PageLayout

Page layout management

Returning, collecting and managing the possible pagebreaks

This is a plan to achieve page layout management in FOP. With page layout management I mean that the application is able to apply an algorithm that makes the best choice from a number of layout possibilities of the current page, or even of a few pages. The best choice is determined by the settings in the FO file, such as keep and break property values, by requirements implemented in the algorithm, and possibly by configuration settings of the user, which are taken into account in the algorithm.

I will describe an approach to page layout management where a single point in the application knows the layout possibilities of the current page, or even of a few pages, and chooses one of them as the best page layout.

Page layout management is done by a page layout manager object. It type, say PageLayout`Manager, does not implement Layout`Manager. This page layout manager knows the break poss objects of the current page, along with their BPD optimum, minimum and maximum, their keep and break settings and other traits that are relevant for determining the page break. It applies an algorithm that satisfies the FO file's keeps and breaks. A good algorithm to start with is one that does not choose the first fit, but collects enough information so that it can choose the page break with the minimum amount of stretch or shrink (deviations from the optimum BPD value).

The page layout manager (pageLM) collects the received BPs in a ordered list. When a layout manager (LM) determines a break possibility (BP) in its getNextBreakPoss method, it sends the BP to the pageLM. The page LM adds the BP to its list, and considers the total accumulated BPD. When it decides that it has collected enough BPs to choose a page break, it applies its algorithm and chooses one BP as the page break. This is not necessarily the last reported BP. It then sends a 'page completed' signal to the reporting LM, in its return value of the method in which the LM reported the BP. The page breaking BP (pageBP) is advertised such that all LMs can see it. (All LMs have a reference to the pageLM.) The pageLM also advertises the stretch or shrink factor of the page, to be applied to all BPD dimensions.

The active LM then returns from its getNextBreakPoss with a 'page completed' signal, which is received by its parent LM, which called this method. The parent LM likewise returns from its getNextBreakPoss with a 'page completed' signal to its parent LM, etc. up to the PageSequenceLM which started the getNextBreakPoss call stack.

The PageSequenceLM then starts the addAreas call stack. Each LM compares each BP with the pageBP. If it is not equal to it, it adds an area and continues with the next BP. If the BP is equal to the pageBP, it also adds an area, but returns to its parent LM with a 'page completed' signal. The parent LM then completes its area and returns likewise with a 'page completed' signal to its parent LM, until the PageSequenceLM is reached again, which completes the page and hands it over to the Area{{`Tree}}`Handler.

If an LM has determined and reported all its BPs without receipt of a 'page completed' signal, it returns from its getNextBreakPoss with a 'finished' signal. The parent LM then knows that it should continue with the next sibling, or, when all children are done, that it should itself return with a 'finished' signal. Likewise, if an LM has added all its areas without receipt of a 'page completed' signal, it returns from its addAreas with a 'finished' signal. This is dealt with in the same manner as for the getNextBreakPoss method.

Each LM should mark the BPs for which it has added an area, or perhaps even remove them, so that it knows which BPs are left over for the next page. An LM is finished adding areas, when it has no unprocessed BPs left, and is finished getting BPs. Only LMs whose area ends at the page break, can be left in an unfinished state.

Compared to the current situation, the LMs do not keep a list of BPs of their children. Only the LMs that contribute leaf block areas (e.g. LineLM) maintain a list of BPs. The addAreas call stack goes depth first, by iterating over its ChildLMs and call addAreas for each childLM. The leaf LMs would contribute areas until the BP is equal to the pageBP. Then the parent creates its own area, containing the returned child areas. This will do away with the complicated BP tree and its iterator. Note that the inline layout still uses it.

It may be useful to have a method PageLM.isOnCurrentPage(BP), which would request the pageLM to compare the BP with the pageBP and inform the caller if the BP is part of the current page. It can do so due to its ordered list of BPs.

Luca Furini http://nagoya.apache.org/eyebrowse/ReadMsg?listName=fop-dev@xml.apache.org&msgNo=10549 pointed to the situation where an FONode has keep-together, which influences the BPs of its whole subtree. This presents some difficulty for the above strategy. The property must be propagated down the subtree of LMs, which must signal it in their BPs so that the PageLM can take it into account.

Finn Bock http://nagoya.apache.org/eyebrowse/ReadMsg?listName=fop-dev@xml.apache.org&msgNo=10550 favours returning the BPs up the stack of LMs up to the LM that does the page breaking, analogous to the LineLM in paragraph layout. It makes it easier to handle the above case. But it is rather expensive in processing, also because each LM creates a new BP which wraps the received BP and which is returned to its parentLM.

Page breaking was discussed in this email thread:
http://nagoya.apache.org/eyebrowse/BrowseList?listName=fop-dev@xml.apache.org&by=thread&from=984205. The above two messages are part of it.

Current situation

LineLM.getNextBreakPoss returns a BP for each line. BlockLM.getNextBreakPoss returns a BP if:

- it overflows (pagebreak),
- it is finished.

Each LM has a list of BPs of its child LMs, one BP for each area that the child LM contributes.

An LM receives from its parent a BP for the next area. For each child BP between the previous BP and this BP, it asks the child LM of that child BP to return the areas up to that child BP. A BP has a direct correspondence to a child BP through its member leafPos, which is an index into the list of child BPs.

