Modular Responsive Web Design using Element Queries

Lucas Wiener[1], Tomas Ekholm[1,2], and Philipp Haller[2]

[1] EVRY AB, Sweden
[2] KTH Royal Institute of Technology, Sweden
lucas.wiener@evry.com,tomase@kth.se,phaller@kth.se

Abstract. It is challenging to develop large, responsive applications with CSS media queries. Responsive elements realized using CSS media queries always depend on the global context, thereby precluding modularity. The lack of true modularity makes certain requirement changes either impossible or expensive to realize.
In this paper we extend Responsive Web Design (RWD) to also include *responsive modules*. We present ELQ, a novel implementation of so-called *element queries* which generalize CSS media queries. Importantly, our design conforms to existing web specifications, enabling adoption on a large scale. Experimental results show speed-ups of the core algorithms of up to 37x compared to previous approaches.

Key words: Responsive web design, Element queries, CSS, Modularity

1 Introduction

Responsive Web Design (RWD) is an approach to make an application respond to the viewport size and other device characteristics. This is currently achieved using CSS media queries which are designed to conditionally design content by the media, such as using serif fonts when printed and sans-serif when viewed on a screen [22]. In order to reduce complexity and enable reusability, applications are typically composed of modules, i.e., interchangeable and independent parts that have a single and well-defined responsibility [15]. In order for a module to be reusable it must not assume in which context it is being used.

The problem is that there is no way to make a module responsive without making it context-aware, due to the fact that media queries can only target the viewport; this means that responsive modules can only respond to changes of the (global) viewport. Thus, a responsive module using media queries is layout dependent and has both reduced functionality and limited reusability [24]. In a world where no better solution than media queries exists for RWD, changing the layout of a responsive application becomes a cumbersome task since it may require many responsive modules to be updated.

The Problem Exemplified. Imagine a company working on a big application that uses media queries for responsiveness (i.e., each responsive module assumes to

have a specific percentage of the viewport size). The ability to change is desired by both developers and stakeholders, but is limited by this responsive approach. The requirement of changing a menu from being a horizontal menu at the top to being a vertical menu on the side implies that all responsive modules break, since the assumed proportionality of each module is changed. Even worse, if the menu is also supposed to hide on user input, the responsiveness of the modules break, since the layout changes dynamically. The latter requirement is impossible to satisfy in a modular way without element queries.

Requirements. The desired behavior of a responsive module is having its inner design respond to the size of its *container* instead of the viewport, to make it independent of its layout context. The W3C has discussed such a feature under the name of *element queries* given its analogy to media queries [23]. We have identified the following requirements: a solution must (a) provide the possibility for an element to automatically respond to changes of its parent's properties; (b) conform to the syntax of HTML, CSS, and JavaScript to retain the compatibility of tools, libraries and existing projects; (c) have adequate performance; (d) enable developers to write encapsulated style rules, so that responsive modules may be arbitrarily composed without any conflicting style rules.

Contributions. This paper makes the following contributions:

- A new design for element queries that enables responsive modules while conforming to the syntax of HTML, CSS, and JavaScript.
- Our approach is the first to enable nested elements that are responsive in a modular way, i.e., modules fully encapsulate any styling required for RWD. As a side effect, responsive modules may also be arbitrarily styled with CSS independent of their context.
- A new implementation[1] that offers substantially higher performance than previous approaches.
- A run-time cycle detection system that detects and breaks cycles stemming from cyclic rules due to unrestricted usage of element queries [24].

2 Overview of ELQ

An *element breakpoint* is defined as a point of an element property range which can be used to define conditional behavior, similar to breakpoints of media queries. For example, if an element that is 300 pixels wide has two width breakpoints of 200 and 400 pixels the *element breakpoint states* are "wider than 200 pixels" and "narrower than 400 pixels".

The main idea is to define element breakpoints of interest so that children can be adapted to the different breakpoint states. As a library, ELQ provides a JavaScript API to registering element breakpoints, and detecting breakpoint state changes. ELQ then observes the elements, in order to automatically let

[1] ELQ, an open-source library (MIT license): `https://github.com/elqteam/elq`

the system know when a breakpoint has changed state. The JavaScript API is extensible through plugins. Mainly, plugins provide alternative behaviors and API's for breakpoint registration and action on breakpoint state changes. In our companion technical report [25] we show an example plugin that provides a grid API similar to the CSS Bootstrap framework.

