The mobile industry has developed immensely in recent years. With this development came more diverse mobile platforms and a variety of device capabilities. Moreover, within this dynamic environment came the need for further advancements. Our work focuses on providing automatic layout solutions for online newspapers and magazines. Consumers want to utilize the special capabilities of their specific devices. This requires that the magazine content adapt to the actual device, e.g., account for screen size, aspect ratio and preferred font size. This requirement highlights the importance of automatic layout approaches. In this paper, we introduce the Horizontal Splitter Algorithm of the Content-Driven Template-Based Layout System (CTLS). The CTLS is a template-based, online magazine layout approach with a unique property: the automatically created magazine layout is driven not only by templates and device properties but also supports the actual content. The Horizontal Splitter Component of the approach facilitates to assemble hierarchical templates in an efficient way. We discuss the equations related to different layout elements and introduces the way how the splitter component derives the resulted adaptation method based on the equation forms of the contracted layout elements.

Keywords: Adaptive Layout, Content-Driven Layout, Template-Based Layout, Online Magazine Layout

1. INTRODUCTION

We live in an information-based, consumer society, in which mobile devices play a significant role in our daily lives. Mobile devices support web browsing, multimedia, and reading different document formats, such as PDF, HTML, or ePub. Based on different surveys [3] [10], the role of tablet devices will determine the immediate future of mobile devices. There are numerous device vendors with a variety of application programming framework. The diversity of these devices and their screens require adaptive layout solutions that utilize the different device capabilities. A reasonable amount of online documents have a fixed layout, like Adobe’s Portable Document Format (PDF). This means inflexibility, which results in a poor online reading experience. Because of the size and resolution of mobile devices, readers have to continually scroll around in order to read a single page [6] [7]. This indicates a problem that requires a solution. Our solution is to automatically adapt the entire digital magazine content, based on the device properties and user preferences. The goal is to achieve articles that look as good on tablet displays (of any size) as they do in printed media. The layout adaptation is required when the reader opens the magazine article or when he/she makes modifications e.g. text size. Therefore, not only differences in tablets but also different settings require rendering an entirely new layout.

Online quality layout and design is mainly based on different table/grid solutions, e.g., HTML tables [5] and CSS [8] or the different layout engines of certain programming frameworks, like Windows Presentation Foundation [9]. We know with certainty, it is possible to produce grid-based page design using HTML. Unfortunately, designers mostly apply fixed-size tables that do not adapt to different displays. We aim to achieve quality online publication and improve the online reading experience. If we take into account the diversity of screens, considering sizes and resolutions, preparing the appropriate layout for each would mean a greater degree of complexity and higher expenses for quality publication design. We can conclude that approaches are required to automatically calculate the appropriate layout on the fly.

The Content-Driven Template-Based Layout System (CTLS) [1] [2] proposes a layout approach that targets tablet devices. The solution facilitates in defining column-based templates and the rules (constraints) of the required layout. Both templates and rules can be expressed in a natural way, using a high-level, editor-friendly language. The layout engine considers the properties of the actual display, applied templates, required text size and actual content, and then prepares the appropriate layout. What sets our approach apart is that not only the templates, but the actual content also has an effect on the resulted layout.

In this paper we discuss the Horizontal Splitter Algorithm of the Content-Driven Template-Based Layout System. The approach does not concentrate on the size of the elements, i.e. the algorithms do not require the requested size of the
layout elements, but instead work with adaptation methods of the elements. The adaptation method defines the behavior of the layout elements during the resizing phase. A layout template is often hierarchical therefore the adaptation must be handled for these types of templates. We address the handling of layout components by providing equation sets which describe their behavior. This paper introduces the horizontal splitter algorithm and discusses its behavior, based on the adaptation methods related equation forms.

Section 2 briefly introduces the features of our content-driven template-based layout approach. Section 3 discusses the details of the horizontal splitter algorithm. Explicitly, we provide the layout definition normalization rules and the horizontal splitter component adaptation methods. Next, we introduce the algebraic structure of our approach. Finally, offer conclusions and remarks

2. THE CONTENT-DRIVEN TEMPLATE-BASED LAYOUT SYSTEM

The Content-Driven Template-Based Layout System (CTLS) [1] [2] provides an adaptive document layout approach. CTLS facilitates to design layout templates that are flexible to many different display capabilities and user settings, such as text size. With this approach, magazine editors can define layout preferences and conditions with minimal effort. This section summarizes the main properties of our adaptive layout approach.

