The Solution Area and Fitness-Based Algorithms of the Content-Driven Template-Based Layout System

ISTVÁN ALBERT, HASSAN CHARAF, LÁSZLÓ LENGYEL
Department of Automation and Applied Informatics
Budapest University of Technology and Economics
HUNGARY
{ialbert, hassan, lengyel}@aut.bme.hu

Abstract: - People use their mobile devices every day to access a wide variety digital content. The diversity of mobile platforms and that of mobile device capabilities requires providing automatic layout solutions for online content. For the purposes of this paper, our solutions focus on online magazines and newspapers. The Content-Driven Template-Based Layout System (CTLS) is a template-based online magazine layout approach. This approach facilitates in defining hierarchical layouts from basic layout elements and splitter components. The goal is to effectively calculate the resulting adaptation method for a particular layout element at any level of the layout hierarchy. This paper introduces both the solution area-based algorithm and the fitness-based algorithm of the CTLS approach. These algorithms apply inequalities rather than equations. Inequalities, represented by solution areas (polygons), provide the flexibility within the approach.

Key-Words: - Adaptive Layout, Content-Driven Layout, Template-Based Layout, Online Magazine Layout, Splitter Algorithm

1 Introduction

Mobile devices play a significant role in the daily lives of the majority of people living in a consumer-based society [1] [2]. Many people own one or more mobile devices, from numerous device distributors, with a variety of special features, capabilities and application programming framework. The diversity of these devices and their screens require adaptive layout solutions that can utilize the capabilities of each different device. High-quality, automatic document formatting is a difficult problem, with many obstacles [3] [4]. The main challenge is to automatically adapt the whole digital magazine content so that articles look as good on a tablet display, of any size, as they do in printed media.

Content-Driven Template-Based Layout System (CTLS) [5] [6] templates are column templates. The height of a column is fixed, based on the display properties of the device. Then the ideal width of the column is automatically calculated, based on the text size, and horizontal scrolling is also made possible.

Templates allow the manipulation of each unique layout and are also applied to define one or more columns, i.e., a template covering the entire column, from top to bottom. Layouts consist of one or more templates. Rules (constraints) relate to and are applied in both templates and layouts. Rectangular areas that are arranged in the template and filled with content provide the basic composition of a template. Text, images and captions make up the basic layout elements. In [7], we have demonstrated the reference layouts of the Content-Driven Template-Based Layout System.

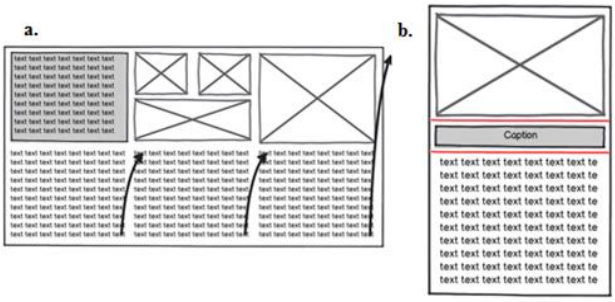


Figure 1. a. Sample basic layout, b. Example horizontal splitter component

The adaptation methods are the focus of the layout elements and size is not deeply considered. The behavior of the layout is decided by the methods during the layout calculation. The adaptation of layout must be handled as layout templates are often hierarchical. This issue of layout component handling is addressed in different ways to describe their behavior.

We have already introduced the Mixed Splitter Algorithm [8], i.e., the common application of the horizontal splitter and vertical splitter components during the template definition. The Mixed Splitter Algorithm applies equation sets to describe the

behavior of different layout elements. Based on the equation forms of the adaptation methods, our automatic layout-related results are created. The approach makes it possible to determine the adaptation method of compound layout elements using the domain-specific properties of the area. The algorithm of the approach accounts for the behavior of the contained layout elements. As a result, we can manage the layout calculation of hierarchically defined compound layout templates.

Unfortunately, in certain situations the equation set-based approach is not flexible enough. Therefore, in this paper we investigate the extension possibilities of the approach. We conclude with a solution area-based approach that applies inequalities instead of equations. We represent these solution areas with polygons and prove their flexibility and wide-range applicability.

The fitness-based algorithm of the CTLS approach is an extension of the solution area-based algorithm with the addition of a fitness function.

We summarize the different solutions provided by the CTLS approach:

1. **Equation set-based solution:** Aids in calculating the exact height of a layout element required by a given or selected width value. There often exists more than one acceptable value, which are expressed as precise height values, not intervals.
2. **Solution area-based approach:** This approach addresses inequalities. Based on the inequalities defining the width of the layout element, the algorithm determines the inequalities of the acceptable height values. These inequalities are visually represented by polygons. The drawback to this solution lies in its inability to differentiate between the valid values, i.e., each of the valid values has the same quality.
3. **Fitness-based solution:** This initiation advances the solution area-based approach with the implementation of a fitness function. The fitness function assigns the quality value to the different *x* and *y* pairs (height-width value pairs).

Section 2 discusses the Solution Area-Based Algorithm. We introduce the principles of the method, the solution area-related main concepts and also work out the solution areas of the basic layout elements. Section 3 accommodates the methods of the solution area-based approach by discussing the modifications addressed on the horizontal splitter algorithm. Section 4 introduces the splitter components-related calculation of the resulting adaptation method. Section 5 discusses the fitness function and fitness algorithm of the approach. Finally, section 6 concludes the paper.

2 The Solution Area-Based Algorithm

The solution area-based algorithm determines the height/width definition methods and the attributes of the splitter components from the equation set-based algorithm. The adaptation methods and the related equation forms are provided in Table 1. Comparing to the earlier version of the approach, one of the basic layout elements and their equation sets has been changed. The modification is made in the *Fixed Area (+)* adaptation method (Table 1).

Table 1. The adaptation methods related equation forms

Eq. ID	Adaptation method	Width / Height / Both can be set	Equations
1	*Free (O)*	X / X / X	–
2	*Fixed Width (W)*	- / X / -	$x = c$
3	*Fixed Height (H)*	X / - / -	$y = c$
4	*Fixed (F)*	- / - / -	$x = c1$ $y = c2$
5	*Fixed Ratio (X) (Calc. Ratio)*	X / X / -	$\frac{x}{y} = c$
6	*Fixed Area (+) (Calc. Ratio)*	X / X / -	-

Research was conducted due to several motivating factors following the completion of the equation set-based algorithm. The main points of the motivation are as follows:

1. The equation set-based approach requires the solution in a closed form: *Closed Form Condition* and *Substitution Condition* [8]. This restricts the complexity of the layout structure.
2. Practically, the approach allows the embedding of splitter components into each other with equation sets maintaining a maximum degree of 2. This can be extended, because there are additional closed formulas to solve cubic and quartic functions, but the involvement of these methods into the algorithm does not provide a globally efficient solution. The related implementation necessitates too many branches and special attention is required to treat these particular branches. In the case of equation sets with a maximum degree of 2, we can effectively handle the number of different branches. However, in the case of equation sets with a higher degree, the implementation and testing costs increase exponentially.
3. The approach provides only exact solutions, meaning it can handle equations but cannot handle inequalities. In effect, there is no built-in flexibility, and no extensions can be made to

support it. For example, assume that, for a layout structure with the actual parameters, there is no solution. However, if we were to modify one of the constant values, e.g., by a few percentage points, then it could be solved. This also means that the approach cannot account for the minimum and maximum height and width values of the layout elements.

