accepts the liquid-style and save the highest
                  priority in priorityMatrix[c][d] and in cloneMatrix[c][d];
Result: updated priorityMatrix and cloneMatrix
```

to the application. c_{video}'s liquid media queries (see section 6) are accepted by device d_{laptop}, d_{tv}. At least one query of priority *2* was accepted by device d_{laptop} and at least one query of priority *4* was accepted by devices d_{tv}. d_{phone1} accepts at least one query encapsulated in components $c_{videoController}$, $c_{suggestedVideo}$, the first one with priority *2* and the latter with priority *1*.

$$\text{priorityMatrix} = \begin{array}{c} \\ c_{video} \\ c_{videoController} \\ c_{suggestedVideo} \\ c_{comments} \end{array} \begin{array}{c} \begin{array}{ccccccc} d_{phone1} & d_{phone2} & d_{phone3} & d_{tablet} & d_{laptop1} & d_{laptop2} & d_{tv} \end{array} \\ \begin{pmatrix} 0 & 0 & 0 & 0 & 2 & 2 & 4 \\ 2 & 2 & 2 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 3 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 \end{pmatrix} \end{array} \quad (1)$$

$$\text{cloneMatrix} = \begin{array}{c} \\ c_{videoController} \end{array} \begin{array}{c} \begin{array}{ccccccc} d_{phone1} & d_{phone2} & d_{phone3} & d_{tablet} & d_{laptop1} & d_{laptop2} & d_{tv} \end{array} \\ \begin{pmatrix} 2 & 2 & 2 & 0 & 0 & 0 & 0 \end{pmatrix} \end{array} \quad (2)$$

Algorithm 1 also computes the *cloneMatrix* shown in (2), which has a similar structure to the *priorityMatrix*, but stores only the information about the components that define at least one clone rule in the attributes of the *liquid-style* elements they encapsulate. The matrix has size $\#components_{clone} \times \#devices$ where $\#components_{clone} \leq \#components$.

Liquid.js runs the Algorithm 1 whenever one of the following *events* occurs:

– **A component is created or deleted from a device**. Creating or deleting a components does not affect the acceptance of the liquid media queries of any other components. When a new component is created (or removed), a row is added (or removed) to the *priorityMatrix* and the algorithm recomputes the highest priority score. If the component defines a liquid media query with the clone attribute, then the highest priority value between the clone styles is also stored in the *cloneMatrix*.

– **The meta-configuration of a device is changed**. When the device *type*, *ownership*, and *role* change, the priority values of the corresponding column are updated for both matrices.

– **A device joins or leaves the current session**. These events affect the *devices*, *min-devices*, and *max-devices* features of the liquid media queries, which triggers the recomputation of the whole *priorityMatrix* and *cloneMatrix*.

– **A user connects or disconnects from the application**; Changes to the *users*, *min-users*, and *max-users* features also require a complete recomputation of the *priorityMatrix* and *cloneMatrix*.

5.2 Phase 2: Migration and Cloning

The migration and cloning phase uses the previously computed *priorityMatrix* and *cloneMatrix* to determine on which device each component should be migrated or cloned on. The algorithm prepares a migration plan where each component is assigned a given target device. The choice follows a best fit algorithm so that the number of components running on each device is minimized, thus spreading the liquid Web application across as many devices as possible. If the component instances outnumber the available devices, some of the components will be co-located on the same device still according to their priority. Equation (3) shows the resulting *migrationPlan* computed by the algorithm. c_{video} is migrated to d_{tv} with the highest priority, $c_{comments}$ is migrated to d_{laptop} with the lowest. Once it is ready, Liquid.js uses the migration plan to redeploy the components across the set of devices.

After the migration step is complete, the cloning routine can start. This process exploits the *cloneMatrix* computed in phase 1 and the clone rules associated to the components that need to be cloned. All the devices that were not used in the previous migration step are flagged as candidates for running a cloned component. The candidates are grouped and prioritized following the clone rules, the device that contains the source component that needs be cloned is never considered as a possible target of the cloning, and every component which can be cloned is associated with a list of target devices on which it can be copied. Similarly to the previous step, the algorithm prepares a clone plan that is used by Liquid.js for cloning components. Equation (4) shows the output *clonePlan* computed with the matrix shown in Equation (2) under the constraints of the liquid media queries of the scenario depicted in Figure 2 (see section 6 for the constraints).