In CTLS, 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 horizontally.

The layout is defined with the assistance of the templates. We use templates to define one or more columns, i.e. a template covers the whole column from top-to-bottom. A layout is assembled from one or more templates. Rules (constraints) relate 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 1a introduces a sample layout. This layout contains a highlighted text area, four images, and three further text areas that represent the body text. We use grey background to indicate that a text area is highlighted. When a text area is highlighted, the area is fixed, i.e. the text neither flows into this area nor out from it. The arrows show the flow of the body text among text areas.

![Figure 1. Sample basic layout, b. Example horizontal splitter component](image)

Templates can be hierarchical and a template may contain substitute templates. The templates are ordered. This means that if the original template cannot be applied for the actual content or screen size/resolution then the layout engine automatically and sequentially applies the next substitute template.

Aside from these elements, the approach supports the following features: different content streams, managing aspect ratio of images, non-rectangular areas, handling empty spaces, calculating margin and padding, and managing backgrounds [1].

3. THE HORIZONTAL SPLITTER ALGORITHM

This section introduces our approach, the Horizontal Splitter Component, and provides the details of its functions. This component facilitates to place layout elements, one below another, and calculates the resulting adaptation method. We
provide the equations related to the behavior of different layout elements and discuss how the component derives the resulting adaptation method, based on the equation forms of the combined layout elements.

### 3.1 The Basics of the Horizontal Splitter Component

Our layout templates are built from basic layout elements: text, image and caption. A splitter component contains two or more elements ordered in one direction, either horizontally or vertically. In a Horizontal Splitter Component elements are arranged in descending order. Figure 1b provides an example, in which three elements are arranged one below another. The red lines indicate the borders between the various elements.

Every primitive content type has its default adaptation mode. The questions are: What happens with the splitter component if we stack different elements and define their height calculation method? What will be the resulted adaptation method of the horizontal splitter component?

Layout templates contain hierarchically assembled cells. Every cell has two basic properties: (i) the adaptation method (resizing mode) of the contained element, and (ii) the height calculation method. The contained element can be a simple or complex layout element, e.g. another horizontal splitter component.

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

- **Free text**: Free (O): containing optional width and height.
- **One text column**: Fixed Width (W): the width is fixed, but the height is optional. (x = Const.)
- **Header image**: Fixed Height (H): the height is fixed, but the width is optional. (y = Const.)
- **Fixed image**: Fixed (F): both height and width are fixed. (x = Const.1, y = Const.2)
- **Resizable image**: Fixed Ratio (X): the ratio is constant. (x/y = Const.)
- **Caption (finite text)**: Fixed Area (+): the consumed area is constant. (x \* y = Const.)
- **Calc. Ratio (C)**: for a given width, it calculates the appropriate height, and vice versa. This is a generalized version of the last two options.

The behavior of the elements related to certain adaptation methods are defined with equations. Table 1 summarizes the adaptation methods related equation forms.

<table>
<thead>
<tr>
<th>Table 1: The adaptation methods related equation forms</th>
</tr>
</thead>
<tbody>
<tr>
<td>Equation ID</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>2</td>
</tr>
<tr>
<td>3</td>
</tr>
<tr>
<td>4</td>
</tr>
<tr>
<td>5</td>
</tr>
<tr>
<td>6</td>
</tr>
<tr>
<td>7-10</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>

Aside from the above considerations, our approach suggests the following height definition methods:

- **Fixed (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 cell content adaptation methods: Free and Fixed Height (the contained element determines the height).

- **FixedHW (FHW)**: This cell determines both the height and width of the template. Therefore, only one cell with this type can be inserted into a table. If a fixed height value is defined similarly to FixedH, then the allowed cell content adaptation methods are: Fixed Ratio, Fixed Area, Calc. Ratio and Fixed Width. If a fixed width value is defined then the allowed cell content adaptation methods are: Fixed Ratio, Fixed Area, Calc. Ratio and Fixed Height. If neither height nor width value is defined, then the cell content adaptation method must be Fixed.
In the previously introduced concepts, we have emphasized the main motivation of our layout algorithm. We do not concentrate on the size of the elements, i.e. inquire about their size or requested size, but instead are interested in their adaptation method. This method defines their behavior during layout calculation. The algorithms of the approach provide answers to the following questions:

- Which cell container type and cell height adaptation method pairs are feasible and which are invalid (contradicting)?
- What is the behavior method of a splitter component? Depending upon the contained element, what is the resulted adaptation method (e.g. Free, Fixed Ratio, or Fixed Height)?
- How are the specific cell heights for a particular device calculated?