4. The adaptation method of the layout elements containing a fixed length of text is *Fixed Area (+)*, i.e., the behavior is defined with an equation that requires a fixed area ($x*y=c$). Indeed, if the width of the layout element is given, then the height of the element is calculated based on the text fragmentation (division and handling of different formatting). The resulting function is not continuous, invertible or monotone. This presents significant problems especially in dealing with short texts and long words. The applied approximation is not precise, therefore, the text often does not fit into the calculated area, e.g., the last word requires an entire additional line.

2.1 The Principles of the Solution-Area Based Algorithm

The novel approach does not apply an approximation for the required area of layout elements containing a fixed length of text. Instead, the approach insists on following the precise text fragmentation. Mathematically, this is a stricter and more complex description, but surprisingly, we will see that it will result in a better solution.

Figure 2 introduces both the approximated (dashed line) and the real function of the *x* and *y* value pairs.

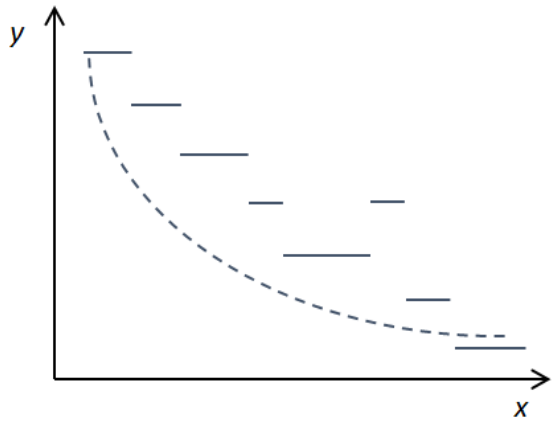


Figure 2. The approximated and the real function of the *x* and *y* value pairs

The discrete values on the *y*-axis result from the text fragmentation. The different lengths of these discrete values are based on the various word lengths and word divisions. If we allow the different text sizes as a text formatting option, then it can result in functions that are not monotonically decreasing, i.e., increasing the width of the layout element results in the height also temporarily increasing.

It should be noted that the result is a function of *x* values but it is not a function of *y* values. In other words, there are solutions only for few *y* values, and regarding these *y* values the result is not a specific x value but instead an interval of *x* values. The consequence is that, in strictly following this solution, the result is not an applicable model. For example, if a template contains one layout element, which has a fixed length of text, then we only get a solution if the height of the screen can be divided by the actual text line height. This type of problem can be uniquely handled, but in a complex layout hierarchy it cannot be propagated between the levels of the layout element hierarchy.

2.2 Area-Based Solution

Considering the above example in a practical sense, we can accept a solution if above or below the text there is a minimal amount of empty space. This empty space can be no higher than the actual text line height. Therefore, for a specific width value, there is no precise height value, which is a discrete value based on the text fragmentation, but we assign an interval of *y* values. The values within this interval can be accepted regarding the actual width value.

This approach uses inequalities instead of equations. Figure 3 introduces the areas that represent the solutions.

Note, that the rectangles are purposefully wider on the right side (*x*-axis) than the lines. This means that the text area can be wider than the ideal width (the narrowest required width). Therefore, in certain cases, the last words of the lines are skipped into the subsequent line, even if there is adequate space for them.

The area-based solution provides the flexibility of the approach. Not only the presented *Fixed Area (+)* function, but other functions can also be extended in a similar way. Furthermore, the flexibility of certain element types can be individually parameterized. Or, if the algorithm does not locate a solution for a given layout, then it can extend the intervals of the accepted values in an

iterative way. Practically speaking, this means larger unfilled spaces between the layout elements.

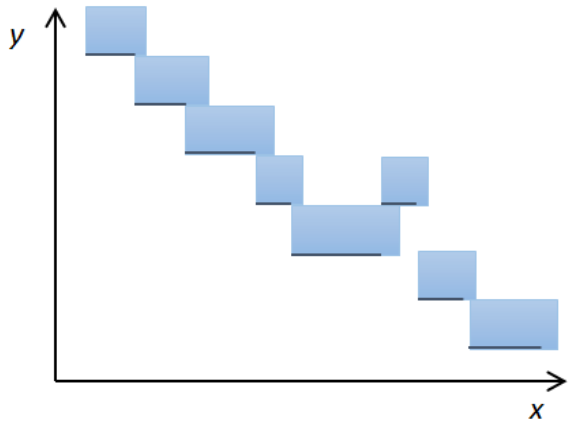


Figure 3. Area-based solution of the *Fixed Area (+)* adaptation method

In our earlier work [7], we introduced a concept which addressed managing the aspect ratio of images. I.e., rendering a layout, while preserving the rules provided by the editor, often requires adapting the image size to the actual conditions. This concept also applies to the area-based solution, because if the ratio of the image can be modified within certain values, then the resulting solutions are not on a single linear function, but an area between two linear functions.

In summary, the extra empty space, which is often a minimal amount, provides flexibility to the approach. It allows for the mapping of a range of height values to a certain width value.

In future work, we will develop the concept to assign fitness values to different values of the solution area. This facilitates in giving preference to certain solutions, e.g., text areas with height values that can be divided with the height of the text line. The fitness values will be uniquely parameterized, which is also a powerful tool for layout editors.

2.3 The Solution Areas of the Basic Layout Elements

Text

Based on the above considerations, each of the basic layout elements has solution areas i.e., inequalities instead of equations. In order to make these inequalities expressive and easier to handle, we use polygons to define them. The x and y values within the polygon represent the solutions of the actual layout elements. The x and y values outside the polygon are the cases in which the actual layout element does not have a solution. This means that we use polygons to define the adaptation method of a layout element.

