ApacheCon Europe 05

FOP

Apache FOP: Optimizing speed and memory consumption

Jeremias Märki <jeremias@apache.org>
2005-07-21, DE1307

Topics

- Short Info Block
  - Project Status
  - The Future
- Tips & Tricks for optimizing FOP
  - Inside FOP
  - Outside FOP
- Q & A

The Big Question

?

?

IS FOP DEAD?

?

Last release: July 2003 (version 0.20.5)

?

Redesign - Why?

- The original design wasn't up to the task
  - Trouble with big documents
- Dead end for certain layout features
  - Keeps on all FOs
  - Better table layout
  - Border painting
  - XSL-FO compliance in general

Why does it take so long?

- Resources short
- Team almost completely replaced
- XSL-FO layout is very complex
- Disagreements on course of action

Current status

Latest news, directly from the project...

Current Project Status

- Completely rewritten layout engine
  - TeX-like approach (Knuth element model)
  - Easier maintenance
  - Fewer side-effects
  - Room for improvements
- Improved FO Tree (Memory, Speed)
- Layout Engine Test Facility

Status (cont.)

- Better validation and conformance
- We have basic keeps on all FOs
- reference-orientation
- Indents and margins behave correctly
  - Fewer nested-table work-arounds!
- Nicer text layout due to TeX-like line and page breaking

The near Future

- Bringing PDF, PS and Java2D renderers up-to-date
- Automated visual testing
- Testing with real-life documents
- Preparations for a first preview release within about 2 months!!!

Missing features

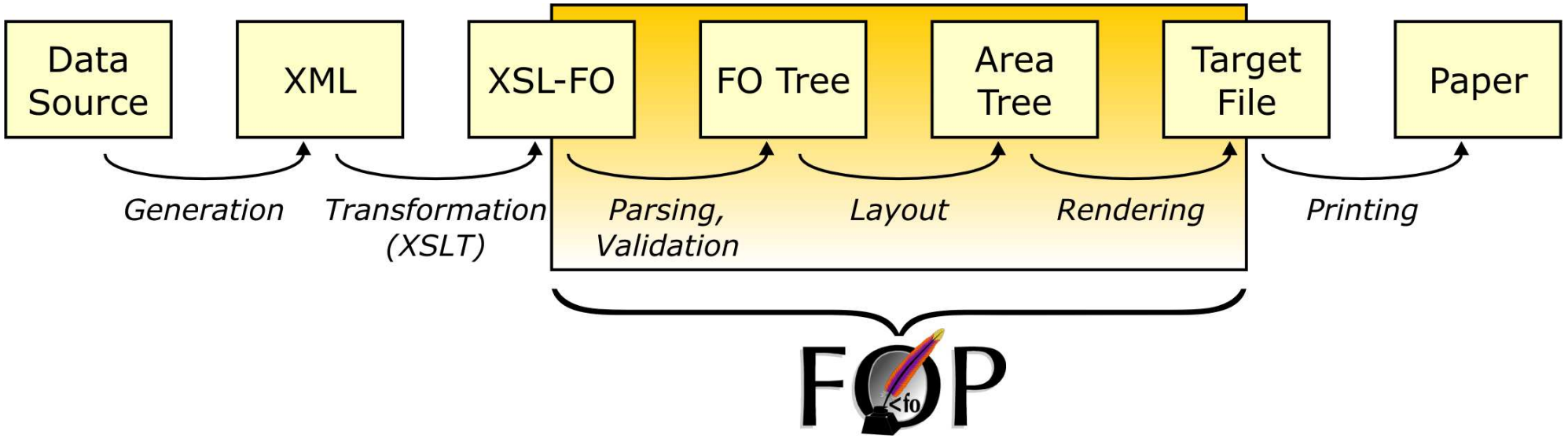
- Proper reflow over pages with different widths.
- Tables: Collapsing border model and auto-table layout
- Not all former renderers available, yet (only PDF, PS and Java2D/AWT)
- Non-western writing modes, floats
- ...and many other little things

Any questions so far?