All possible layout element combinations should be examined. In order to minimize the number of combinations they are normalized which enables us to handle only those cases which behave differently. The combinations and their possible normalizations are discussed in the next section.

### 3.2 Normalizing Layout Combinations and Calculating the Resulting Equations

In order to finish with a manageable number of cases, we consolidate the similarly behaving height definition methods and adaptation methods.

We assume that all horizontal splitter combinations can be composed, i.e. the layout element equations can be combined and the result is another equation. This requires calculating the resulting equation for optional existing equation forms (e.g. \( x*y=c \)) and not only simple ones (e.g. \( x=c \)). The relevant questions we will answer are the following: What will result from the possible combinations of the layout equations? Will the scope remain within the already introduced equation forms (Table 1) or are new equation forms necessary? If the combinations introduce new equation forms, what will happen during the combination of these new equations? When will we reach (if possible) a closed list of the equation forms?

The above questions raise the topic of hierarchically applied horizontal splitter components. In this approach, it is necessary to understand the resulting equation forms, because these forms should be applied during the combination of two or more horizontal splitter components. From the algorithm’s point of view, this represents a recursive calculation method.

The key motivation of the layout element composition is as follows. A layout template can contain several layout elements, which are often hierarchically defined. In the case of a horizontal splitter component, we want to replace it, as well as the contained layout elements, with a similarly behaving single element. The supplanting element should have the same layout behavior as the original horizontal splitter. In this way, we can efficiently manipulate the layout calculation of hierarchically defined compound layout templates, even in the case of a lengthy hierarchy. In the next section, Table 3 summarizes the possible compositions of the element-related, layout equations.

In order to reduce the number of the different content types and height definition method combinations, we consolidate the similarly behaving cases. This process includes the normalization of both of the height definition methods as well as the adaptation methods. 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 placed. Of course, the actual content and method should fulfill 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 \(N\).

In Figure 2, the consolidation of \(N\) and \(AutoN\) height definition methods are provided by cases 1 and 2. The consolidation consideration of the \(FixedH\) (FH) height definition method is presented in case 3. Cases 4-7 depict the consolidations related to the \(Fixed Area\) (+), \(Fixed Ratio\) (X) and \(Calc. Ratio\) (C) adaptation methods. Furthermore, the table in the bottom-right corner summarizes the results of the layout definition normalization rules. This table shows the possible containment type and height definition method pairs.

![Figure 2](image)

**Figure 2** Consolidation of \(N\) and \(AutoN\) height definition methods (cases 1-2), consolidation of \(FixedH\) (FH) height definition method (case 3), consolidation of \(Fixed Area\) (+), \(Fixed Ratio\) (X) and \(Calc. Ratio\) (C) adaptation methods (cases 4-7), and the summary of the normalization

Based on case 2, we can analyze the resulting adaptation method:

- The general equation of the first cell is

\[
y_2 = f(y_3)
\]

- The equations of the second cell are \(x_2 = (\text{they have the same width})\) and \(y_2 = n \cdot y\), where \(n\) is a constant. \(n\) denotes how the height of the actual cell is related to the Auto height.

- The resulted equations are \(x_2 = x_3 = (\text{they have the same width})\) and \(y_3 = y_2 + y_2 = y_2 + (n \cdot y_2) = y_2 \cdot (1 + n) = f(y_2) \cdot (1 + n)\). The second one is the equation of the original Auto cell multiplied with a constant value.

If we multiply the original equation \(f(y)\) with a non-zero constant then we get the same equation form. Therefore, the result is the same adaptation method. This is also true for all of the equation forms provided in Table 2.

Sample equation form consolidations are the followings:

- \(1+1=1\) and \(1+2=2\), these are related to case 1 in Figure 2.
- \(6+6=6\) is related to case 4 in Figure 2.
- \(5+5=5\) is related to case 5 in Figure 2.
- \(5+6=9\) is related to case 6 in Figure 2.