Table 2 depicts the inequalities and the polygons for the different adaptation methods. The dark blue values represent the original equation-based solutions, and the light blue areas are the extension of it. The light blue areas represent the flexibility of the area-based approach.

We can see in the first four cases, the resulting solutions are represented by rectangles, in the fifth case the solution is the area between two linear functions (it is almost an open rectangle), and in the sixth case the solution is a list of rectangles. These solution areas are similar, but it is important to handle and qualify these cases separately, because their distinctive treatment supports the work of the magazine editors. Furthermore, the validation rules of the different cases facilitate in providing better layout solutions with more possibilities.

Introducing the inequalities provides a method that is closer to the practice and makes more precise modeling possible. The results of the contraction operations are also more precise because they also describe the range of applicable values (co-domain).

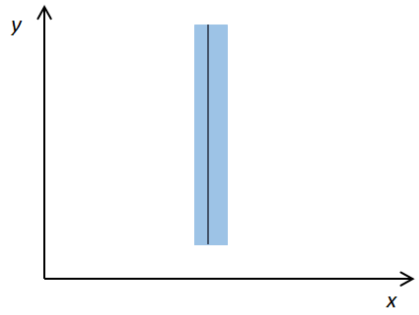
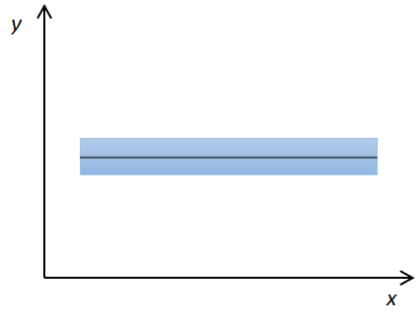
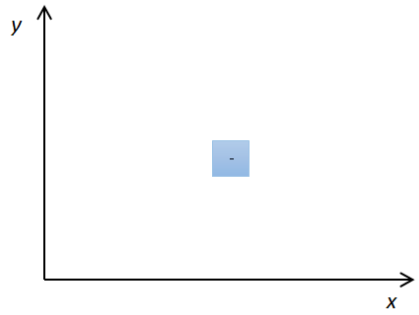
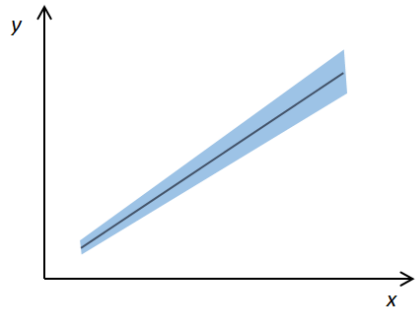
Another advantage to the area-based solution is that introducing additional margins to layout element is no more challenging than it was in the case of equation set-based approach. We only have to move the functions up and right, but their handling method remains the same.

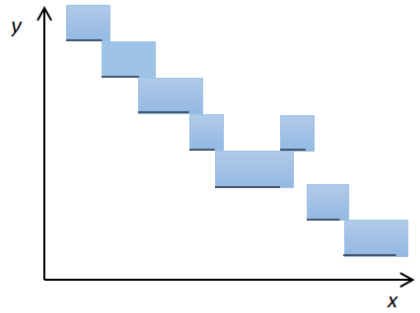
3 Horizontal/Vertical Splitter Algorithm – Based on the Results of the Solution Area Approach

Our layout templates are composed of basic layout elements: texts, images and captions. 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 contains a 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 resulting adaptation method of the horizontal splitter component?

Table 2. The inequalities and solution areas of the adaptation methods

ID	Adaptation method	Inequalities	Solution Areas
1	*Free (O)*	$c1 \leq x \leq c2$ $c3 \leq y \leq c4$	
2	*Fixed Width (W)*	$c1 \leq x \leq c2$ $c3 \leq y \leq c4$	
3	*Fixed Height (H)*	$c1 \leq x \leq c2$ $c3 \leq y \leq c4$	
4	*Fixed (F)*	$c1 \leq x \leq c2$ $c3 \leq y \leq c4$	
5	*Fixed Ratio (X)* *(Calc. Ratio)*	$c1 * x + l1 \geq y \geq c2 * x - l2$ $w1 \leq x \leq w2$	

6	*Fixed Area (+) (Calc. Ratio)*	-	

Layout templates contain cells assembled hierarchically. 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.

In this section, we discuss the results using the Horizontal Splitter Component, which are also true for the Vertical Splitter Component in a similar way.

3.1 The Basics of the Splitter Algorithms

The behavior (adaptation) methods of the basic layout elements (text, image and caption) are provided in Table 1, are the followings:

1. *Free text; Free (O):* containing optional width and height.
2. *One column text; Fixed Width (W):* the width is fixed, but the height is optional.
3. *Header image; Fixed Height (H):* the height is fixed, but the width is optional.
4. *Fixed image; Fixed (F):* both height and width are fixed.
5. *Resizable image; Fixed Ratio (X):* the ratio is constant.
6. *Caption (finite text); Fixed Area (+):* the consumed area is constant.
7. *Calc. Ratio (C):* for a given width, it calculates the appropriate height, and vice versa. This is a generalized version of points 5 and 6.

Besides these considerations, the 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 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*, or the fixed width value is defined, 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*.
- *Auto (A):* The height is determined by the contained element. Given a 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 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 used to calculate the final height of the element. Allowed adaptation methods: *Free* and *Fixed Width*.
- **N*: This method defines the utilization of the remaining space. *N* is a multiplication factor used to calculate the final height of the element. *Allowed* adaptation methods: *Free* and *Fixed Width*.

The size of the elements is not an area of major concentration in our layout approach. In fact, this lack of concentration regarding size is the motivating factor in our layout approach, i.e., we do not inquire about their size or requested size, but instead focus on their adaptation methods. This method defines their behavior during layout calculation. The algorithms of the approach provide answers to the following questions:

- Which cell containment 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 resulting adaptation method (e.g. *Free, Fixed Ratio*, or *Fixed Height*)?
- How are the specific cell heights for a particular device calculated?

It is necessary to examine all possible layout element combinations. In order to minimize the number of combinations, they are normalized, which enables us to handle only those cases that 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 an effort to finish with a reasonable number of cases, we consolidate the similarly behaving height definition methods and adaptation methods.

It is assumed that all horizontal splitter combinations can be composed. The approach has to account for the hierarchically applied horizontal splitter components. This is necessary to understand the resulting adaptation method, because these rules 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 layout element composition's key motivations are the following: A layout template can contain several layout elements, which are often hierarchically defined. In the case of a horizontal splitter component, we aim 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 deep hierarchy.

