International Journal of Engineering Research and Development
e-ISSN: 2278-067X, p-ISSN: 2278-800X, www.ijerd.com
Volume 4, Issue 7 (November 2012), PP. 61-69

Layout Definition of Online Magazines with Splitter Components

[1]István Albert, [2]Hassan Charaf and [3]László Lengyel
Budapest University of Technology and Economics, Hungary

Abstract*—The capabilities of current mobile devices and the quality of their screens reached a level, where online reading experience competes with the printed media. Commercially printed magazines and newspapers commonly apply different grid-based page designs. In case of the online magazines the variable conditions, e.g. screen resolution, user preferences and the actual content require to provide adaptive online document formatting solutions. The paper discusses the adaptation methods and the height/width definition modes of a content-driven template-based layout system. The approach supports online magazine editors to design magazine layouts. Based on the layouts the approach automatically adapts the actual content to different device screens. Current paper concentrates on the concept of the Horizontal Splitter Component and the Vertical Splitter Component, furthermore, discusses the related layout definition considerations. We provide the normalization rules of the different content type and height/width definition method combinations, and as a result, we derive the adaptation methods of compound layout elements.*

***Keywords*—** *Adaptive Layout, Layout Normalization Rules, Content-Driven Layout, Template-Based Layout, Online Magazine Layout*

I. INTRODUCTION

Mobile devices are significant part of our daily life, we use them to reach and consume digital data. Based on different surveys [1] [2] the role of the tablet devices is already quite relevant and they are going to determine the close future of the mobile devices. A reasonable part of online documents have a fixed layout, like Adobe's Portable Document Format (PDF). This inflexibility leads to a poor online reading experience, since the size and resolution of monitors and mobile devices require readers to scroll around in order to read a page [3] [4].

The diversity of mobile platforms and the device capabilities requires providing automatic layout solutions for online newspapers and magazines. The challenge is to automatically adapt the whole digital magazine content, text and graphics, in order to articles look as good on tablet displays of any size as they do in printed media. To improve the limitations of current online reading experience, we need adaptive layout solutions that support automatic layout preparation for a wide range of displays. Ideally, these solutions are easy to use for magazine editors, and provide solutions for different device capabilities, variant user preferences, furthermore, take into account the actually rendered content that has an effect on the final article layout.

The Content-Driven Template-Based Layout System (CTLS) [5] [6] proposes a layout approach that targets tablet devices. The solution facilitates to define column-based templates and the rules (constraints) of the required layout. Both templates and rules can be expressed in an intuitive way, using a high-level, editor-friendly language. The layout engine takes into account the properties of the actual display, the applied templates, required text size, the actual content, and prepares the appropriate layout. A remarkable property is that not only the templates, but the actual content also has an effect on the resulted layout.

In this paper we introduce our content cell related height and width definition methods that magazine editors can use during the template authoring. This approach facilitates that template definition be free of contradictions, and provides a simple correspondence between the content and the width of layout elements. With the approach magazine editors can describe almost all practically relevant layouts.

The rest of this paper is organized as follows. Section 2 briefly introduces the main features of our content-driven template-based layout approach. Section 3 discusses the layout definition considerations. We provide novel height and width definition approaches. Then we introduce both the Horizontal Splitter Component and the Vertical Splitter Component. Next, we go through the normalization rules of the different content and height/width definition method combinations. Based on these results we identify the compound layout elements related adaptation methods. In Section 4 we provide the related work. Finally, conclusions are elaborated.

II. A CONTENT-DRIVEN TEMPLATE-BASED LAYOUT SYSTEM

`The principal challenge behind adaptive document layout for dynamically aggregated content lies in the design of layout templates that are flexible to many different display capabilities and user settings, such as text size. The goal of our approach is that magazine editors can define layout preferences and conditions with minimal effort. Furthermore, the layout engine should make reasonable choices about content display without intervention. In this section we introduce the main properties of our adaptive layout approach.

Defining layout templates. In our layout solution templates refer to column templates. The height of the column is fixed based on the display properties of the device. The ideal width of the column is automatically calculated based on the text size. The screen can be scrolled in the horizontal direction.

The layout is defined with the help of the templates. We use *templates* to define one or more columns, i.e. a template covers the whole column from the top till the bottom. A *layout* is assembled from one or more templates. *Rules* (constraints) are related to both templates and layouts.

The basic building blocks of templates are *rectangular areas* that are arranged in the template and filled with content. Each area receives content from one of the document streams. Figure 1 introduces a sample layout. This layout contains a title text area, a highlighted text area, an image, and four further text areas that present the body text. We use grey background to indicate that a text area is a highlighted text area. This means that the length of the text in this area is fixed, i.e. the text does not flow into this area and flow out from it. The arrows show the flow of the body text among text areas.