How FOP works

- Two approaches for conversions
  - Page-oriented formats (PDF, PS, PCL...)
  - Flow-oriented formats (RTF, OpenOffice...)
- Layout-Engine used only for page-oriented formats

Formatting Process



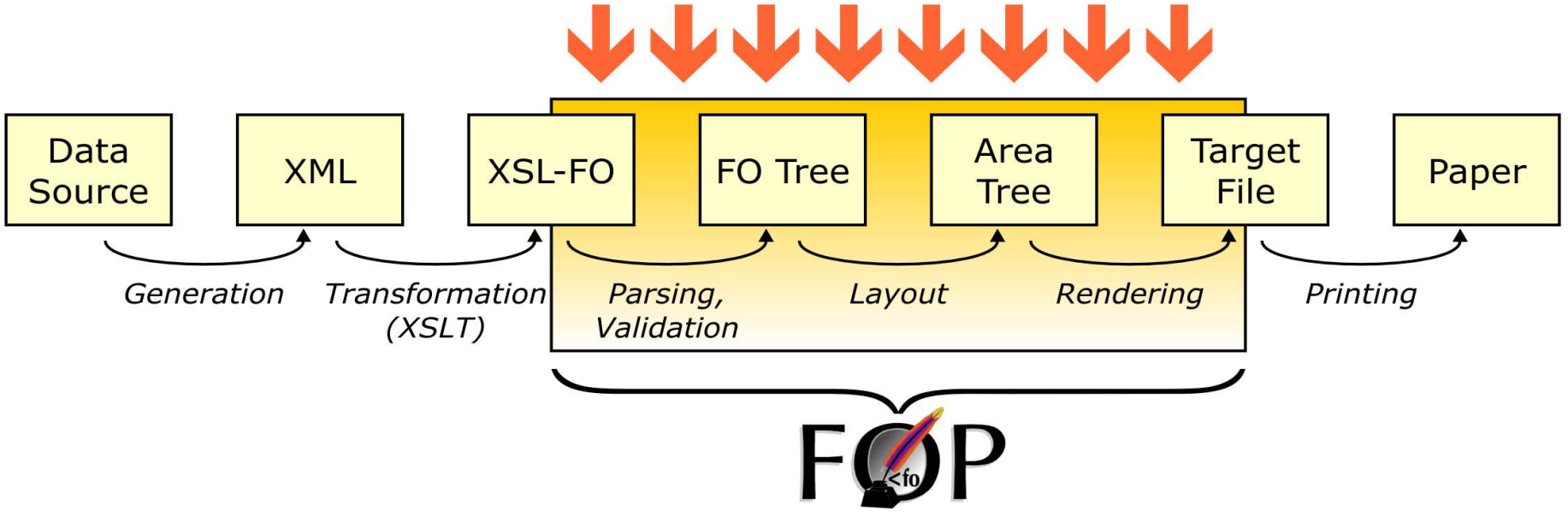
Types of documents

- Book-style documents
  - Books, Manuals
  - Lots of references, long text passages
  - Possibly large images
- Business-style documents
  - Invoices, Forms, Insurance Policies...
  - Few references, lots of small texts
  - Many smaller images, many tables

Starting with Adam & Eve

- Increase JVM memory
- Update your JRE
- Don't use the command-line for repeated calls
  - Alternatives: Servlets, Web Services, etc.

Part 1: Inside FOP



Direct optimization potential

References

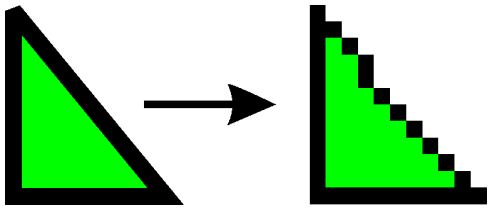
- Backward references are ok
- Caution with forward references
- “page x of y” is a forward reference
- FOP 0.20.5 can't do out-of-line rendering (FOP 1.0 will support it)

Breaks vs. Page Sequences

- Page sequences allow FOP to release memory.
- Breaks don't.
- Use as many page sequences as possible.
- Business docs: 1 page-sequence per subdocument.