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 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 signals a cell where optional content type or height definition method can be inserted. Of course, the actual content and method should fulfill the conditions provide by the surrounding elements.
- *!:* Invalid case.
- *../..:* We use a 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*.

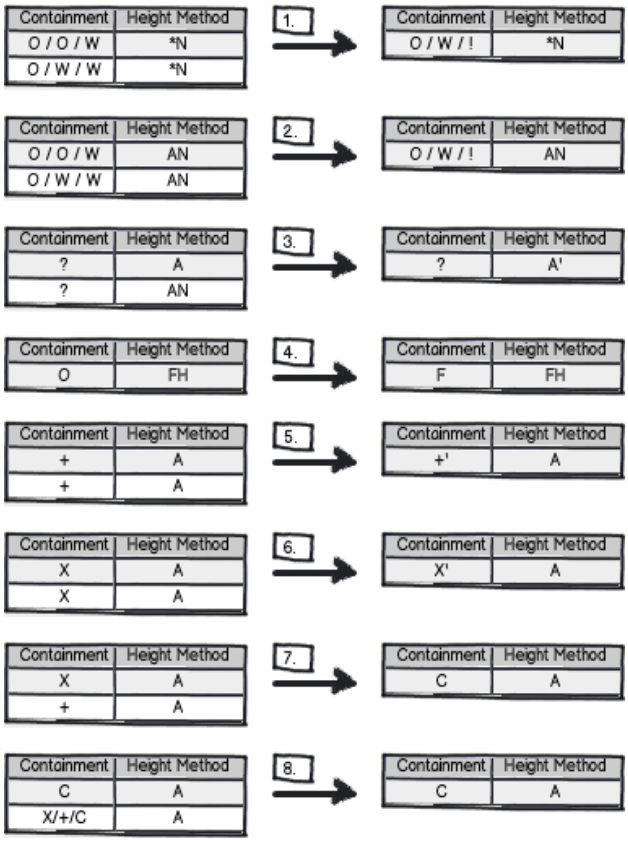


Figure 4. Consolidation of **N* and *AutoN* height definition methods (cases 1-3), consolidation of *FixedH (FH)* height definition method (case 4), consolidation of *Fixed Area (+), Fixed Ratio (X)* and *Calc. Ratio(C)* adaptation methods (cases 5-8)

In Figure 4, the consolidation of **N* and *AutoN* height definition methods are provided by cases 1-3. The consolidation consideration of the *FixedH (FH)* height definition method is presented in case 4. Cases 5-8 depict the consolidations related to the *Fixed Area (+), Fixed Ratio (X)* and *Calc. Ratio(C)* adaptation methods. For example, in the case of the first consolidation we realize the following:

- 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, represents an invalid case.

In case 3, there is a horizontal splitter with two cells, where the first has *Auto (A)* and the second has *AutoN (AN)* height definition method. According to our earlier definitions, the *AutoN* cell can contain only layout elements with optional height (*Free* and *Fixed Width*). This ensures that the layout has a widespread solution. The new, consolidated cell adaptation method is also Auto. The resulting polygon is the stretched polygon from cell *Auto*. For all valid y values the resulting value will be *y*(N+1)*, because the height of the contracted cell is y_{Auto}* (N+1)*. Therefore, the stretching factor of the original polygon is *(N+1)*.

Figure 5 provides the details of the layout element consolidation of case 3 from Figure 4 (above). The horizontal splitter component contains two layout elements. These elements have *Auto* and *AutoN* adaptation methods. In the current case, the value of *N* is 2. The resulting layout element is presented on the right side of Figure 5. This element has an *Auto* adaptation method.

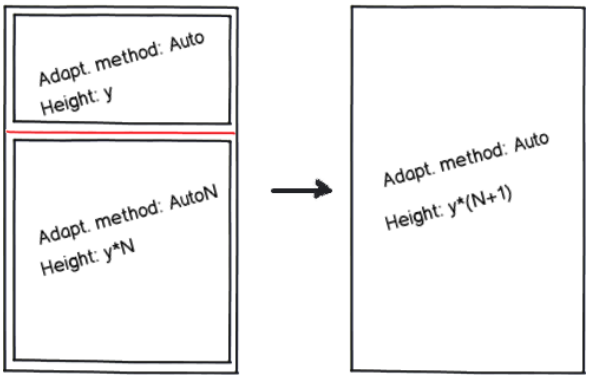


Figure 5. Example consolidation (Figure 4, Case 3)

The next aspects we must consider are the minimum and maximum values assigned to the cell *AutoN*. These values cut the resulting polygon with the appropriate lines. These cutting lines are parallel to the *x* or *y*-axis.

In case 4, the layout element has a definite fixed height. We can apply free elements as well, but due to the *FixedH* height definition method, the resulting adaptation method will be *FixedH*. This consolidation also reduces the number of the possible combinations.

The main point of the consolidation rules are provided in case 8. This is because the *Calc. Ratio (C)* containment is general; this is typically not a basic layout element but the result of another splitter component.

Figure 6 summarizes the results of the layout definition normalization rules. This table shows the possible containment type and height definition method pairs.

Containment	Height Method	Inequalities
X / + / C / F	FHW	4
X / + / C	A	5 / 6 / 7
O / W	*	1 / 2
H	FH	3

Figure 6. The summary of the normalization

We investigated the different content type combinations and determined the resulting adaptation method of the horizontal splitter component, using the normalization rules. If the splitter contains one cell, it corresponds to the adaptation of the actual containment. In Figure 7, 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 resulting inequalities. 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 *. The resulted adaptation method of the splitter component is *Fixed Width (W)*. The tables also provide the numbers of the related inequalities.

In Figure 7, 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 in a contradiction related to the resulting width. Combination 14 is also forbidden, but an identical situation is handled in case 2.

It is sufficient to contract the appropriate equations, when the algorithm has to determine the resulting behavior method of a splitter component. It is unnecessary 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 polygons defined by inequalities.

1. Fixed (F, 4)

Containment	Height Method	Ineq. ID
?	FHW	4
?	A	5-7

2. Forbidden

Containment	Height Method
?	FHW
W	?