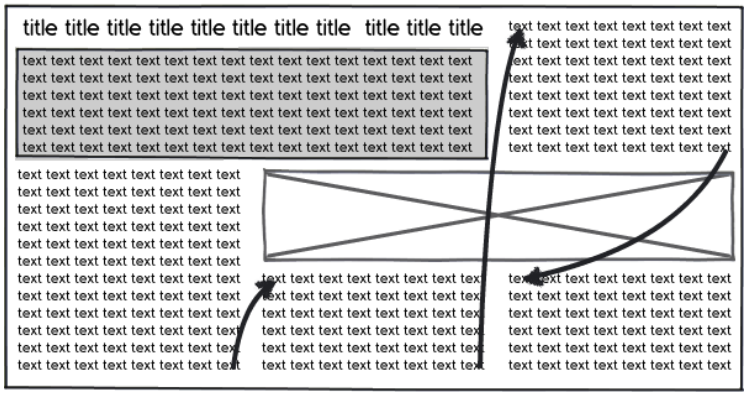

Fig. 1 Example layout template

`Templates can be hierarchical and a template may have substitute templates. They are ordered, and if the original template cannot be applied for the actual content or screen size/resolution then the layout engine uses the next substitute template.

Content streams. We maintain a clean separation between document content and presentation style in our solution. The document content is represented as a set of individual streams, each of which contains content that is laid out sequentially. Streams are differentiated by media type and also by purpose. A magazine article may have several streams. In our system, a document is represented as a set of parallel streams of content.

Managing aspect ratio of images. Rendering a layout while keeping the rules provided by the editor often requires to adapt the image size to the actual conditions. Our approach provides a solution that helps to define the ways how to alter the aspect ratio of images. To achieve it an image is extended or cut. The editor defines the prioritized ideal cuts with focus points on the original image. When a different aspect ratio is required by the layout engine these cuts are extended horizontally or vertically according to the focus point of the original image.

Non-rectangular areas. Basically the layout is built form rectangular areas that are realized as tiles. In certain cases the actual content requires to support non-rectangular areas as well. A solution is that within a rectangular area we define non-rectangular compound areas in which we can allow text to flow around a custom shaped image. Other method is to define exclude areas for images and the text automatically flows around those areas.

Empty spaces. The generic goal of the layout algorithm is that based on the templates and actual conditions arrange the content in a way to utilize the available space. Our solution does not enforce the ratios defined by the template definition. Instead we take it as a starting point, but the final layout is driven by the content itself. This means that taking into account the display properties, the text size, the layout templates, the rectangular areas, the actual content, and the configured requirements related to them, we handle the rectangular areas as tiles and the layout algorithm is about to utilize all of the available space.

Empty spaces that highlight certain part of the article or that make the layout more harmonic are added as rectangular areas of the templates. These areas do not contain text, but they may have background colour, pattern, image or a combination of the previous elements.

Margin and padding. The font size of the text determines the ideal margin and padding widths and heights. This means that the margin and the padding adaptively change when the reader modifies the actual font size. The layout engine automatically calculates and handles the margin and the padding between the rectangular areas of the created layout.

Background. The background can be a colour, a colour gradient, a pattern, an image or a combination of these elements. The ratio of a background image is kept for each layout. If the background image does not cover the whole space then the remaining area is filled with the background colour or a defined colour gradient. If the background image is wider than the actual screen, then with horizontal scrolling we can reach the whole image.

III. CONSIDERATIONS OF THE LAYOUT DEFINITION

This section discusses our layout definition considerations. We introduce both the *Horizontal Splitter Component* and the *Vertical Splitter Component*. Next, we suggest a novel method how to define the height and width of layout content cells. Furthermore, we discuss the possible layout normalization rules, and finally we analyse the adaptation methods of compound layout elements.

A. Basic Considerations

A layout template is built from basic layout elements, like text, image and caption. From these elements optional templates can be assembled. We have two different templates, the single column template, where the elements are arranged one below another, and the single row template, where the elements are arranged one next to another. This means that we discuss both horizontal and vertical splitter related issues. A *splitter component* contains two or more elements ordered in one direction (horizontal or vertical direction). Figure 2 provides an example for a horizontal splitter component, where three elements are arranged one below another, and an example for a vertical splitter component, where three elements are arranged one next to another. The red lines indicate the borders between the different elements.