Images

- Choose the resolution of bitmaps wisely
- Use JPEG if possible (no decoding in PDF and PS Level 3)
- Image cache may need a reset
- Bitmaps instead of SVG may make sense.



Creative Optimization

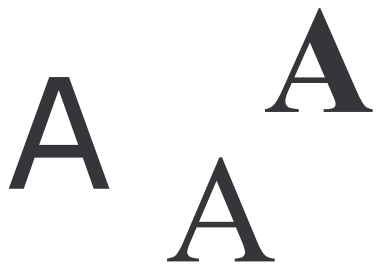
- Skip images or even whole parts
  - Post-process PostScript to add logos as forms
  - PPML (just an idea for the future)
- PDF can reuse images
- PostScript cannot (yet)
- EPS with PostScript

Image performance

Repeats: 10	PDF	PS
SVG	9.5sec	9.8sec
PNG 300dpi	5.4sec	4.1sec
PNG 72dpi	3.5sec	3.4sec
JPEG 300dpi	2.7sec	2.7sec
JPEG 72dpi	2.6sec	2.6sec
EPS	(5.4sec)	3.1sec

Fonts

- TrueType fonts can be big and are more complex to parse.
- If possible use Type 1 fonts.
- Avoid embedding (difficult for TTF)
- Make fonts resident on printer



Font performance

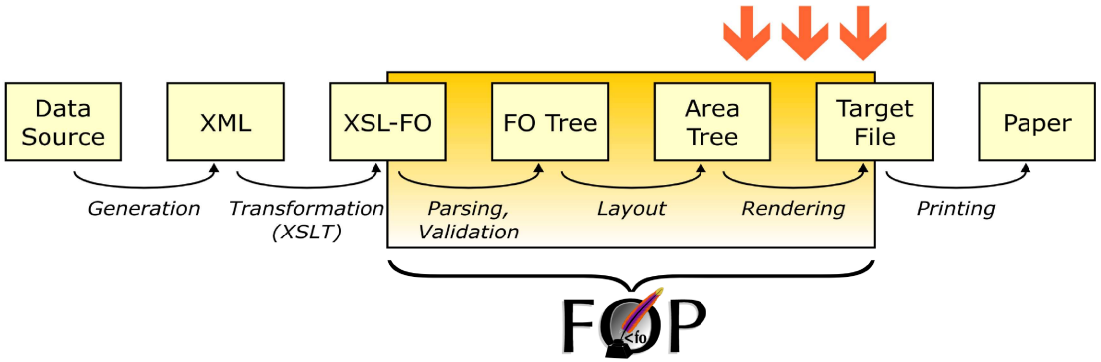
Repeats: 1	Time
TrueType embedded	3.78sec
Type1 embedded	1.65sec
Type1 referenced	1.40sec

Tables are tough

- CPU- and memory-intensive
- Try to split long tables
- blind tables in 0.20.5
  - as work-arounds for buggy spacing and keeps

Renderers

- Choose the right renderer for the job
- Remember to buffer the OutputStream
- PostScript is 50% slower if not buffered
- Native Renderers are faster than Java2D/AWT Renderer



Additional Ideas

- Distribute processing on multiple CPUs/machines (Blade servers!)
- Page Sequences as obvious split points

Part 2: Outside FOP

Don't underestimate this part!

Simplify XSL-FO

- Avoid clutter!
- Stylesheets are programs, too, so design carefully!
- Modularize to get a better overview
- Use property inheritance
- Fewer attributes means less memory consumption and less to process