3. Fixed Width (W, 2)

Containment	Height Method	Ineq. ID
?	FHW	4
O	*	1

4. Fixed (F, 4)

Containment	Height Method	Ineq. ID
?	FHW	4
H	FH	3

5. Free (O, 1)

Containment	Height Method	Ineq. ID
?	A	5-7
O	*	1

6. Fixed Width (W, 2)

Containment	Height Method	Ineq. ID
?	A	5-7
W	*	2

7. Calc. Ratio (C, ?)

Containment	Height Method	Ineq. ID
?	A	5-7
H	FH	3

8. Fixed Width (W, 2)

Containment	Height Method	Ineq. ID
W	*	2
H	FH	3

9. Free (O, 1)

Containment	Height Method	Ineq. ID
O	*	1
H	FH	3

10. Fixed Width (W, 2)

Containment	Height Method	Ineq. ID
?	FHW	4
?	A	5-7
O	*	1

11. Fixed (F, 4)

Containment	Height Method	Ineq. ID
?	FHW	4
?	A	5-7
H	FH	3

12. Free (O, 1)

Containment	Height Method	Ineq. ID
?	A	5-7
O	*	1
H	FH	3

13. Fixed Width (W, 2)

Containment	Height Method	Ineq. ID
?	A	5-7
W	*	2
H	FH	3

14. Forbidden

Containment	Height Method
?	FHW
?	A
O / W	*
H	FH

15. Fixed Width (W, 2)

Containment	Height Method	Ineq. ID
?	FHW	4
?	A	5-7
O	*	1
H	FH	3

Figure 7. The summary of the normalization

The composition of layout elements is commutative, meaning that the order of the elements in a splitter component is optional. This also relates 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.

4 Calculation of the Resulting Adaptation Method

This section introduces an iterative algorithm for the splitter components related resulting adaptation method calculation.

4.1 The Hierarchy of the Layout Elements

Primitive elements are the basic building blocks of the hierarchical templates. They represent the leaves of the template tree. Splitter components are composed from leaves. Furthermore, splitter components can contain not only primitive elements, but other splitter components as well. The metamodel (a UML class diagram) of the *Template Tree* language is shown in Figure 8. Green nodes represent the primitive layout elements, and blue nodes indicate the two types of splitter components.

The leaf elements on a template tree correspond to the inequalities 1-6 (Table 1), and the further tree nodes are based on the inequalities of a splitter element. The supplemental constraints of the splitter components (e.g., fixed height or fixed width values) are defined with additional inequalities.

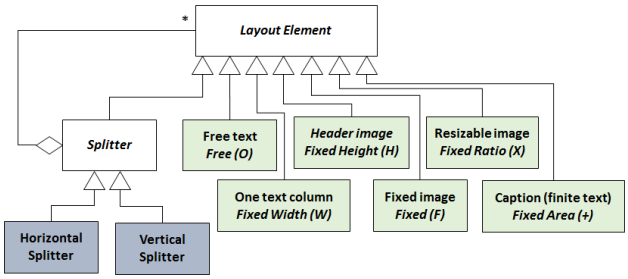


Figure 8. The metamodel of the Template Tree language

Figure 9 depicts an example hierarchical layout definition and the corresponding template tree. The template contains two splitter components (*S1* and *S2*), an image (1) and the splitter *S2* are embedded into splitter *S1*. Splitter *S2* contains a text area (2), an image (3), and a caption (finite text) component (4).

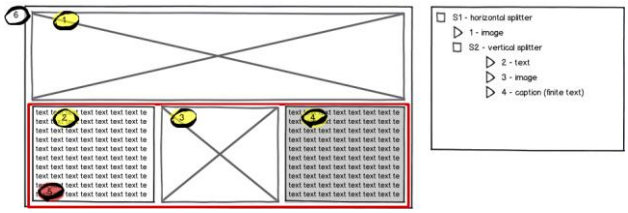


Figure 9. Example hierarchical layout definition and the corresponding template tree

Definition (Template Tree). The template tree is a model built from primitive layout elements (text, image and caption) and splitter components (horizontal splitter and vertical splitter) according to the Template Tree language metamodel.

In regards to the online magazine we have defined the following requirements:

1. The magazine should work on optional screen, i.e. the approach should support optional screen resolution. This means that in vertical direction, the layout should automatically adapt to the device properties.

2. The content can be scrolled in the horizontal direction. Therefore, the calculated layout has no horizontal direction related restrictions.

These requirements are applied to the root template. Therefore, it should be possible to parameterize the root template in the vertical direction, and the template should be either calculated or contain a fixed size in the horizontal direction. If the horizontal direction values are calculated, then they are based on the vertical settings. These considerations constraints the possible adaptation methods of the root template element: *Fixed Width (W), Fixed Ratio (X), Fixed Area (+), Calc. Ratio (C)* or *Free (O)*. In the case of the Free (O) adaptation method, the approach can determine the width of the column template, e.g., it can apply the default column text element width.

Note. If, instead of the horizontal scrolling feature, we wanted to provide a page-based layout, then the root template should have a *Free (O)* adaptation method in order to be applicable on optional screen sizes. In a similar way, we can conclude that the vertical scrolling could be conducted using the following adaptation methods: *Fixed Height (H), Fixed Ratio (X), Fixed Area (+)* or *Calc. Ratio (C)*.

Definition (Basic Layout Element). The adaptation method of a basic layout element is one of the following adaptation methods: *Free (O), Fixed Width (W), Fixed Height (H),* or *Fixed (F).*

These adaptation methods correspond to the inequalities 1-4 in Table 1.

4.2 The Steps of the Adaptation Method Calculation Algorithm

Splitter components allow the placing of layout elements one below another (or one next to another) and calculate the resulting adaptation method. Based on the polygons (defined by the inequalities) related to the behavior of different layout elements, the algorithm calculates the polygon of the splitter element. The resulting polygon determines the adaptation method of the whole component.

Supposing that the algorithm can accomplish this for all splitter elements, the result is a single polygon (inequality set). In order to end with a single polygon (or list of polygons), the algorithm applies the *convolution* operation.

The inputs of the convolution operation are the adaptation methods of the two layout elements. An adaptation method is defined by one or more polygons. The result is also a polygon set, therefore, the convolution operation, taking into account two polygon sets (*Set1* and *Set2*), produces a third polygon set (Set_{Result}).

The convolution operation contracts each of the polygons from Set1 with each of the polygons from Set2. The convolution operation is performed according to the rules of horizontal and vertical contractions. The convolution can be performed based on the inequality sets, however, given the information from the polygons, it is more efficient but also more expressive.

Figure 10 shows an example result of the convolution operation. The contracted layout elements are an image and a fixed length of text. The input polygons are related to a resizable image (*Fixed Ratio (X)*) and a finite text (*Fixed Area (+)*). They are provided in Table 2 (Case 5 and Case 6).