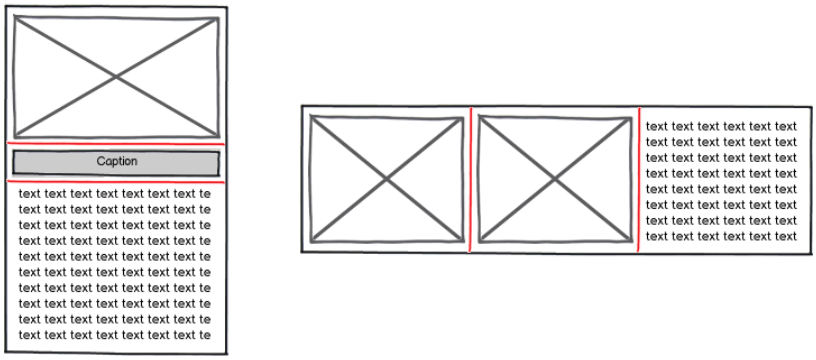

Fig. 2 Example horizontal and vertical splitter components

By default every content type has its adaptation rules. The question that our approach is about to answer is the following: if we put different elements one bellow/next to another and define their height/width calculation method, what will be the resulted adaptation method of this complex component (horizontal/vertical splitter component)?

In order to effectively support the work of magazine editors, the goal is that the approach should target unambiguous concepts, consistent layout arrangement and as few forbidden element combinations as possible.

In a template every element represents a cell. Every cell has two basic properties: (i) the adaptation method (resizing mode) of the contained element, and (ii) the height/width calculation method. An example for a template is shown in Figure 3. There are two elements one below another, where the first one has fixed ratio (X) and the second one has fixed area (+). According to the height calculation method (*), they should have exactly the same height. These notations are discussed in the following part of this paper.

Containment	Height Method
X	*
+	*

Fig. 3 A layout example

Table I: **The degrees of freedom related to the different adaptation methods**

Adaptation method	Width can be set	Height can be set	Both can be set
Free (O)	x	x	x
Fixed Area (+)	x	x	-
Fixed Ratio (X)	x	x	-
Fixed Width (W)	-	x	-
Fixed Height (H)	x	-	-
Fixed (F)	-	-	-
Calc. Ratio (C)	x	x	-

The basic layout elements (text, image and caption) have the following behaviour (adaptation) methods:

- *Free (O)*: it has optional width and height.
- *Fixed Area (+)*: the consumed area is constant *(x*y=Const.)*.
- *Fixed Ratio (X)*: the ratio is constant *(x/y=Const.)*.
- *Fixed Width (W)*: the width is fixed, but the height is optional.
- *Fixed Height (H)*: the height is fixed, but the width is optional.
- *Fixed (F)*: both the height and the width are fixed.

- *Calc. Ratio (C)*: for a given width it calculates the appropriate height, and vice versa.

Table 1 summarizes the degrees of freedom related to the above adaptation methods.
Each of the basic layout elements has a minimal and maximal height/width property. In this paper we do not highlight their role in the algorithms and different considerations, but the implementation takes into account them during the final layout calculations.
Based on the above considerations, our approach suggests the following height definition methods:

- *FixedH (FH)*: The height is defined with a fixed value. It can be a screen rate (e.g. 30% of the screen height) or expressed with ideal row height (the number of the text rows). Allowed adaptation methods: *Free* and *Fixed Height* (the contained element determines the height).
- *FixedHW (FHW)*: The fixed height value is defined like in the case of *FixedH*, but this cell determines the width of the template as well. Therefore, only this type of cells can be added into the table. Allowed adaptation methods: *Fixed Ratio*, *Fixed Area*, *Calc. Ratio*, *Fixed Width* and *Fixed*.
- *Auto (A)*: The height is determined by the contained element. For a given width the height is automatically calculated. The width can be defined by the splitter component (e.g. by a nearby *Fixed Width* type cell) or by a surrounding condition. Allowed adaptation methods: *Fixed Ratio*, *Fixed Area* and *Calc. Ratio*.
- *AutoN (AN)*: *AutoN* stands for *1..n*Auto*. This means that the cell height is proportioned to an automatically sized (*Auto*) cell height, i.e. *N* is a multiplication factor to calculate the final height of the element. Allowed adaptation methods: *Free* and *Fixed Width*.
- **N*: This method defines how to utilize the remaining space. *N* is a multiplication factor to calculate the final height of the element. Allowed adaptation methods: *Free* and *Fixed Width*.

