Allowing Responsive Web Modules

1st Author Name
Affiliation
City, Country
e-mail address

2nd Author Name
Affiliation
City, Country
e-mail address

3rd Author Name
Affiliation
City, Country
e-mail address

ABSTRACT

UPDATED—September 17, 2015. This sample paper describes the formatting requirements for SIGCHI conference proceedings, and offers recommendations on writing for the worldwide SIGCHI readership. Please review this document even if you have submitted to SIGCHI conferences before, as some format details have changed relative to previous years. Abstracts should be about 150 words and are required.

Author Keywords

Authors' choice; of terms; separated; by semicolons; commas, within terms only; this section is required.

ACM Classification Keywords

H.5.m. Information Interfaces and Presentation (e.g. HCI): Miscellaneous; See http://acm.org/about/class/1998/ for the full list of ACM classifiers. This section is required.

INTRODUCTION

- Why modules? Reusability (even across applications), reduced code complexity.
- Why responsive design?
- Responsive Modules of today need to be context aware (thus, not very reusable [they only work in a specific layout]).
- What do we want and why? Modules that are responsive relative to its outer frame.

A module is an interchangeable and independent part of a program that typically has a single and well-defined responsibility. Modular programming is a technique to reduce complexity and enable reusability. In order for a module to be reusable it must not assume in which context it is being used.

Responsive Web Design (RWD) is an approach to make an application respond to the viewport size and device characteristics. This is achieved by using CSS media queries to define conditional style rules. In this paper we will focus on the size responsiveness of modules.

The problem is that there is no way to make a module responsive without it being context-aware, due to media queries only being able to target the viewport. Thus, a responsive module using media queries is layout dependent and has therefore limited reusability.

The desired behavior of a responsive module is having its inner design responding to the size of *its container* instead of the viewport. Only then is a responsive module independent of its layout context.

This can be achieved with the theoretical feature *element/container queries* that enables conditional style rules by an arbitrary element size. This note presents a novel implementation of element queries in JavaScript, and discusses the new possibilities of GUI design.

EXAMPLES OF BROKEN RWD TODAY

- MQ is not the solution to RWD. (MQ was not designed for RWD as the feature was released long before RWD)
- All elements adapt their inner design by the viewport width.
- Menu Example shows how MQ are broken.
- Limitations of MQ regarding font-size (em).

Media queries were designed to enable developers to conditionally design content by the media, such as using serif fonts when printed and sans-serif when viewed on a screen. Therefore, it is only applicable for RWD of non-modular static applications. In a world where no better solution than media queries exists for RWD, changing the layout of a responsive application become a cumbersome task.

Imagine an application that displays the current weather of various cities as widgets, by using a weather widget module. The module should be responsive, so that more information such as a temperature graph over time is displayed when the widget is big. When the widget is small it should only display the current temperature. Users should also be able to add, remove and resize widgets.

Such application cannot be built with media queries. Since the widgets can have varying size, the module cannot change design by breakpoints relative to the viewport. To overcome this problem we must change the application so that widgets always have the same size. This implies that the size of the module and the media query breakpoints are coupled/intertwined, i.e. they are proportional to each other. The problem now is that we have removed the reusability of the

weather module, since it requires the specific width that is correctly proportional to the media query breakpoints.

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 top menu to being a vertical side menu 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 module breaks since the layout changes dynamically.

Additionally, it is popular to define breakpoints relative to the font size. Media queries can only target the font size of the document root, limiting the functionality drastically. With element queries, breakpoints may be defined relative to the font size of the targeted element.

As we can see, even with limited requirements there still are significant flaws with using media queries for responsive modules.

REQUIREMENTS OF A SOLUTION

- Parents should decide the layout of their children, and the children should adapt their inner design accordingly.
- Valid language syntaxes (HTML, CSS, JS).

First, a solution must enable developers to change the design of an element by its parent size. Elements should automatically respond to changes of the parent size so that the correct design can be activated for each size.

Second, a solution must conform to the syntax of HTML, CSS, and JS so that the compatability of tools, libraries and existing projects is retained.

WHY IS A NATIVE IMPLEMENTATION TROUBLESOME?

- Performance issues.
- Cite Tab Atkins of RICG (he states that it is infeasible to standardize this).

A JAVASCRIPT IMPLEMENTATION

- Why is this pragmatic? Compatability, no impact (performance, language) on apps that do not need responsive modules.
- Satisfies the requirements for a solution given above.
- Present Elq's API.
- Present the performance.
- Note drawbacks (but only drawbacks for added functionality!).

DISCUSSION AND SUMMARY OF RELATED WORK

- Performance, APIs, Features.
- The mirror functionality of Elq makes it uniquely suitable for nested modules.

CONCLUSION

- Production ready.
- Probably no standard (or not in a long time).

ACKNOWLEDGMENTS

Sample text: We thank all the volunteers, and all publications support and staff, who wrote and provided helpful comments on previous versions of this document. Authors 1, 2, and 3 gratefully acknowledge the grant from NSF (#1234–2012–ABC). *This whole paragraph is just an example.*