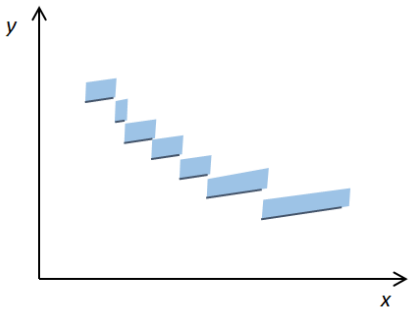


Figure 10. Example result of the convolution operation

We will now introduce the rules of the polygon convolution operation. Let *f(x,y)* and *g(x,y)* be the describing functions of the two polygons (or polygon sets) in the following way. The values of these functions are 1 where the corresponding polygon (or polygon set) has an internal point or the point is on its circumference, and otherwise is 0.

Definition (Horizontal Convolution Operation). The horizontal convolution operation is defined in the following way:

$$h(x0,y0) = \begin{cases} 1, if\ \exists y1\ and\ y2, where\ y0 = y1 + y2\ and \\ \quad f(x0,y1) = 1\ and\ g(x0,y2) = 1 \\ \\ 0\ otherwise \end{cases}$$

The definition of the horizontal convolution operation shows that, in the case of the horizontal splitter component, the result is 0 for those polygon pairs, where the intersection of the polygon projections on the x-axis is empty. For example, in

Figure 11, there are no common points of the two input polygons before *x1* and after *x2*, therefore, in these ranges the polygon projections on the *x*-axis are empty.

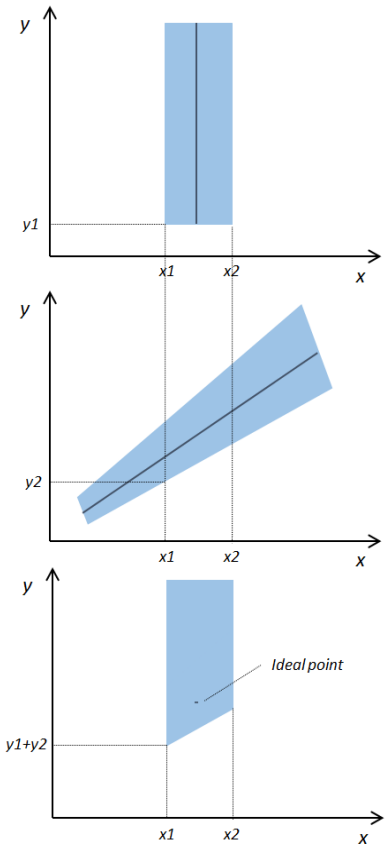


Figure 11. Convolution of Fixed Width (W) and Fixed Ratio (X) layout elements

Figure 11 depicts an example convolution. In this case, the two input layout elements are a one column text (*Fixed Width (W)*) and a resizable image (*Fixed Ratio (X)*). The result is presented in the third frame of reference. The ideal point depicts the intersection of the two original functions.

In our CTLS approach, concerning its basic layout components and splitter elements, our goal is to characterize the different, describing polygons and polygon contour types. These polygons define the adaptation methods of the layout elements. In order to achieve the goals of the layout approach, we define the *Monotone Polygon, Upper Polygon Contour, Bottom Polygon Contour, Monotone Contour, Monotone Increasing (Decreasing) Polygon* and *First Degree Monotone Polygon*. Furthermore, we use these definitions to provide the CTLS approach related propositions. The final goal is to make both the layout algorithms and the related implementation more effective.

Definition (Monotone Polygon). A polygon *P* in the plane is called monotone with respect to a straight line *L*, if every line orthogonal to *L* intersects *P* no more than twice [9].

Proposition. Assume two monotone polygons (*P1* and *P2*) with respect to the x-axis. Contracting polygons *P1* and *P2* with Horizontal Convolution Operation the resulting polygon P_{Result} is also a monotone polygon with respect to the *x*-axis.

Proof. For an optional *x0* let $y2 \geq y1$, width $y1 = \min\{y: f(x0, y) = 1\}$ and $y2 = \max\{y: f(x0, y) = 1\}$.

In a similar way for *g(x,y)*: for an optional *x0* let $Y2 \geq Y1$, width $Y1 = \min\{y: g(x0, y) = 1\}$ and $Y2 = \max\{y: g(x0, y) = 1\}$.

The original polygons are monotone, therefore, every point between *y1* and *y2*, as well as between *Y1* and *Y2*, are parts of the original polygons. The resulting *h(x,y)* function is defined in the following way:

$$h(x0, y0) = \begin{cases} 1, if\ y1 + Y1 \leq y0 \leq\ y2 + Y2 \\ 0\ othewise \end{cases}$$

This definition corresponds to the original definition of *h(x,y)*, because $f(x0, y0) = 1\ if\ y1 \leq y0 \leq\ y2,\ othewise\ 0$, and $g(x0, y0) = 1\ if\ Y1 \leq y0 \leq\ Y2,\ othewise\ 0$. So any $y \in [y1 + Y1, y2 + Y2]$ has a decomposition. The resulting definition of *h(x,y)* function is a monotone polygon with respect to *x*-axis.

A monotone polygon, with respect to the *x*-axis, can be described with two sets of edges. The *y* values of the points belonging to the upper edge set ($ESet_{Upper}$) is always greater or equal to the *y* values of the points belonging to the bottom edge set ($ESet_{Bottom}$) for any given *x0*.

Definition (Upper Polygon Contour). The edges of the set $ESet_{Upper}$ and the related nodes represent the upper polygon contour.

Definition (Bottom Polygon Contour). The edges of the set $ESet_{Bottom}$ and the related nodes represent the bottom polygon contour.

Definition (Monotone Contour). The contour of the polygon is a monotone increasing (decreasing) if for all (x_i, y_i) orderd by *x* values is true that $y_{i+1} \geq y_i$ $(y_{i+1} \leq y_i)$.

Note. The nodes are connected by straight lines, therefore, the monotone behavior is also true for the function defined by these lines.

Definition (Monotone Increasing (Decreasing) Polygon). A polygon that is a monotone polygon

with respect to *x*-axis is a monotone increasing (decreasing) polygon and its upper polygon contour is monotone increasing (decreasing) and its bottom polygon contour is monotone increasing (decreasing).

Proposition. Applying the convolution operation for two monotone increasing (decreasing) polygons produces a similarly monotone increasing (decreasing) polygon.

Proof. The upper and bottom polygon contours are defined by the following steps.