In the same way, according to the above considerations, our approach suggests the following width definition methods:

- *FixedW (FW)*: The width is defined with a fixed value. It can be a screen rate (e.g. 30% of the screen width) or expressed with ideal column width. Allowed adaptation methods: *Free* and *Fixed Width* (the contained element determines the width).
- *FixedWH (FWH)*: The fixed width value is defined like in the case of *FixedW*, but this cell determines the height of the template as well. Therefore, only this type of cells can be added into the table. Allowed adaptation methods: *Fixed Ratio*, *Fixed Area*, *Calc. Ratio*, *Fixed Width* and *Fixed*.
- *Auto (A)*: The width is determined by the contained element. For a given height the width is automatically calculated. The height can be defined by the splitter component (e.g. by a nearby *Fixed Height* type cell) or by a surrounding condition. Allowed adaptation methods: *Fixed Ratio*, *Fixed Area* and *Calc. Ratio*.
- *AutoN (AN)*: *AutoN* stands for *1..n*Auto*. This means that the cell width is proportioned to an automatically sized (*Auto*) cell width, i.e. *N* is a multiplication factor to calculate the final width of the element. Allowed adaptation methods: *Free* and *Fixed Height.*
- **N*: This method defines how to utilize the remaining space. *N* is a multiplication factor to calculate the final width of the element. Allowed adaptation methods: *Free* and *Fixed Height*.

Now, we can define the main goal of our layout algorithm, which differentiates it from several currently applied approaches. We do not concentrate on the size of the elements, i.e. we do not ask its size or requested size, but we query their adaptation method. This method defines its behaviour during the resizing. The algorithm is about to answer the following questions:

- Which cell containment type and cell height/width adaptation method pairs are feasible and which are invalid (contradicting) ones?
- What is the behaviour method of a splitter component? Depending on the contained element what is the resulted adaptation method (e.g. *Free*, *Fixed Ratio*, *Fixed Height* or *Fixed Width*)?
- How to calculate the certain cell heights/widths for a specific device?

The combinations of different containment types and their possible normalizations are discussed in the following sections.

B. Horizontal Splitter Related Layout Definition Normalization Rules

In order to reduce the number of the different content type and height definition method combinations, we consolidate the similarly behaving cases. This process includes the normalization of both of the height definition methods and the adaptation methods as well. As a result we end with a manageable number of cases. We use the following special notations:

- *?*: It signs a cell where optional content type or height definition method can be put. Of course the actual content and method should fulfil the conditions provide by the surrounding elements.
- *!*: Invalid case.
- *../..*: We use slash as a separator to enumerate the different input and output cases of the layout normalization. For example *O / O / W*.

Furthermore we use the already introduced cell height definition methods: *FixedH (FH)*, *FixedHW (FHW)*, *Auto (A)*, *AutoN (AN)*, and **N*.
We introduce the consolidation of **N* and *AutoN* height definition methods in Figure 4 with numbers 1, 2 and 3. For example in the case of the first consolidation we realize the followings:

- Two cells, where both have *Free (O)* adaptation method with a **N* height definition, behave in the same way as a single *Free (O)* cell with a **N* height definition method.

- Two cells, where the first has *Free (O)* and the second has *Fixed Width (W)* adaptation method with a **N* height definition, behave in the same way as a single *Fixed Width (W)* cell with a **N* height definition method.
- Two cells, where both have *Fixed Width (W)* adaptation method with a **N* height definition, represent an invalid case.

Fig. 4 Consolidation of **N* and *AutoN* height definition methods (cases 1, 2 and 3), and consolidation of the *FixedH (FH)* height definition method (case 4)

The consolidation consideration of the *FixedH (FH)* height definition method is presented also in Figure 4 with number 4. Figure 5 depicts the height related consolidations of the *Fixed Area (+)*, *Fixed Ratio (X)* and *Calc. Ratio (C)* adaptation methods.

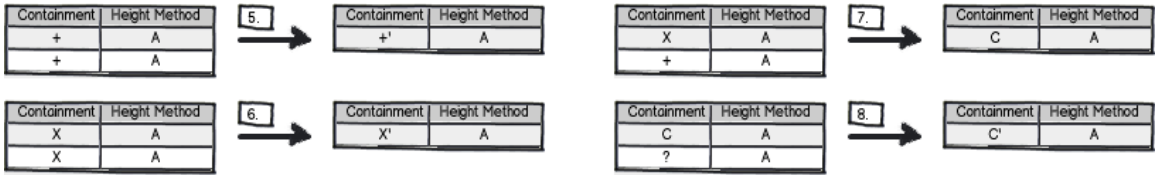

Fig. 5 Height related consolidation of *Fixed Area (+)*, *Fixed Ratio (X)* and *Calc. Ratio (C)* adaptation methods

Figure 6 summarizes the results of the horizontal splitter related layout definition normalization rules. The figure shows the possible containment type and height definition method pairs.

Containment	Height Method
X / + / C / F	FHW
X / + / C	A
O / W	*N
O	FH

Fig. 6 Summary of the horizontal splitter related normalization

With this result we can enumerate the possible content type combinations. In the next section using this information we analyse the adaptation methods of the horizontal splitter component.