The algorithm that computes the migration plan attempts to minimize the number of component instances running on each device. Also, it resolves ties by selecting components based on the order of instantiation. This could be improved by prioritising components with higher score values that have the least number of possible targets devices. This approach works with an initial configuration where all components are initially running on one devices, so the outcome does consider the overall migration cost, seen in terms of the number of migration operations to be performed and the time required to migrate a given component instance. Minimizing such cost would become important when the algorithm is applied to an input configuration of components already instantiated across multiple devices.

$$\begin{aligned}\text{migrationPlan} = [&\{c_{video}, d_{tv}\}, \{c_{suggestedVideo}, d_{phone1}\},\\ &\{c_{videoController}, d_{tablet}\}, \{c_{comments}, d_{laptop2}\}]\end{aligned} \tag{3}$$

$$\text{clonePlan} = [\{c_{videoController}, d_{phone3}\}] \tag{4}$$

5.3 Phase 3: Component Adaptation

The *component adaptation* phase happens once the migration and cloning is complete. Each device checks for each instantiated component which liquid media queries are accepted and activates the associated style sheet. The standard CSS mechanisms for dealing with overlapping selectors take over.

5.4 Run-time Complexity

The complexity of the algorithm we discussed in section 5 depends on three factors: the number of devices (D), the number of the components (C), and the number *liquid-style* elements (S). In the worst case, the run-time complexity of Algorithm 1 is $\mathcal{O}(D * C * S)$. However, the actual run-time complexity depends on the event that triggered the incremental version of the algorithm: $-\ \mathcal{O}(D * S)$ for newly created components; $-\ \mathcal{O}(C)$ for deleted components; $-\ \mathcal{O}(C * S)$ for changed device configurations; $-\ \mathcal{O}(D * C * S)$ for all other events. The run-time complexity of the migration and cloning phase is $\mathcal{O}(C * D^2)$, and the adaptation algorithm explained in subsection 5.3 has complexity $\mathcal{O}(S)$.

The execution for Algorithm 1 can be parallelized as the responsibility for computing the priority Matrix columns can be offloaded on each device, assuming that they all have access to the component liquid style definitions. Each device takes care of updating their columns whenever an event occurs and stores the result in the application shared state, which is automatically synchronized among all devices.

6 Liquid UI Adaptation Example

We show the expressiveness of liquid media queries by designing the *liquid-style* components on a realistic multi-device video player application.

The video player is built with four components (see Figure 1): – the **video** component which displays and plays the video; – the **video controller** component which allows the user to play/pause and seek to a specific time in the selected video; – the **suggested videos** component that displays a list of recommended videos, which can be selected to be played; – the **comments** component where the user can read or post comments about the video.

These components can be deployed across different devices (phones, tablets, laptops, and televisions) owned by one or multiple users.

Listing 1.4. The *liquid-style* elements defined for the **video** component.

```
<liquid-style device-ownership="shared" min-users="2"
    priority="4">
<!-- CSS Style Sheet --></liquid-style>
<liquid-style device-role="display" priority="3">
<!-- CSS Style Sheet --></liquid-style>
<liquid-style device-type="laptop" priority="2">
<!-- CSS Style Sheet --></liquid-style>
```

Listing 1.5. The *liquid-style* element defined for the **comments** component.

```
<liquid-style device-type="laptop">
<!-- CSS Style Sheet --></liquid-style>
```

Listing 1.6. The *liquid-style* element defined for the **video controller** component.

```
<liquid-style device-type="phone" priority="2"
    clone="1-user">
<!-- CSS Style Sheet --></liquid-style>
```

Listing 1.7. The *liquid-style* elements defined for the **suggested videos** component.

