GridFly: Generating and Transforming Web Layouts using Lightweight Grid Templates

Marcio dos Santos Galli
TelaSocial
São Carlos, Brazil
mgalli@telasocial.com

Rudinei Goularte
ICMC, University of Sao Paulo
São Carlos, Brazil
rudinei@icmc.usp.br

ABSTRACT

Modern JavaScript toolkits and standards continue to interoperate, and to support developers positively to design pages with a greater level of separation of concerns, thus ensuring reasonable structure to HTML documents and supporting client interaction with external systems. This paper focuses in the layout area and the demands for responsive design and fluid layout experiences, which is also supported by industry factors, such as screen resolution, which is a response of the availability of Web engines present in tablets, mobile devices, and other appliances.

The project proposes a declarative model that enables the generation and transformation of HTML layout which adheres to good design principles, such as separation of concerns, and modern demands for responsive layout. A demonstration will cover the benefits for developers, first with an use case for modern HTML layout generation based on a lightweight notation based in a 2D grid. The work also covers the use of such declarative notation to achieve ongoing layout transformations that can benefit developer when creating dynamic web applications.

Categories and Subject Descriptors

H.5 [**Information Interfaces and Presentation**]:
H.5.2 User Interfaces
H.5.4 Hypertext/Hypermedia User Issues

General Terms

Human factors, algorithms, design

Keywords

W3C; DOM; HTML5; CSS3; JavaScript; Web; Table-less, Grid Layout

1. INTRODUCTION

GridFly demonstration
`http://labs.telasocial.com/grid-layout`

Web engines are ubiquitous among mobile devices and desktops; and it continues to reach out in a new range of appliances, which motivates developers to deliver interfaces that can scale among different screen sizes. Such practice suggests a reduced cost of development and aligns with good design principles such as separation between content and style.

The general concept of reusing a markup with multiple views is debated long before the modern idea of fluid and responsive user interfaces. In Going to Print [7], the author discusses design principles and the use of CSS Media types [1] for printing online documents, and suggests to avoid the regeneration of markup which was common in the early days of the web.

As for responsive web design, CSS3 Media queries [8] plays a central role, as it allows web pages to have associated style rules based on conditional expressions, thus delivering styles based in particular media features, such as screen resolution. The following example shows the media query and style rules for a page that can be displayed in two different screen conditions.

```
@media (max-width:800px) {
    .cell { width:33%}
}
@media (max-width:320px) {
    #thirdColumn { display:none }
    .cell { width:50%}
}
```

This example has a markup that creates columns implemented with 3 sibling DIV elements which are nested under a main DIV. In order for the inner DIV elements to be next to each other, they depend on the "display:inline" CSS property.

```
<div id='gridrow'>
   <div id='firstColumn' class='cell'>...
   </div>
   <div id='secondColumn' class='cell'>...
   </div>
   <div id='thirdColumn' class='cell'>...
   </div>
</div>
```

The figure 1 illustrates the end results for this layout under the 800x600 and 320x200 case.

Such experiences were commonly written with JavaScript and DOM interfaces. While this can now be achieved using CSS, using modern browsers, there are limitations in

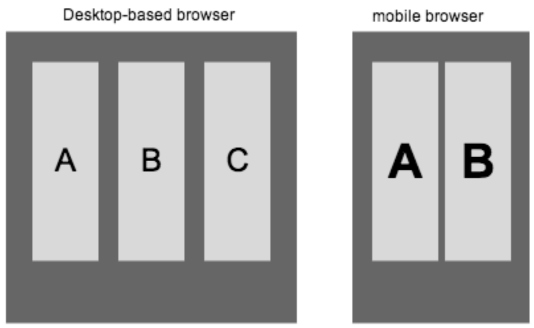

Figure 1: A web page, on the left, displays the 3 column layout. On the right the 3rd column is hidden and font is larger

terms of what CSS can style because modern pages are currently written using nested HTML elements. In the next section, we present more about the case for nested elements, i.e. hierarchical markup, that is commonly found in modern pages.