Page breaking strategies

The current BP system uses a first-fit strategy. As soon as the page height is exceeded, the page break is lain at the preceding BP. This strategy is simple, but does not provide choice. Therefore it is hard, although not impossible to take keeps into account.

The next simple strategy is a best-fit strategy. One collects at least one BP that no longer fits on the page. There may be several possible page breaks due to stretchability/shrinkability on the page. Then one weighs the possible page breaking points with a suitable merits and demerits system, and selects the best page break. This method requires a better overview over the available break points and stretchability/shrinkability on the page. This strategy is used by TeX in its vertical list.

The next possible strategy uses look ahead [1]. There is a sliding window of *N* pages. The best page breaks are calculated over all pages in the window, but only the page break of the first page is used. Then the window is moved forward one page, and the effort is repeated. This strategy may result in a better placement of floating elements.

In a variant of the best-fit strategy and of the look-ahead strategy, one may use the page breaks of *M* pages. This is useful for balancing facing pages (*M*=2, even page/odd page).

A total-fit strategy, as used in paragraph breaking, can only be applied to a complete page sequence. This will almost always to expensive, both in computing effort and in memory requirements.

---

[1] JM: One problem I see with a look-ahead approach is the different available IPD value on subsequent pages which would cause too much recalculation of the lines (and table layouts if auto-layout is active). In line-breaking we only have constant boxes while in page-breaking the boxes have different dimensions due to other line breaking decisions being made. Too much look ahead can cause too much wasted effort in complex documents. (See normal-breaking4.xml)

[2] JM: I've added a page that tries to list all influencing elements which should help us determine the right algorithm to implement: PageLayout InfluencingFeatures

Layout around a page break

The layout of a part of the page depends on whether the part is layed out at a page break or not. For example, the resolution of space specifiers is different when they occur at the edge of a reference area or not. When a page break occurs in a table, a footer is added. The borders of the footer interact with those of the last rows and cells.

A BP should always represent the situation which would occur if it were the selected page break. The following BP makes the calculation for its predecessor partly undone, because it represents itself as the selected page break.

When a page break has been selected, the calculations for the BP after the selected page break must be revised. They must now take elements at the start of a new page into account. The resolution of the space specifiers before the first block is different because it now occurs at the before edge of a reference area. When a page break occurs in a table, a header is added. The borders of the header interact with those of the first rows and cells.

The paragraph layout mechanism of Knuth has elements, penalties, which are taken into account when they occur at a line break, but are ignored when they do not occur at a line break. See especially the use of a penalty to represent a possible hyphenation point. Other elements, glue items, are ignored when they occur at a line break, but are taken into account when they do not occur at a line break.

It may be useful to do the same for page breaks (see Finn Bock's ideas). But the situation both at the end and at the start of a page is more complicated. At the end elements are not only removed, they may also be added. Similarly at the start of a page.

Expressing layout around a pagebreak as Knuth elements

We consider the following layout situations.

Space specifiers

When the space specifiers resolve to zero around a page break, we are in the same situation as that of a word space in line breaking. It is represented by the sequence `box - glue - box`.

When the space specifiers do not resolve to zero around a page break, we are in the same situation as that of a word space in line breaking in the case of centered lines. It is represented by the sequence

```
box - infinite penalty - glue(ha) - zero penalty - glue(hn-ha-hb) - zero width box - infinite penalty - glue
(hb) - box
```

where ha is the bpd of the space-after before the page break, hb is the bpd of the space-before after the page-break, hw is the space when there is no page break.

Possible page break between content elements

Here the most general situation is that when the content is different with and without page break:

- content Cn when there is no page break,

- content Ca at the end of the page before the page break,
- content Cb at the start of the page after the page break.

An example of this situation is a page break between table rows:

```
no page break:    page break:

---------         ---------
  row 1             row 1
---------         ---------
 border n          border a
---------         ---------
  row 2             footer
---------         ---------
                  page break
                  ---------
                   header
                  ---------
                   border b
                  ---------
                    row 2
                  ---------
```

This situation cannot be dealt with using Knuth's box/glue/penalty model. We introduce two new type of elements, which are a kind of penalties:

1. A penalty with height hn, which is the height of Cn, and with height ha, which is the height of Ca.
2. An infinite penalty with height hn', which is the height of Cn', and with height hb, which is the height of Cb, and with a flag value of start-page. Here Cn' is the part of Cn which is not represented in the penalty of type 1; normally it is empty.

Both penalty types differ from a normal penalty in that they also insert a height when there is no page break. For type 2 that is a rather theoretical possibility. Penalty type 2 differs from a normal penalty in that it is not discarded when it occurs at the start of a new page. Note that in that case it is not the page break itself; the page break is at a preceding glue or penalty from which it is separated by glue or normal penalty items, which are discarded at the start of a page. Both types share some features of a box and of a penalty; they are a kind of conditional box.

The above table rows are then represented as:

```
box(row 1) - penalty(hn, ha, 0, 1) - penalty(0, hb, inf, start-page) - box(row 2)
```

Without a page break this becomes:

```
box(row 1) - hn (border n) - box(row 2)
```

With a page break this becomes:

```
box(row 1) - ha (border a + footer)
hb (header + border b) - box(row 2)
```

Here ha and hb include the border-after of the footer and the border-before of the header.

Finn Bock launched the idea to use Knuth semantics to express page breaking.

This plan was written by SimonPepping