```
<liquid-style device-type="phone">
<!-- CSS Style Sheet --></liquid-style>
<liquid-style device-type="tablet" priority="3">
<!-- CSS Style Sheet --></liquid-style>
```

It is best to display the video component (see Listing 1.4) on the devices with big screens, for this reason we define three liquid media query expressions

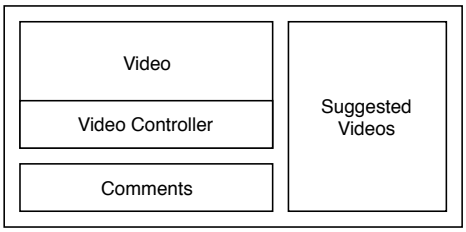

Fig. 1. Liquid video player user interface split into four components: *video*, *video controller*, *suggested videos*, *comments*

	Component	Video	Video Controller	Comments	Suggested Videos
Initial Configuration	*Priority*	2	2	1	3
	Migration Target	laptop 1	phone 1	laptop 2	tablet
User B connects phone	*Priority*	4	2	1	3
	Migration Target	**television**	(phone 1)	(laptop 2)	(tablet)
	Cloning Target		**phone 2**		

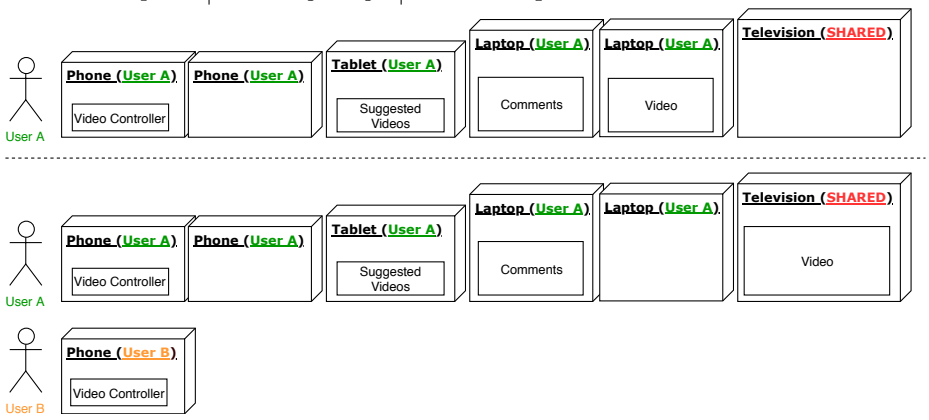

Fig. 2. When a second user connects to the application the video component is migrated to the shared device and a new instance of the video controller is deployed on the new user's phone.

including the attributes `device-type: laptop`, `device-role: display`, and `device-ownership: shared` with different priorities. The rule for `device-type: laptop` has an higher priority over the rule defined for the *comments* component (see Listing 1.5) so that whenever a laptop device is available, the video component is migrated to the laptop. If the user configures the role of any device and assigns the role *display* to it, then this device will have priority over the laptop. Finally, if there are multiple users connected to the application (attribute *min-users:2*), the priority for deploying the *video* component is given to *shared* devices (e.g., a television).

The video controller component (see Listing 1.6) defines a liquid media query expression with the attribute *clone:1-user*. The clone rule migrates the component to a phone owned by a user, then it clones the component for every other user, if they connect at least another phone to the application.

The suggested video component (see Listing 1.7) defines two styles: one for tablets and the other for phones. The tablet style has an higher priority with respect to the phone style.