C. The Horizontal Splitter Component Adaptation Methods

Using the horizontal splitter related normalization results we investigate the different content type combinations and determine the resulting adaptation method of the horizontal splitter component. If the splitter contains one cell it corresponds to the adaptation method of the actual containment.

Figure 7 introduces the possible content combinations when the horizontal splitter component contains two content cells. In the figure, we numbered the different cases and denoted the resulting adaptation method of the actual combination. For example the case with number 3 contains a cell with an optional content (*?*) and a cell with a *Free (O)* content. The related height definition methods are *FixedHW (FHW)* and *. The resulted adaptation method of the splitter component is *Fixed Width (W)*.

Figure 8 provides the possible content combinations for three and four cell horizontal splitter components. These figures continue the numbering of the different cases and also provide the resulting adaptation method of the actual combination.

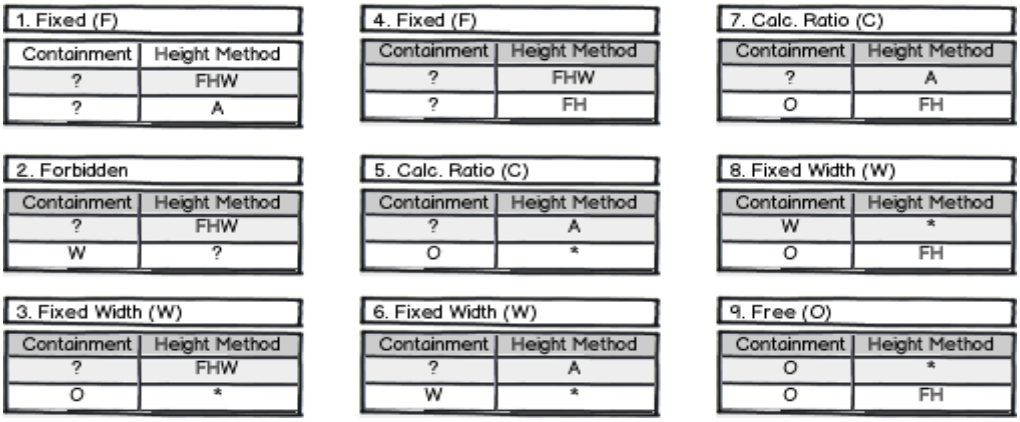

Fig. 7 Two content cell related combinations and the resulting adaptation methods (horizontal splitter)

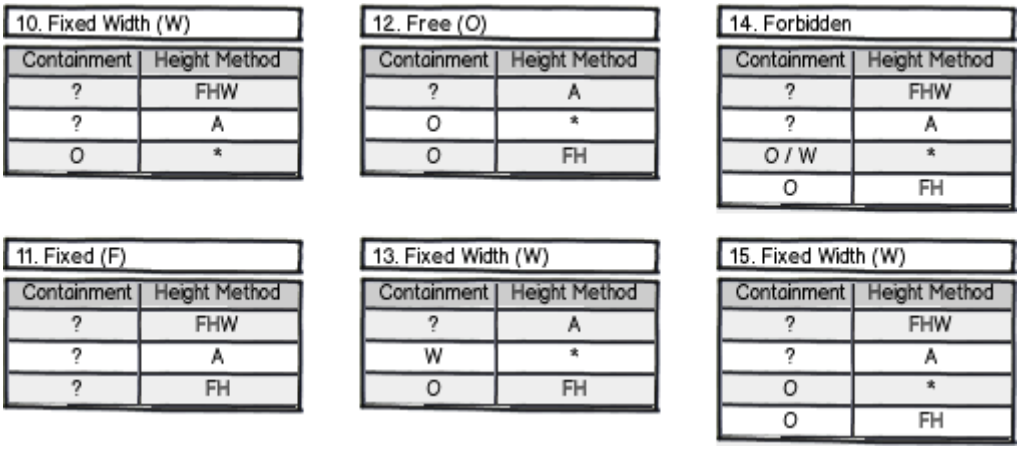

Fig. 8 Three and four content cell related combinations and the resulting adaptation methods (horizontal splitter)

Among the discussed content combinations, the case with number 2 introduces a content combination which results a contradiction related to the resulting width. The combination 14 is also forbidden but the same situation (contradicting width) is handled by the case 2.

D. Vertical Splitter Related Layout Definition Normalization Rules

Similarly to the horizontal splitter component, in the case of the vertical splitter component we can reduce the number of the different content type and width definition method combinations. During the consolidation we use the already introduced special notations (*? !*, and *../..*), and the defined width definition methods: *FixedW (FW)*, *FixedWH (FWH)*, *Auto (A)*, *AutoN (AN)* and **N*.