1. Remove those parts of the contours that are not part of the common projections on the *x*-axis. The monotone behavior of the resulting contours and polygons are not affected, because we only removed a part of the contours.
2. Let *f(x)* and *F(x)* define the upper contours of the two polygons, and let *a(x)* and *A(x)* define the bottom contours of the two polygons. Based on these, we can determine the contour of the resulting polygon. It is always true that $a(x) \leq f(x)\ and\ A(x) \leq F(x)$.
   Based on the fact that the two input polygons are monotone, the resulting polygon can be defined in the following way:
   $$h(x0, y0) = \begin{cases} 1, if\ y1 + Y1 \leq y0 \leq\ y2 + Y2 \\ 0\ otherwise \end{cases}$$
   Where $y1 = \min\{y: a(x0, y) = 1\}$ and $y2 = \max\{y: f(x0, y) = 1\}$. Furthermore, $Y1 = \min\{y: A(x0, y) = 1\}$ and $Y2 = \max\{y: F(x0, y) = 1\}$.

   Based on the upper and bottom contours, the resulting polygon can be defined in the following way:
   $$h(x0, y0) = \begin{cases} 1, if\ a(x) + A(x) \leq y0 \leq \\ \quad f(x) + F(x) \\ \\ 0\ otherwise \end{cases}$$
   Based on this, the upper contour of the resulting polygon is the following:
   $$e(x) = f(x) + F(x)$$
   Based on this, the bottom contour of the resulting polygon is the following:
   $$r(x) = a(x) + A(x)$$
   The sum of the similarly monotone increasing (decreasing) functions remains monotone increasing (decreasing), therefore the convolution operation preserves the monotone behavior of the upper and bottom contours. Thus, the polygon remains monotone as well.

Proposition. If the first derivatives of the functions describing the contour of the input polygons (*P1* and *P2*) are monotone increasing (decreasing), then the first derivative of the function defining the contour of the polygon (P_{Result}) resulting from the convolution operation is also monotone increasing (decreasing).

Proof. Based on the proof of the previous proposition and the utilizing of the sum rule in differentiation: *f' + g' = (f + g)'*. If *f'* and *g'* have a certain property, then *f' + g'* also has that property [10].

Definition (First Degree Monotone Polygon). The polygon is monotone in the first degree if: its upper polygon contour is monotone increasing with respect to the *x*-axis, the first derivation of the upper polygon contour is monotone decreasing, the bottom polygon contour is monotone increasing with respect to the *x*-axis and the first derivation of the bottom polygon contour is monotone increasing.

Note. If the upper contour of the polygon is monotone increasing with respect to the *x*-axis, then it is also monotone increasing with respect to the *y*-axis. If the first derivation of the polygon contour with respect to the *y*-axis is monotone increasing, then the first derivation of the polygon contour with respect to the *x*-axis is monotone decreasing.

Proposition. Commuting the *x* and *y* coordinates, the above statements related to the horizontal convolution operation are also true for the vertical convolution operation.

Proof. Based on the similarities of the horizontal and vertical splitter components, and based on the above proofs.

Proposition. The CTLS layout system provides only those layout elements in which the adaptation methods are described with first degree monotone polygons.

Proof. The proof is based on a mathematical induction:

1. The polygons related to the 6 basic layout elements are first degree monotone polygons (Table 2).
2. The basic layout elements are contracted to each other. During this contraction, the convolution operation preserves the first degree monotone behavior of the polygons.

4.3 Propagating the Results Down in the Layout Hierarchy

We can calculate the adaption method of the root layout element by recursively applying the *adaptation method calculation algorithm*. Reaching the root level, every polygon has the valid height-width value pairs. The algorithm picks a height-width value pair and propagates the result down in the layout element hierarchy to the child elements. Each polygon takes into account the actual settings of the higher level polygons. Finally, the available areas are proportionally divided between the layout elements.

5 The Fitness-Based Algorithm

The fitness-based algorithm extends the solution area-based approach with the implementation of a fitness function. The fitness function assigns the quality value to the different *x* and *y* pairs (height-width value pairs).

5.1 The Fitness Function

The solution area-based approach can be regarded as a binary fitness function, which defines whether a given height-width value pair is acceptable for a layout element. This can also be extended to provide a more sophisticated fitness function.

The requirements related to the fitness function:

- Simple and easy to define with few parameters. This assures its ease of use among the magazine editors. In this way, they can easily contemplate the layout issues and deliberate them. E.g., what is the ideal ratio, or what is an acceptable variation from the ideal settings?
- An efficient method should be developed to determine the optimal height-width value pairs (maximum points of the fitness function) for the layout elements.
- Contracting layout elements, embedded into horizontal and/or vertical splitters, facilitate the calculation of the resulting adaptation method. The resulting fitness function should be easily calculated in a similar way. Furthermore, the resulting fitness function should be in a form that can be contracted with another fitness function on the subsequent level of the layout hierarchy.
- The maximum points (optimal height-width value pairs) of the resulting fitness function should also be efficiently calculated.

Our proposition is to apply a linear fitness function, in which planes determine the quality of the height-width value pairs.

5.2 The Steps of the Fitness-Based Algorithm

Splitter components facilitate in the placement of layout elements, one below another (or one next to another), and calculate the resulting adaptation method. Based on the polygons (defined by the inequalities) and fitness functions of different layout elements, the algorithm calculates the polygon and fitness function of the splitter components. The resulting polygon and fitness function determine the adaptation method of the entire component and the quality of the specific height-width value pairs. The algorithm performs this step by contradicting the layout elements. The algorithm always contradicts two elements using the convolution operation. The convolution operation is commutative, making the contradiction order optional.

Assuming the algorithm can conceive the contradiction for all splitter elements, the result is a single polygon set (inequality set) and fitness function (a set of planes).

Definition (Fitness Function). The function, *Fit(x,y)*, for a particular layout element provides the quality value related to an *x,y* value pair (width-height value pair). Invalid height-width value pairs have a function value of 0 within the layout element. These points are positioned outside of the polygon defined by the inequalities of the layout element, i.e., the polygon provides a mask for the fitness function.

The quality value related to a certain height-width value pair tells that how good, how accurate, the layout element with a certain height-width value pair is.

Now let's examine the convolution operation related to the horizontal splitter component.

The fitness function is an equation of a plane

- *f(x):* the function of the upper polygon contour is monotone with respect to the *x*-axis.
- *a(x):* the function of the bottom polygon contour is monotone with respect to the *x*-axis.
- *p0(x0,y0,z0):* the reference point is the bottom-left point of the polygon. Due to the two way monotony, this point is clear. *z0* represents the value of the fitness function for this point.
  - The following formulas assume that this point is related to the result of the contracted polygons, i.e., *x0=X0*. Figure

12 provides an example of this, where the contraction of the polygons is based on their common projections on the *x*-axis.
  - For the bottom-left point: *y0 = a(x0)*.
- The normal vector of the plane related to the fitness function is *n(mh; mv; -1)*.
  - *v(0; 1; mv):* determines the rotation of the plane defining the fitness function around the *x*-axis.
  - *h(1; 0; mh):* determines the rotation of the plane defining the fitness function surrounding the *y*-axis.