Default plugins. The default plugins of ELQ let users define element breakpoints by HTML attributes in addition to the JavaScript API:

```
<div class="foo" data-elq-breakpoints-widths="300 500">
  <p>When in doubt, mumble.</p>
</div>
```

The plugins also update element classes to reflect the current breakpoint states, which may be targeted in CSS selectors. For instance, if the element is 400 pixels wide, the element has the two classes `elq-min-width-300px` and `elq-max-width-500px`. For each breakpoint only the min/max part changes, to mimic CSS media queries. This is how the classes may be used in CSS to conditionally style the children:

```
.foo.elq-max-width-300px                          { background-color: blue; }
.foo.elq-min-width-300px.elq-max-width-500px { background-color: green; }
.foo.elq-min-width-500px p                        { color: white; }
```

This is however not sufficient for nestable modules since there is no way to limit the CSS matching search of the selectors. The last style rule specifies that all paragraph elements should have white text if *any* `.foo` ancestor breakpoints element is wider than 500 pixels. Since the ancestor selector may match elements outside of the module, such selectors are dangerous to use in the context of responsive modules. The problem may be somewhat reduced by more specific selectors and such, but it cannot be fully solved for arbitrary styling [24].

To enable nestable modules, the default plugins let us define elements to "mirror" the breakpoints classes of the nearest ancestor breakpoints element (the target of the mirror element). This means that the mirror element always reflects the element breakpoint states of the target. The following is an example of using mirroring to have a `.foo` module contain another `.foo` module:

```
<div class="foo" data-elq-breakpoints-widths="300 500">
  <div class="foo" data-elq-breakpoints-widths="300 500">
    <p data-elq-mirror>...</p>
  </div>
  <p data-elq-mirror>...</p>
</div>
```

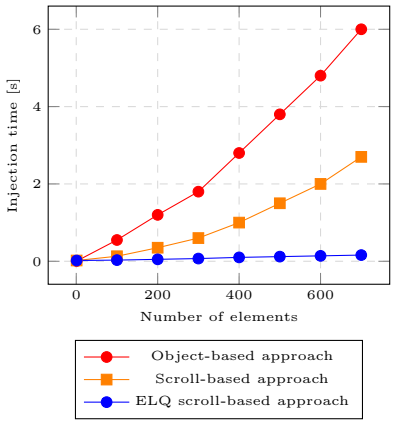
The paragraph elements are told to mirror the nearest breakpoints element by the `data-elq-mirror` annotation. Then, the conditional style of pragraph elements may be written as a combinatory selector:

```
.foo p.elq-min-width-500px { color: white; }
```

Since the breakpoint state class is now combined with the paragraph, the conditional style will only be applied in relation to the actual desired breakpoints element parent.

3 Empirical evaluation

Only the performance of the element resize detection system has been evaluated. This is due to the fact that detecting element resize events entails all the significant performance penalties of ELQ. Fortunately, element resize detection is the common denominator of all automatic libraries and the results of this system can be compared faithfully. Measurements and graphs show evaluations performed in Chrome version 42 unless stated otherwise. Previous implementations use one of two approaches [3]: (a) *object-based* resize detection, which uses `object` elements, and (b) *scroll-based* resize detection, which uses overflowing elements. The approach of ELQ extends the scroll-based approach with batch processing to increase performance [25].



The plot compares the start-up performance of ELQ's scroll-based approach with the other two approaches. ELQ achieves a 37-fold speedup compared to the object-based approach and a 17-fold speedup compared to the scroll-based approach when preparing 700 elements for resize detection. It also performs well when detecting resize events, which it does with a delay of 25 ms for 100 elements.

3.1 Case studies

Modularizing Bootstrap. We have adapted the popular Bootstrap [14] framework to use element queries instead of media queries, to enable its use by responsive modules. By using CSS preprocessors to make the syntax of ELQ element queries similar to media queries, only 0.6% of LOC changes were required. Our companion technical report [25] provides more details, including code examples. In summary we have shown that it is easy to adapt existing responsive code to use ELQ's element queries instead of media queries.

Industrial use of ELQ. We have also been gathering experience with the application of ELQ in large financial applications developed at EVRY. Our practical experience shows that complex applications require a variety of features to be

supported by element queries. Such features can be provided effectively by ELQ plugins. We have noticed that in most of our responsive modules, it has been beneficial for us to use the JavaScript API to conditionally render whole chunks of HTML instead of only changing the style using CSS. Two teams at EVRY have independently come to this same conclusion, and have developed plugins to ease the usage with the different frameworks that the teams are using (Angular and React).