Scenario 1: Second user connects a phone In Figure 2 we show the component redistribution for a set of devices before and after a second user connects to the application. The initial configuration with only devices owned by $UserA$ is obtained following the priorities associated with the liquid-style elements of each component. Starting from the suggested video component, which migrates to the

	Component	Video	Video Controller	Comments	Suggested Videos
Initial Configuration	*Priority*	2	0	1 (0)	3
	Migration Target	laptop	tablet	television	tablet
Television role set to display	*Priority*	4	0	1	3
	Migration Target	**television**	(tablet)	**laptop**	(tablet)

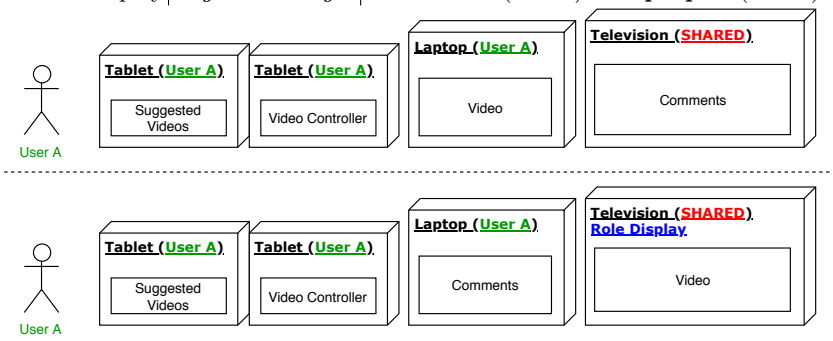

Fig. 3. After the television device changes role configuration, the video and comments components are swapped following different priorities.

tablet, then the video component migrates to a laptop device, because the higher priority rules it holds are not accepted by any other device. The video controller migrates to a phone device, but it is not cloned on both available phones because of the clone rule set to `1-user`. Finally, the comments component migrates to the second laptop device.

After $UserB$ logs in the application and connects an additional phone device, the user interface is redistributed as follows. The video component is migrated to the television device because of the *ownership* and *min-users* rules have now higher priority 4. The video controller component is cloned to $UserB$'s phone. All other components do not migrate.

Scenario 2: Dynamic device role change In Figure 3 we show a dynamic change in the metadata configuration of the connected devices. In this example the initial configuration of the devices is not accepted by at least one of the liquid media queries defined in the video controller component, and the target device for the video and comments components points the same laptop. Starting from the highest priority, the suggested video component is deployed on the tablet and the video component is deployed on the laptop. Since the laptop component is already the target of the video component, the comments component migrates to the television, which was ranked as the next possible target for migration. The video controller component is deployed on the tablet device with the lowest priority.

When $UserA$ assigns the role *display* to the television, the device metadata changes. The user interface redistribution is recomputed and the video component migrates to the television, because the *liquid-style* that defines the property *device-role* is now accepted by the device with an higher priority. The comment

component migrates to the available laptop device. All other components do not migrate.

7 Conclusion and Future Work

This paper describes a rule-based approach that can be used by developers to declaratively specify how the components within a liquid web application can dynamically and automatically be deployed across multiple devices. The liquid media query concept allows developers to define CSS style sheets for Web components in relation to the dynamic multi-device environment they are expected to be deployed on. We identify the main features defining the liquid environment properties (e.g., the number of connected devices, their types, the number of users, various kinds of device ownership, and the application-specific role played by a device). The *liquid-style* element we designed takes care of encoding the liquid media queries so that the Liquid.js for Polymer framework can automatically choose where to deploy a component by evaluating which devices accept the corresponding liquid media query constraints.

The algorithms in this paper are designed under the assumption that the number of devices running a liquid Web application is limited. While this is true for single user environments, in which the number of devices owned by one user is small (3 on average [6]), further work is needed to assess the scalability of the approach to deal with a large number of devices in a multi-user collaborative scenario where it may become impractical to declare liquid media queries matching all possible device combinations.

Another direction for future work concerns the use of logical operators such as *not* and *only* found in standard CSS media queries but which are not supported by the proposed encoding using attributes of the *liquid-style* element.