Our work explores a model that enables the creation of modern layouts using a simple 2D notation representing the grid. This approach offers a further level of separation suggesting that cell arrangements, in a given grid-based layout specification, can be generated or transformed using simplified rules, thus simplifying the process of adapting 2D grid specifications to the hierarchical structure demanded by modern layout techniques.

A prototype is currently available in JavaScript, and it uses an underlying toolkit API that enables developers to create fluid and responsive layouts. Two scenarios are demonstrated in order to cover the benefits of the proposed model. The first, covers the generation of grid layouts using 2D specifications based in short strings - the grid generator. This section will cover the mechanism behind the transformation and also an example that shows an interactive 2D editor that generates a table-less [2] markup. The second scenario involves a layout that gets transformed during an user session and will cover the benefits of using lightweight strings as events of transformation for dynamic pages.

These demonstrations are implemented based in a toolkit currently written in JavaScript. The paper focuses in the benefits for the developer and also highlights the importance for declarative models with a greater level of control when generating and manipulating layouts. In the final section, we will conclude with a focus the strategic long term benefits for developers and the industry in general.

2. TABLE-LESS LAYOUTS AND STYLE

Table-less is a general concept for implementing HTML pages without the table element. A table can also be seen as a grid - with rows and columns and cells arranged within. The following represents a specification for a table with 2 columns and 2 rows with cells A, B, C, and D. With early HTML specifications, a page would require the use of the HTML table element which provides a markup that is closer to the actual 2D representation.

```
A B
C D
```

Such layout can be implemented with tables with ”tr” elements enclosing rows and ”td” elements enclosing the cells.

```
<table>
  <tr>
      <td>A</td><td>B</td>
  </tr>
  <tr>
     <td>C</td><td>D</td>
   </tr>
 </table>
```

The Table element was widely used in the 1990s. Mainly to its intuitive and easy to implement model, since the markup representation is closer to the 2D representation; and due to the limitations provided by CSS. These aspects also contributed to a high availability of WYSIWYG tools back then. With improved CSS being available in browsers, developers found a way to create table-equivalent layouts using elements for content and having the layout control in the CSS. Additionally, with demands for more dynamic sites, developers found difficult to use the DOM to modify tables. As an example, if a developer wants to remove the right column of the table, one solution is iterate through all the rows.

A problem, however, is that more complex grid layouts can demand hierarchical HTML elements, i.e. the concept of containers. The following figure 2 illustrates a visual representation for a page with the following markup. Note that the class=”inline” notation is being used to signal the difference between inline and block-level DIV elements. The actual presentation behaviour for such desired differences is achieved simply using CSS.

```
<div>
  <div class='inline'>A</div>
  <div class='inline'>B</div>
  <div class='inline'>C</div>
</div>
<div>
  <div class='inline'>
      <div> D </div>
      <div>
         <div class='inline'> F </div>
         <div class='inline'> G </div>
      </div>
  </div>
  <div class='inline'>E</div>
</div>
```

The above example presents a case of a very simple table which has a reasonably complex hierarchical structure in terms of markup. So while table-less is elegant, due to style rules being outside the markup, it also requires deep understanding and the developer needs to create a mental model between the 2D representation and the hierarchical model using containers of two main types: block level (which breaks lines) and inline (which can display elements side by side). This may also be a reason that WYSIWYG tools are not common anymore.

