Apache FOP: whitespace management

I. Introduction and goals

The aim of this project is to develop an extension for Apache FOP that helps managing whitespace in a page, similar to what is described in this article. The user can define a set of alternatives, each one might have different properties and variable content lengths; a layout manager considers each alternative separately and chooses the one that best matches the selection criteria. There are many strategies to choose from, and each strategy has its own merits and may uniquely influence the final document.

II. Selection criteria

The complexity of finding the best alternative is highly dependent on the criteria the user chooses. For my purposes, I have defined three types of strategies a user can choose from:

- **First fit:** as the name implies, the first alternative that fits into the available space is selected, the rest are just ignored.
- **Smallest fit:** the best alternative is the one that occupies the minimum amount of space and can be fit inside the current page.
- **Biggest fit:** the best alternative is the one that occupies the maximum amount of space and can be fit inside the current page.

III. Implementation

Adding an extension for FOP is surprisingly straightforward, even for beginners. First, you have to write a Java *class* that represents your new formatting object. It must inherit from ***FObj*** or any other descendant of ***FONode*** depending on your actual needs. It might be useful to inherit from an existing FO class (***Block***, ***BlockContainer***, ***FObjMixed***, etc…) and reuse some code.

Having done that, you need to tell FOP about the extension otherwise he won't recognize it when parsing the XSL-FO. In ***ExtensionElementMapping.java***, register your XML element by defining a Maker class that implements a way to instantiate your object ,then add a mapping from your XML element's name to the maker you have just defined:

```
public class ExtensionElementMapping extends ElementMapping {

        …

        static class BestFitBlockMaker extends ElementMapping.Maker {
            public FONode make(FONode parent) {
                return new BestFitBlock(parent);
            }
        }

        …
```

```
    protected void initialize() {
        if (foObjs == null) {
            foObjs = new HashMap<String, Maker>();
            …
            foObjs.put("best-fit-block", new BestFitBlockMaker());
        }
    }

}
```

At this point, your extension is recognized by FOP but it doesn't output anything when you produce a document. The reason is that you didn't define a layout manager yet.

A layout manager is an essential component in the transformation process of an XSL-FO file. It formats and renders an FO to an output target (e.g. PDF, PS, SVG, etc…).

The core logic of my extension is trivial, but given the architecture of FOP I must implement it in a way that fits nicely with the main processing engine. Ideally, I don't want it to be some sort of hack, neither to be too much intrusive in FOP's core engine. The best solution I have found so far is to pass the remaining BPD for the current page to each descendent layout managers as part of the layout context information. After I terminate generating a Knuth sequence for my current element, I calculate its occupied size taking into account any alignment properties, then do the necessary checks to see if the formatting object is a valid candidate to be inserted into the current page.

IV. Usage

Here is a simple usage scenario that demonstrates the extension. Please note that the current implementation does not accept more than an alternative. But that will change in future versions.

```
<fox:best-fit-block>
        <fox:alternative-block>
                <fo:block>
                        Sample text
                </fo:block>
        <fox:alternative-block>
</fox:best-fit-block>
```

V. Limitations and Known problems

The way the content's BPD is currently calculated has many shortcomings and does not play well with different writing modes and orientations. For instance, if we apply rotation to an alternative, it becomes challenging to calculate its occupied size, and it will no longer be a matter of summing up the BPDs (block progressing dimension) of its underlying Knuth elements.

In case you didn't notice, the current implementation is only limited to one alternative. I did simplify the problem on purpose because this is my first work on Apache FOP and I'm still familiarizing myself with the code base.

VI. Conclusion

There are many other ways to implement the aforementioned requirements. One particular approach that seems interesting is to use Knuth elements to control how the content is laid out. For example, by inserting a penalty with width equal to the content's length at the beginning of a Knuth sequence, it is possible to know beforehand whether an element fits inside a page or not. However, such an approach requires a good understanding of FOP layout engine and the Knuth algorithm, but it is definitely worth trying.