4 Related Work

Syntax extensions. The libraries [5, 8, 16, 1, 21] have in common that they require developers to write custom CSS, unlike ELQ. Since they do not conform to the CSS standard, new features are supported through custom CSS parsed using JavaScript. As shown by [8, 21] quite advanced features can be implemented this way. Additionally, adding new CSS features implies that it is possible to implement a solution to element queries that does not require any changes to the HTML, which may be preferable since all styling then can be written in CSS. However, there are numerous drawbacks with libraries that require custom CSS. Extending the CSS syntax violates the requirement of compatability and also introduces a compilation step which decreases the performance [24].

Resize detection. The libraries [8, 19, 12, 11, 10, 13, 26, 7, 17] simply observe the viewport resize event, which may be enough for static pages, but not enough to satisfy the requirements of reusable responsive modules [24]. Approach [20] does not detect resize events at all. Like ELQ, [1, 9, 16, 21, 4, 18] observe *elements* for resize events. The libraries [1, 9] use polling while ELQ and [16, 21, 4, 18] use different injection approaches. As shown in Section 3, the injection approaches used by related libraries have significantly less performance than the element resizing detection system used in ELQ.

Constraint-based CSS. CCSS [2] proposes a more general and flexible alternative to CSS. The idea of CCSS is to layout documents based on constraints. The Grid Style Sheets library [21] builds upon the ideas of CCSS. While not directly offering element queries, the library enables the possibility to conditionally style elements by element criteria and thus makes it a good candidate to solve the problem of responsive modules. However, the library has two major issues: performance and browser compatibility [6]. In contrast, ELQ only considers element queries, but without browser compatibility limitations and with higher performance.

5 Conclusion

This paper extends Responsive Web Design (RWD) with *responsive modules* through element queries. Our approach is the first to enable nested elements

that are responsive in a modular way, i.e., modules fully encapsulate any styling required for RWD. Our implementation, ELQ, is fully compatible with existing web standards and technologies. The element resize detection of ELQ performs up to 37x better than previous algorithms. We present a case study which shows that changing only about 0.6% of the LOC is sufficient to enable the use of the popular Bootstrap framework in responsive modules. We also report on first commercial usage of ELQ.

References

1. C. Ashton. Localised CSS. `https://github.com/ChrisBAshton/localised-css`.
2. G. J. Badros, A. Borning, K. Marriott, and P. Stuckey. Constraint cascading style sheets for the web. In *Proceedings of the 12th annual ACM symposium on User interface software and technology*, pages 73–82. ACM, 1999.
3. D. Buchner. Backalleycoder. `http://www.backalleycoder.com/`.
4. D. Buchner. `https://github.com/csuwildcat/element-queries`.
5. G. Felipe. MagicHTML. `https://github.com/gabriel-felipe/MagicHTML`.
6. Grid Style Sheets. Element queries with precompilation. `https://github.com/gss/engine/issues/178`.
7. D. Hägglund. breaks2000. `https://github.com/judas-christ/breaks2000`.
8. T. Hodgins and M. Euzière. EQCSS. `http://elementqueries.com/`.
9. A. Hume. `https://github.com/ahume/selector-queries/`.
10. K. Hunaid. `https://github.com/kumailht/responsive-elements`.
11. T. Matanich. `https://github.com/tysonmatanich/elementQuery`.
12. J. Neal. MediaClass. `https://github.com/jonathantneal/MediaClass`.
13. T. Nguyen. SickleS. `http://singggum3b.github.io/SickleS/`.
14. M. Otto and J. Thornton. Bootstrap. `http://getbootstrap.com/`.
15. D. L. Parnas. On the criteria to be used in decomposing systems into modules. *Communications of the ACM*, 15(12):1053–1058, 1972.
16. F. Remy. `https://github.com/FremyCompany/prollyfill-min-width/`.
17. S. Richard. eq.js. `github.com/Snugug/eq.js`.
18. M. J. Schmidt. `https://github.com/marcj/css-element-queries`.
19. J. Stoutenburg. `https://github.com/reusables/breakpoints.js`.
20. M. Stow. Class Query. `https://github.com/stowball/Class-Query`.
21. D. Tocchini. Grid Style Sheets 2.0. `http://gridstylesheets.org/`.
22. W3C. Media queries. `http://www.w3.org/TR/css3-mediaqueries/`.
23. W3C. W3C public mail archive: The :min-width/:max-width pseudo-classes. `https://lists.w3.org/Archives/Public/www-style/2013Mar/0368.html`.
24. L. Wiener. ELQ: Extensible Element Queries for Modular Responsive Web Components. Master's thesis, KTH Royal Institute of Technology, Sweden, 2015.
25. L. Wiener, T. Ekholm, and P. Haller. Modular responsive web design using element queries. *CoRR*, abs/1511.01223, 2015.
26. C. Worrell. `https://github.com/coreyworrell/responsive-elements`.