References

1. Elmqvist, N.: Distributed User Interfaces: State of the Art. In: Distributed User Interfaces, pp. 1–12. Springer (2011)
2. Frain, B.: Responsive Web Design with HTML5 and CSS3. Packt Publishing Ltd (2012)
3. Gallidabino, A., Pautasso, C.: Maturity Model for Liquid Web Architectures. In: Proc. of 17th International Conference on Web Engineering (ICWE2017). vol. 10360, pp. 206–224. Springer, Springer, Rome, Italy (June 2017)
4. Gallidabino, A., Pautasso, C.: The Liquid User Experience API. In: Companion of the The Web Conference 2018 on The Web Conference 2018 (TheWebConf2018). pp. 767–774 (2018)
5. Gallidabino, A., Pautasso, C., Mikkonen, T., Systa, K., Voutilainen, J.P., Taivalsaari, A.: Architecting Liquid Software. Journal of Web Engineering **16**(5&6), 433–470 (September 2017)
6. Google: The connected consumer. `http://www.google.com.sg/publicdata/explore?ds=dg8d1eetcqsb1_` (2015)

7. Husmann, M., Spiegel, M., Murolo, A., Norrie, M.C.: UI Testing Cross-Device Applications. In: Proc. of the 2016 ACM on Interactive Surfaces and Spaces (ISS2016). pp. 179–188. ACM (2016)
8. Jokela, T., Ojala, J., Olsson, T.: A Diary Study on Combining Multiple Information Devices in Everyday Activities and Tasks. In: Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems (CHI2015). pp. 3903–3912. ACM (2015)
9. Kawsar, F., Brush, A.: Home Computing Unplugged: Why, Where and When People Use Different Connected Devices at Home. In: Proceedings of the 2013 ACM international joint conference on Pervasive and ubiquitous computing (UbiComp2013). pp. 627–636. ACM (2013)
10. Levin, M.: Designing Multi-device Experiences: An Ecosystem Approach to User Experiences Across Devices. O'Reilly (2014)
11. Luyten, K., Coninx, K.: Distributed User Interface Elements to Support Smart Interaction Spaces. In: Multimedia, Seventh IEEE International Symposium on. IEEE (2005)
12. Marcotte, E.: Responsive Web Design. Editions Eyrolles (2011)
13. Melchior, J., Grolaux, D., Vanderdonckt, J., Van Roy, P.: A Toolkit for Peer-to-Peer Distributed User Interfaces: Concepts, Implementation, and Applications. In: Proceedings of the 1st ACM SIGCHI symposium on Engineering interactive computing systems. pp. 69–78. ACM (2009)
14. Mikkonen, T., Systä, K., Pautasso, C.: Towards Liquid Web Applications. In: Proc. of the 15th International Conference on Web Engineering (ICWE2015), pp. 134–143. Springer (2015)
15. Mori, G., Paterno, F., Santoro, C.: Design and Development of Multidevice User Interfaces through Multiple Logical Descriptions. IEEE Transactions on Software Engineering **30**(8), 507–520 (2004)
16. Müller, J., Alt, F., Michelis, D., Schmidt, A.: Requirements and Design Space for Interactive Public Displays. In: Proc. of the 18th ACM international conference on Multimedia. pp. 1285–1294. ACM (2010)
17. Nebeling, M., Mintsi, T., Husmann, M., Norrie, M.: Interactive Development of Cross-Device User Interfaces. In: Proc. of the 32nd annual ACM conference on Human factors in computing systems. pp. 2793–2802. ACM (2014)
18. Nicolaescu, P., Jahns, K., Derntl, M., Klamma, R.: Yjs: A Framework for Near Real-Time P2P Shared Editing on Arbitrary Data Types. In: Proc. of the 15th International Conference on Web Engineering (ICWE2015). pp. 675–678. Springer (2015)
19. Paternò, F., Santoro, C.: A Logical Framework for Multi-Device User Interfaces. In: Proceedings of the 4th ACM SIGCHI symposium on Engineering interactive computing systems. pp. 45–50. ACM (2012)
20. Taivalsaari, A., Mikkonen, T., Systa, K.: Liquid Software Manifesto: The Era of Multiple Device Ownership and Its Implications for Software Architecture. In: 38th Computer Software and Applications Conference (COMPSAC2014). pp. 338–343 (2014)
21. Zorrilla, M., Borch, N., Daoust, F., Erk, A., Flórez, J., Lafuente, A.: A Web-Based Distributed Architecture for Multi-Device Adaptation in Media Applications. Personal and Ubiquitous Computing **19**(5-6), 803–820 (2015)