Based on the above considerations the fitness function is the following:

$$h(x,y) = \begin{cases} z0 + mh * (x - x0) + mv * (y - y0), \\ if\ a(x) \le y \le f(x) \\ \\ 0\ othewise \end{cases}$$

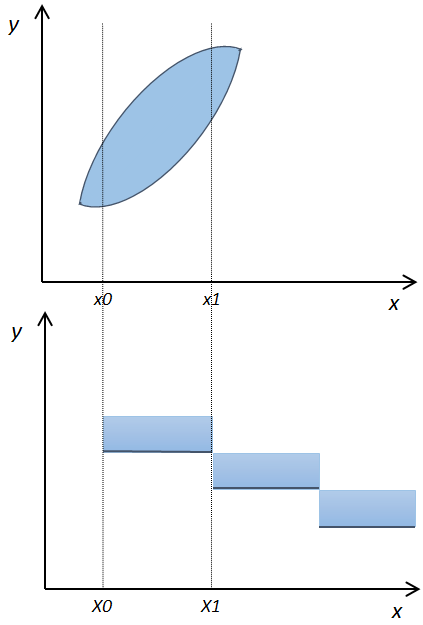


Figure 12. Input polygons of the convolution

During the convolution, the algorithm calculates the function describing the resulted adaptation method of the contracted layout element. Assume that *h(x,y)* and *H(x,y)* are the adaptation methods of the two layout elements. The resulting adaptation method is as follows:

$$e(x,y) = \begin{cases} \arg\max_{y1,y2}\big(h(x,y) + H(x,y)\big), \\ where\ y = y1 + y2\ \ and\ h(x,y1) \neq 0 \\ and\ H(x,y2) \neq 0 \\ \\ 0\ otherwise \end{cases}$$

This means that the volume defined by the (planes of the) fitness function should be maximized.

In the case of planes, based on the principles of linear programming, we can see that, when convolving two polygon-fitness pairs, the result is two polygon-fitness pairs. In the space, the resulting polygons are one stacked on top of each other. They have a common contour, but are described using different fitness functions.

Assuming that *mv < MV*, we define the resulting polygon-fitness pairs.

Definition (Upper Polygon).
- Upper contour: *fe(x) = f(x) + F(x)*
- Bottom contour: *ae(x) = F(x) + a(x)*
- Reference point: *pe0(x0; y0=ae(x0); z0 + H(x0,y0) = z0 + Z0 + Mv * (F(x0) - A(x0)))*
- Normal vector: *n(mh+Mh; mv; 1)*

Definition (Bottom Polygon).
- Upper contour: *Fe(x) = F(x) + a(x)*
- Bottom contour: *Ae(x) = a(x) + A(x)*
- Reference point: *Pe0(X0; Y0 = Ae(X0); Z0 + z0)*
- Normal vector: *n(Mh + mh; Mv; 1)*

Figure 13 illustrates the above definitions.

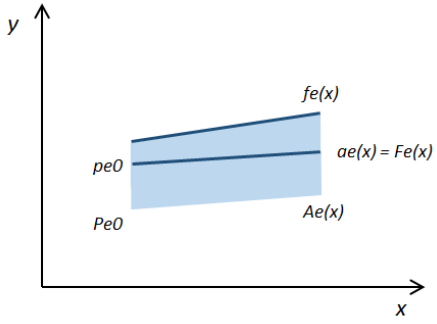


Figure 13. The resulting polygon-fitness pairs of the convolution operation

5.3 Propagating the Results Down in the Layout Hierarchy

We can calculate the adaption method of the root layout element by recursively applying the *fitness-based algorithm*. Reaching the root level, every polygon has the valid height-width value pairs and fitness values, where the higher fitness value means the best solution. The algorithm picks the height-width value pair with the highest fitness value and propagates the result down in the hierarchy to the child elements. At each hierarchy level, the number of the affected polygons equals to the number of the

source polygons (we do not count polygons with no intersect). This differs from the solution-area based approach. On each hierarchy level the solution is related to only one of the polygons. For that polygon the height or width of the element is proportionally calculated and propagated further, while the adjacent element size is maximized or minimized.

6 Conclusion

Mobile device owners are continually consuming digital data, such as online magazines. Therefore mobile devices play a significant role in our lives [11]. Appropriate mobile applications and supporting methods are necessary for these platforms. Such a supporting method is the automatic and adaptive layout calculation [12]. This paper has introduced the Solution Area-Based Algorithm of the Content-Driven Template-Based Layout System.

The Content-Driven Template-Based Layout System approach enables the application of both horizontal and vertical splitter components and the embedding of them into each other. Our algorithms facilitate to deduce the resulting adaptation method on any level of the layout element hierarchy.

We have introduced the principles of the solution area-based approach and worked out the solution areas of the basic layout elements. Furthermore, we have introduced the splitter algorithms, the normalizing layout combinations and we have calculated the resulting adaptation methods.

We have worked out the concept to assign fitness values to different values of the solution area. This facilitates to give preference to certain solutions within the solution area. The approach facilitates to parameterize the fitness value calculation. This is also a powerful tool for layout editors.

Acknowledgements

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

References:

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

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

[3] 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.

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

[5] I. Albert, H. Charaf and L. Lengyel, Layout Definition of Online Magazines with Splitter Components, *International Journal of Engineering Research and Development* 4:(7), pp. 61-69, 2012.

[6] Content-Driven Template Based Layout System (CTLS) web page, https://www.aut.bme.hu/CTLS

[7] 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, 2012.

[8] I. Albert, H. Charaf and L. Lengyel, The Mixed Splitter Algorithm of the Content-Driven Template-Based Layout System, 6th WSEAS International Conference on Visualization, Imaging and Simulation, Lemesos, Cyprus, March 21-23, 2013. Accepted.

[9] F. P. Preparata and M. I. Shamos, *Computational Geometry - An Introduction*, Springer-Verlag, ISBN 0-387-96131-3. 1st edition: ISBN 0-387-96131-3, 1985, 2nd printing, corrected and expanded, 1988: ISBN 3-540-96131-3.

[10] F. W. J. Olver, D. W. Lozier, R. F. Boisvert, C. W. Clark, *NIST Handbook of Mathematical Functions*, Cambridge University Press, 2010, ISBN 978-0-521-19225-5.

[11] Vision Mobile Developer Economics 2013: The tools report, 2013, http://www.visionmobile.com/product/developer-economics-2013-the-tools-report/

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