Using the 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 of the actual containment. In Figure 3, cases 1-9 introduce the possible content combinations when the horizontal splitter component contains two content cells. The figure denotes the resulted adaptation method of the actual combination and the number of the resulted equation forms. For example, case 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 \(N\). The resulted adaptation method of the splitter component is \(Fixed Width\) (W). The tables also provide the numbers of the related equation forms.

In Figure 3, cases 10-13 provide the possible content combinations for three cell horizontal splitter components and cases 14-15 for four.

Among the discussed content combinations, case number 2 introduces a content combination which results a contradiction related to the resulting width. Combination 14 is also forbidden, but an identical situation is handled in
When the algorithm has to determine the resulting behavior method of a splitter component, it suffices to contract the appropriate equations. There is no need to account for the actual height definition methods. The height definition methods affect the validation (what type of primitive layout elements can be placed into particular cells), as well as the calculation of the final layout. This corresponds to our goal: to determine the behavior of compound elements (splitter components) based on the equation forms.

The composition of layout elements is commutative (see next section). This means that the order of the elements in a splitter component is optional. This is also related to our motivation. If we are able to prove this statement, then the layout editor should disregard the order of certain layout elements, i.e. the layout elements can be optionally rearranged.

3.3 Algebraic Structure

In this section, we discuss the algebraic, structure-related considerations of the splitter components. We handle the layout equations as a Set of equation sets. Furthermore, we prove certain properties of this Set.

Table 2: The elements of the Horizontal Splitter Equation Set

<table>
<thead>
<tr>
<th>Equation ID</th>
<th>Equations</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>( y = 0 )</td>
</tr>
<tr>
<td>1</td>
<td>( - )</td>
</tr>
<tr>
<td>2</td>
<td>( x = c )</td>
</tr>
<tr>
<td>3</td>
<td>( y = c )</td>
</tr>
<tr>
<td>4</td>
<td>( x = c_1 ) ( y = c_2 )</td>
</tr>
<tr>
<td>5</td>
<td>( \frac{x}{y} = c )</td>
</tr>
<tr>
<td>6</td>
<td>( x \cdot y = c )</td>
</tr>
<tr>
<td>7</td>
<td>( y = c_1 \cdot x + c_2 )</td>
</tr>
<tr>
<td>8</td>
<td>( y = \frac{c_1 \cdot x + c_2}{c} )</td>
</tr>
<tr>
<td>9</td>
<td>( y = c_1 \cdot x + c_2 \cdot \frac{1}{x} )</td>
</tr>
<tr>
<td>10</td>
<td>( y = \frac{c_1 \cdot x + c_2 \cdot \frac{1}{x}}{c} )</td>
</tr>
</tbody>
</table>
Table 3: The operational table of the splitter component

<table>
<thead>
<tr>
<th>Eq. ID</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
</tr>
<tr>
<td>1</td>
<td>-</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>-</td>
<td>-</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>3</td>
<td>4</td>
<td>7</td>
<td>8</td>
<td>7</td>
<td>8</td>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>4</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>5</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>5</td>
<td>9</td>
<td>7</td>
<td>10</td>
<td>9</td>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>6</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>6</td>
<td>10</td>
<td>10</td>
<td>10</td>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>7</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>7</td>
<td>10</td>
<td>10</td>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>8</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>8</td>
<td>10</td>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>9</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>9</td>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>10</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>10</td>
</tr>
</tbody>
</table>

The elements of a Set are based on the equations describing the behavior of the layout elements. More importantly, we are interested in the form of the equations. The elements are a set of equations (zero, one or two equations). The name of the Set is Horizontal Splitter Equation Set (HSE Set). The elements of the Set formed by our splitter component are provided in Table 2. The Set has 11 elements.

The Horizontal Splitter Equation Set is equipped with a single binary operation $M \times M \rightarrow M$. A binary operation is closed by definition, but no other axioms are imposed on the operation [4].

The binary operation of the HSE Set is the contraction operation (+). The contraction simplifies two sets of equations with the help of a third equation set. For the horizontal splitter component, the third equation set is always as follows:

$x_2 = x_2 = x_1$

$x_2 = x_2 + x_1$

The result of the operation is a set of equations (an element of the HSE Set). The operational table of the contraction operation is summarized in Table 3.