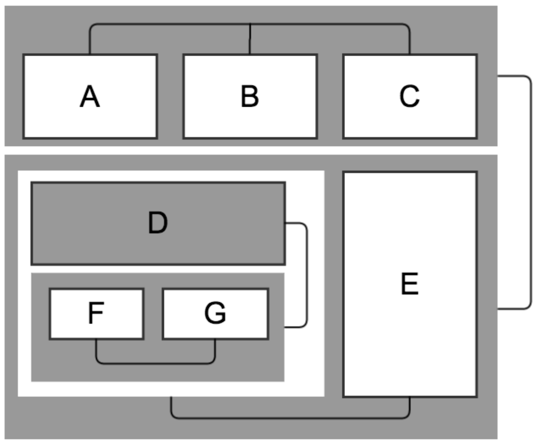

Figure 2: Each square within a square represents a nested HTML container. A bordered square represents a real cell which spaces within the borderless placeholder element. Connectors indicate sibbling DOM elements

3. SCENARIO 1: GRID EDITOR

Our initial case is to offer a model that allows the creation of table-less layouts using a simple to understand specification. We recognise that the actual markup, served to a browser, should remain consistent to the demands for table-less and grid layouts. Figure 3 shows an example with a 2D specification and our proposed model to be used with our transformation toolkit.

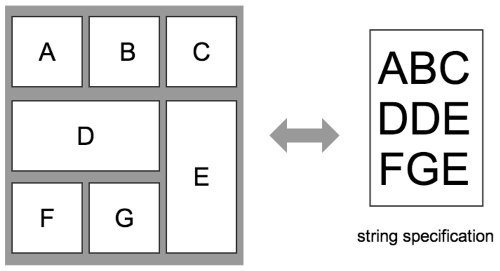

Figure 3: A short string of characters at the right, is used to represent how cells expand in a 3x3 underlying grid

The toolkit basically takes arguments, such as "ABCDDEFGE" and additional parameters, such as the number of columns, and walks through these cells looking for sibling elements - one or more repetitions. If a cell, say E, repeats, it means it's the same cell. As the toolkit walks through, it creates clusters that should never break cells. For instance EE and DFG are contained in in a cluster and this should imply that DIV will contain these cells, aside from the possibility of other inner DIVs within. Our editor demonstration is an application that allows an user to draw "table" using a 2D model, which gets generated, via the toolkit, and becomes markup in hierarchical form as expected by modern techniques. In figure 2, each letter represents a cell however the containers are also indicated with alternating colors.

While dependent on JavaScript, the implementation shows that a potential developer would define its layout using structures that are more declarative in comparison to the nested markup which is what the browser expects. The the above example, the JavaScript API call would look like:

```
// JavaScript API call
grid_generate(3,"ABCDDEFGE");
```

And the markup would have cells one cell for each character, thus A, B, C, D, E, F, and G. Similarly to any table-less layout, the actual cell sizes would be defined in the CSS. These values are inferred as the toolkit walks through the cells and while the model is being assembled by our prototyped engine.

4. SCENARIO 2: GRID TRANSFORMER

Dynamic HTML became widely adopted once browser engines enabled JavaScript accessing DOM interfaces. With further support, such as XML and a data-loading mechanism called XMLHTTPRequest, a new type of web applications gained popularity. Technical models, referred as Inner-browsing [4] and AJAX [5], are indicators of the extent iof interest by developers, also driven by a market trend referred as rich internet applications. The impact was that many pages became complex if taken in consideration the content structure. As an example, a search crawler, when visiting an AJAX-based web page, will parse mainly the declarative portion of the page to build its index.

W3C's JavaScript Web APIs [6] recognises the importance of declarative approaches which improving reusability and creates ground for innovation. The greater adoption of rich applications also impacted in a wide recognition of challenges, such as the challenges of testing [3], and created opportunities for models of pages analysis and toolkits for a more organised and structure development.

Our grid transformer example shows a structured approach to accomplish the generation and modification of a table-less layout for an user session. As shown in the first scenario, it uses the 2D string specification for building the underlying table-less layout. In addition, this demonstration focuses in ongoing transformations that gets applied to the existing content. While it ensures a rich interface in terms of adaptation of layout, it highlights that the use of declarative transformations can go beyond content document and cover layout changes over time based in events. The example starts with the following layout:

```
A B
C D
```

The JavaScript code now involves the concept of a container, which is the location where the grid is going to be placed:

```
<script> grid_gen("2,ABCD","container"); </script>
<body>
     <div id='container'>  ... </div>
</body>
```