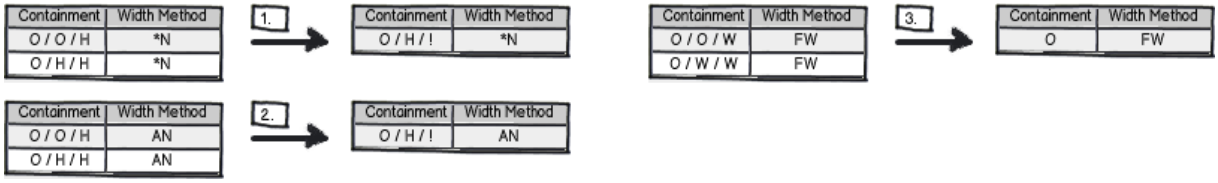

Fig. 9 Consolidation of **N* and *AutoN* width definition methods (cases 1 and 2), and consolidation of the *FixedW (FW)* width definition (case 3)

Figure 9 introduces the consolidation of **N* and *AutoN* width definition methods (number 1 and 2), and the consolidation consideration of the *FixedW (FW)* width definition method (number 3).

Figure 10 depicts the width related consolidations of the *Fixed Area (+)*, *Fixed Ratio (X)* and *Calc. Ratio (C)* adaptation methods.

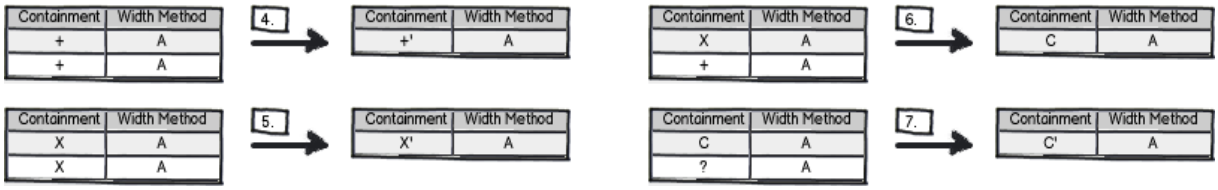

Fig. 10 Width related consolidation of *Fixed Area (+)*, *Fixed Ratio (X)* and *Calc. Ratio (C)* adaptation methods

Fig. 11 Summary of the vertical splitter related normalization

Figure 11 summarizes the results of the vertical splitter related layout definition normalization rules. This figure shows the possible containment type and width definition method pairs. This result facilitates to enumerate the possible content type combinations. Based on this table, in the next section, we analyse the adaptation methods of the vertical splitter component.

E. The Vertical Splitter Component Adaptation Methods

Using the vertical splitter related normalization results we introduce the different content type combinations and determine the resulting adaptation method of the vertical splitter component. If the splitter contains one cell it corresponds to

the adaptation method of the actual containment. Figure 12 introduces the possible content combinations when the vertical splitter component contains two content cells.

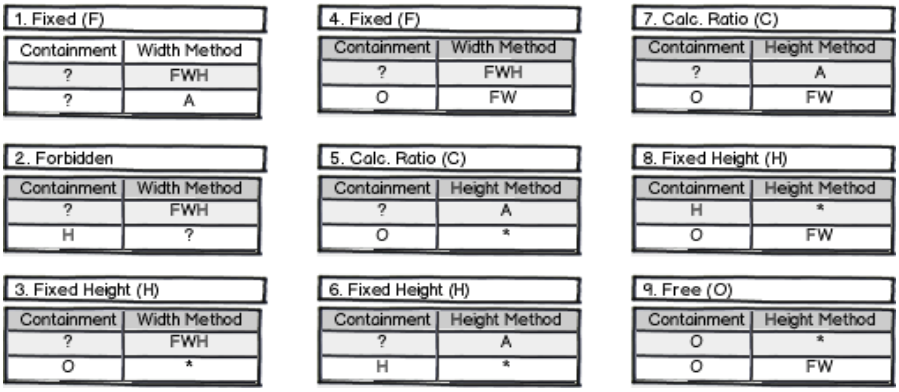

Fig. 12 Two content cell related combinations and the resulting adaptation methods (vertical splitter)

Figure 13 provides the possible content combinations for three and four cell vertical splitter components. The only forbidden case is the content combinations with number 2 that results a contradiction related to the resulting height. The combination 14 is also forbidden but the same situation (contradicting height) is handled by the case 2.