Proposition 1. The contraction operation is commutative for the Horizontal Splitter.

Proof. The operation has two operands (two sets of equations). The variables of the first operand are $x_1$ and $x_2$. The variables of the second operand are $x_3$ and $x_4$. Regarding the contraction operation, the order of the two operands affects only the indexes of the variables. In the third set of equations (the hard-wired helper equations) the $y$ variables are added ($x_1 + y$). The addition is a commutative operation and, it follows that, the indexes can be commuted. This means that the order of the original set of equations is optional.

Consequence: Commutability means that the order of the layout elements in a horizontal splitter component does not affect the resulting behavior of the splitter.

Proposition 2. The contraction operation can be applied for the Vertical Splitter component and its contraction operation is commutative.

Proof. The vertical splitter is similar to the horizontal splitter, except that the elements are placed side-by-side. Therefore, the statement follows the proof of Proposition 1.

Proposition 3. The contraction operation cannot be applied for the composition of horizontal and vertical splitter components.

Proof. The composition of horizontal and vertical splitter components can create new forms of equations. For example, Figure 4 depicts a sample layout with a vertical and horizontal splitter component embedded into each other. Layout components denoted with numbers 1, 2 and 3 are simple resizable image (Fixed Ratio (X)) and finite text (Fixed Area (+)) elements. The red area (indicated by the number 4) is a horizontal splitter component. This splitter contains the layout elements 2 and 3. The whole template (denoted in number 5) is a vertical splitter component. This splitter contains the layout elements 1 and 4. In this way, the horizontal splitter (number 4) is embedded into the vertical splitter (number 5).

The equations related to the template are the followings (the indexes denote the number of the layout element):
Based on the operational table (Table 3), the contraction operation is idempotent:
\[ A + A = A \]

This contradicts the definition of the contraction operation; the binary operation of the HSE Set is closed.

**Figure 4** Example layout with a horizontal and a vertical splitter component

**Proposition 4.** The HSE Set and the contraction operation are associative.

**Proof.** The HSE Set is associative, if the contraction operation (+) is an associative operation. The contraction operation is associative, if \( \forall x, y, z \in M \), where \( M \) is the HSE Set, \( (x + y) + z = x + (y + z) \). Based on the operational table (Table 3), we can assume all of the cases. For example:

\[
\begin{align*}
(1 + 2) + 3 &= 1 + (2 + 3) \\
2 + 3 &= 1 + 2 \\
2 &= 2
\end{align*}
\]

Because of the space restrictions, we do not deduce all of the combinations.

**Proposition 5.** The contraction operation is idempotent.

**Proof.** Based on the operational table (Table 3), the contraction operation is idempotent: \( A + A = A \). There are two forbidden cases (2+2, and 4+4), in which the layout algorithm would result inconsistency.

The identity element (or neutral element) of the HSE Set is a special type of element in a set with respect to a binary operation of that set. It leaves other elements unchanged when combined with them. By default we do not have an identity element (Table 1) but it has been defined for the HSE Set (\( \gamma = A \)).

4 Conclusion

We know that mobile devices play a significant role in our lives. Therefore, appropriate mobile applications and supporting methods are required. Such a supporting method is the automatic and adaptive layout calculation. In this paper, we have introduced the horizontal splitter algorithm and the related considerations of our content-driven template based layout system.

The discussed method facilitates to calculate the adaptation method of complex layout elements, i.e. entire templates. We can also build hierarchical layout components. The layout algorithm calculates the adaptation method at each level of the layout hierarchy, and what results is the re-adaptation method of the whole compound component.

We have introduced the Horizontal Splitter Component, discussed the adaptation method (resizing mode) of the basic layout elements and the approach providing height calculation methods. The layout definition normalization rules and
the horizontal splitter component adaptation methods were detailed in this paper. Finally, we have introduced the algebraic structure of the provided approach, defined the Horizontal Splitter Equation Set and proved certain properties of the Set.

We believe that the results of our research activities contribute to the achievement of adequate, adaptation solutions which support the efficient automatic layout calculation on mobile platforms.

ACKNOWLEDGEMENTS

This work was partially supported by the European Union and the European Social Fund through project FuturICT.hu (grant no.: TÁMOP-4.2.2.C-11/1/KONV-2012-0013).

REFERENCES