When the grid is generated the container element will receive the nested DIV elements accordingly. The demonstration has also an interactive component which is is attached as a behaviour to the page once the HTML is loaded. For each quadrant, a click event observer is attached. When the click event happens, the original transformation (2,ABCD) is applied to the target element the event. The following pseudo-code is an example of what happens after the click:

```
<script>
  // after click
  grid_gen("2,'A1','B1','C1','D1'", targetElement );
</script>
```

The expected effect is the generation of a new inner grid, similar to the prior, placed in the target element that follows from the click event. The demo allows for the creation of a fractal effect as shown by figure 4.

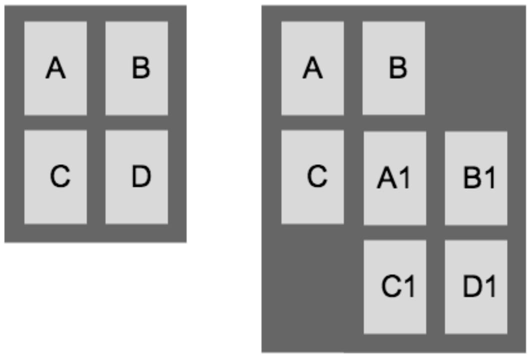

Figure 4: On the left, original quadrants. On the right, after click in quadrant D, it gets populated with a grid similar to the original (A1, B1, C1, D1)

5. CONCLUSION AND FUTURE WORK

HTML was originally designed to be more semantic. On the other hand, the extensive use of JavaScript provided mechanisms that enabled the development of rich interfaces and real-world explorations with use cases pushing Web standards further. In this work, we aim that our functional demonstrations, while written in JavaScript, can contribute to the discussion in the future of grid layouts.

The use of such simplified contextual grids, to specify layout and layout transformations, implies that web page content reaches a new level of separation between content and layout structure. A web page can have a grid layout and yet the content elements can have a linear structure - and perhaps structure that represents better the hierarchy of data instead relying on a need for hierarchical structure that is associated with layout requirements.

Our proposition for a lightweight model also suggests that the rules of transformation can be used to coordinate ongoing transformations within HTML, thus supporting web applications and aiming for a more consistent model enabling external systems to fetch and infer on layout behavior. This model may also support the original notion for a more semantic HTML while ensuring that rich layout transformations which are highly demanded by today's applications from desktop to mobile, and more.

This greater level of separation may also serve as a ground to further exploration for navigational and transformation models that could be generated as a result of analysis with external tools, such as tools using user experience heuristics. The constraints and conditions for layout arrangements could benefit from a wider range of attributes, and even events, thus enabling more adaptability to the context in which applications are being used, as opposite to layout constraints tied to media-only specific characteristics, such as screen resolution.

6. REFERENCES

[1] Css2 media types. `http://www.w3.org/TR/CSS2/media.html`.

[2] Tableless web design. `http://en.wikipedia.org/wiki/Tableless_web_design`.

[3] K. Benjamin, G. V. Bochmann, G. vincent Jourdan, and I. viorel Onut. Some modeling challenges when testing rich internet applications for security.

[4] M. Galli, R. Soares, and I. Oeschger. Inner-browsing extending the browser navigation paradigm. `https://developer.mozilla.org/en-US/docs/Inner-browsing_extending_the_browser_navigation_paradigm`.

[5] J. J. Garrett. What is ajax. `http://www.adaptivepath.com/ideas/ajax-new-approach-web-applications/`.

[6] D. HazaÃńl-Massieux and W3C. Javascript web apis. `http://www.w3.org/standards/webdesign/script#beyond`.

[7] E. A. Meyer. Css design: Going to print. `http://alistapart.com/article/goingtoprint`.

[8] F. Rivoal, H. W. Lie, T. ÃĞelik, D. Glazman, and A. van Kesteren. Media queries. `http://www.w3.org/TR/css3-mediaqueries/`.