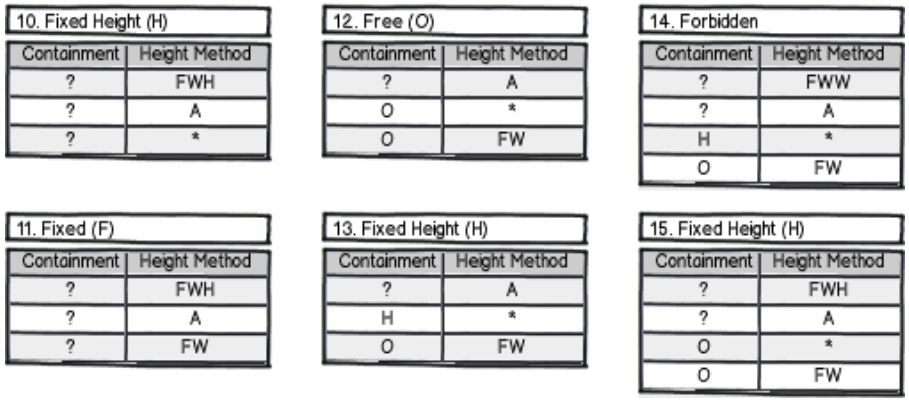

Fig. 13 Three and four content cell related combinations and the resulting adaptation methods (vertical splitter)

F. Summary

The discussed approach facilitates to calculate the adaptation method of complex elements. Therefore, we can build hierarchical templates, where every cell can contain not only a single element but a whole template as well. The layout algorithm calculates the resulting adaptation method on every level of the template hierarchy, thus finally we get the resulting adaptation method of the full compound component as well.

The advantages of the proposed cell height and width definition mode are as follows:

- There are no contradictions. The above mentioned cases are the two invalid and therefore forbidden cases. Optional other combination can be defined.
- There is a simple correspondence between the content type and the cell height/width: the content defines the possible cell height/width and there are no tangled rules.
- The approach is flexible, because with these methods we can describe almost all practically relevant requirements.

The combination of horizontal and vertical splitter components results that not only single column or row templates but optional ones can be built from the layout components.

IV. RELATED WORK

This section reviews the most relevant technologies and related layout approaches.

HyperText Markup Language (HTML) [7] is the main and most popular markup language for displaying web pages and other information that can be displayed in a web browser. HTML elements form the building blocks of all websites. HTML allows images and objects to be embedded and can be used to create interactive forms.

HTML documents have *a what you see is all you get* nature. Web page layout is quite important to provide better look to websites. There are many websites, which arrange their content in multiple columns thus they are formatted like a magazine or newspaper. This is easily achieved by using tables or division or span tags. Sometime we use Cascading Style Sheets (CSS) [8] [9] as well to position various elements or to create backgrounds or colourful look for the pages.

Currently HTML and CSS tables are still not powerful enough [3]. It is not possible to specify non-rectangular compound cells in which for instance we can allow text to flow around an image. It is also not possible to specify that text flows sequentially through a number of cells, allowing for instance multicolumn layout in tables. The current layout

algorithm does not handle multi-paragraph text in cells or multi-column cells. The constraints on column width that may be specified by the designer are limited. Constraints spanning more than one table are not allowed. For instance, it is not possible to specify that columns in two different tables should have the same width.

Unlike CSS 2, which is a large single specification defining various features, CSS 3 [10] is divided into several separate documents called modules. Each module adds new capabilities or extends features defined in CSS 2, over preserving backward compatibility. Regarding to our current layout approach goals the main achievements of the CSS 3 are the multi-column layout solution, the different text flow possibilities, and the feature that makes possible that the layout adjusts to the actual screen width.

A popular unified programming model for building rich user experiences on Windows platform that incorporate UI, media, and documents is Windows Presentation Foundation (WPF) [11] [12]. The core of WPF is a resolution-independent and vector-based rendering engine that is built to take advantage of modern graphics hardware. WPF extends the core with a set of application development features that include Extensible Application Markup Language (XAML) [13], controls, data binding, layout, 2-D and 3-D graphics, animation, styles, templates, documents, media, text, and typography.

We know that the layout of UI controls is critical to application usability. Therefore, arranging controls based on fixed pixel coordinates may work for a limited environment, but as soon as we want to use it on different screen resolutions or with different font size it will fail. WPF provides a set built-in layout panels that help to solve the common requirements. The five most popular layout panels of WPF are as follows: Grid Panel, Stack Panel, Dock Panel, Wrap Panel, and Canvas Panel.

The motivation of the work provided by Jacobs et al. [4] is quite similar to our motivation, i.e. to bring online publication design to the level of its paper-based counterpart. They proposed grid-based design principles that automatically adapt content to appealing page layouts and that match the displays on which they will appear. They introduced the area and collected the most important questions. Next they started to work out a layout solution that is based on page templates. The main difference between their suggested layout approach and our approach is that we apply column templates and the horizontal scrolling is allowed in our case.

Schrier et al. [14] build on the prior work on adaptive grid-based document layout [4], which allows users to design grid-based document layouts that adapt to different window dimensions. That work showed that high-quality, adaptive document layout is possible through constraint-based layout templates. The work focuses on high-quality layout of content with a fixed document structure. Schrier et al. [14] extend those adaptive, grid-based layout ideas to the rendering of dynamically aggregated documents. They presented an interaction model for viewing and reading the documents rendered with their system. They worked out a template language for building templates that include multiple possibilities for displaying each piece of content, with rules for choosing between them. Unfortunately, the solution does not provide graphical template design tool. Further major limitations of the approach are that it requires designers to author templates in XML code, and they apply a one-way constraint solver for the layout engine, rather than a general solver.

The approach is promising and has certain common feature with our approach. One important common property of the solutions is the separation of content and style, which allows designs to accommodate unknown content. This is not a new idea, several standards support this separation, e.g., the Extensible Stylesheet Language (XSL) and Cascading Style Sheets (CSS).

V. CONCLUSIONS

The paper has introduced the layout definition considerations of our content-driven template based layout approach. The approach provides a language, a tool set and the related layout algorithms for online magazine editors. Editors define the templates and assemble the magazine layouts that is driven by the actual content and rendered for the actual mobile device.

We have introduced the concepts of the *Horizontal Splitter Component*. This component facilitates to arrange optional number of layout elements one below another. Furthermore, we have discussed the concept of the *Vertical Splitter Component* that facilitates to arrange optional number of layout elements one next to another. We have defined the adaptation methods of the basic layout elements. Besides, we have suggested a novel approach for defining the height and width of layout elements. The paper has also provided the normalization rules of the different content type and height/width definition method combinations, and as a result we have summarized the adaptation methods of compound elements, i.e. elements contained by a horizontal/vertical splitter component.

The relevance of tablet devices and the number of the device owners is continuously growing. Device displays have reached the technical level which facilitates that the on-screen reading be a rival for the printed documents [1] [2]. Furthermore, we have several reasons to believe that when documents can look as good on the screen as they do in print, the on-screen reading experience will exceed the experience of reading on paper. This is because both desktop and mobile devices provide several opportunities for content and style customization, and there is a possibility for animation and interactivity as well. These features are going to make the on-screen reading experience superior in several ways.

We believe that our approach addresses several open issues related to the current online magazine layouts, and makes easier the life of the magazine editors and provides a more enjoyable result for the readers.

ACKNOWLEDGMENT

This work is connected to the scientific program of the "Development of quality-oriented and harmonized R+D+I strategy and functional model at BME" project. This project is supported by the New Széchényi Plan (Project ID: TÁMOP-4.2.1/B-09/1/KMR-2010-0002).

REFERENCES

[1]. Gartner survey 2010. [Online]. Available: http://www.gartner.com/it/page.jsp?id=1529214

[2]. Vision: Developer Economics 2012. [Online]. Available: http://www.visionmobile.com/devecon.php

[3]. N. Hurst, K. Marriott, Towards optimal table layout, 2005. Available: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.60.3858

[4]. C. Jacobs, W. Li, E. Schrier, D. Bargeron, and D. Salesin, "Adaptive document layout", Commun, ACM 47, 8, pp. 60-66, 2004.

[5]. I. Albert, H. Charaf and L. Lengyel, "The Reference Layouts of the Content-Driven Template-Based Layout System – A Technical Report", Budapest University of Technology and Economics, Hungary, 2012.

[6]. Content-Driven Template Based Layout System (CTLS) webpage. [Online]. Available: https://www.aut.bme.hu/CTLS

[7]. HTML 4.01 Specification. [Online]. Available: http://www.w3.org/TR/html401/

[8]. CSS2.1 Specification. [Online]. Available: http://www.w3.org/TR/CSS21/syndata.html#q10

[9]. E. A. Meyer, Cascading Style Sheets: *The Definitive Guide*, Third Edition, O'Reilly Media, 2006.

[10]. CSS 3 Multi-column Layout Module. [Online]. Available: http://www.w3.org/TR/css3-multicol/

[11]. C. Anderson, *Essential Windows Presentation Foundation (WPF)*, Addison-Wesley, 2007.

[12]. A. Nathan, *WPF 4 Unleashed*, Sams, 2010.

[13]. M. Dalal and A. Ghoda, *XAML Developer Reference*, Microsoft Press, 2011.

[14]. E. Schrier, M. Dontcheva, C. Jacobs, G. Wade, and D. Salesin, "Adaptive layout for dynamically aggregated documents", in *Proceedings of the 13th international conference on Intelligent user interfaces (IUI '08)*, ACM, New York, NY, USA, pp. 99-108, 2008.