Using XSL-FO editors

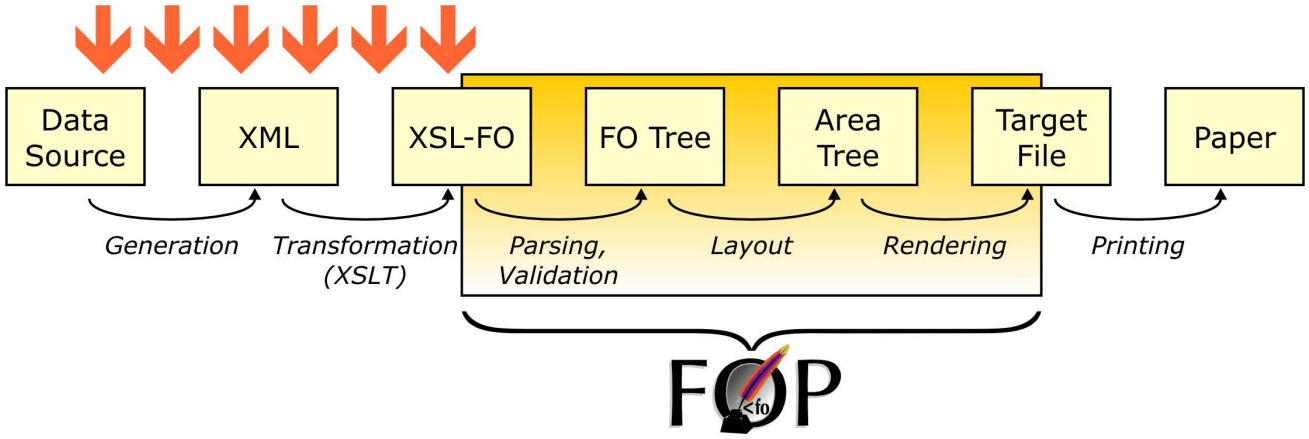
- Editors can produce huge FO files
- This can be inefficient at runtime
- They tend not to make full use of inheritance
- Post-processing the XSLT may be possible (using XSLT)

Other pitfalls

- Strip out DTD references
- Or use a resolver to access local versions of DTDs
- Or switch off validation entirely

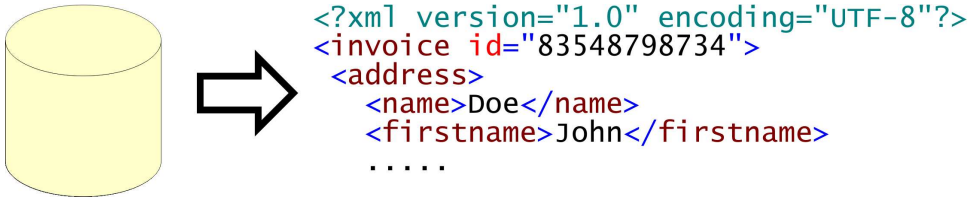
Optimizing Input

- Often, data generation and transformation is already inefficient
- Create an efficient transformation chain
- Avoid serialization/deserialization
- Use SAX instead of DOM



Data/XML generation

- People often create DOMs from Beans
- Create SAX events instead – it's easy
- Examples on FOP's embedding page
  - *http://xml.apache.org/fop/embedding.html*
- Or use Jakarta Commons Betwixt's SAXBeanWriter for Beans



On the wrong track?

Checklist:

- Do I build Strings?
- Do I create DOMs?
- Do I write temp files?
- Do I use a ByteArrayOutputStream?

Don't get me wrong!
It's not always a bad sign!

SAX pipelines

- Stay in SAX pipelines whenever possible
- Apache Cocoon is the best example
- Learn the basic JAXP usage patterns involving Transformer(Handler)
- Examples on FOP website

XSLT

- Master JAXP! (javax.xml.transform)
- Reuse “Templates” instances! (Cache)
- Use the latest Xalan-J, not the JRE's!
- Try another XSLT processor
  - SAXON may be faster than Xalan-J
- Use compiled stylesheets (Example: Xalan's XSLTC)

Improve your XSLT skills

- You can do one thing in different ways
- It's easy to do inefficient things
- Sort in DB queries not in XSLT
- etc. etc.
- Get a good XSLT book!

Alternatives to XSLT?

- Creating XSL-FO directly is not ideal
- Template engines like Velocity or FreeMarker? Maybe...

How to identify problems

- Split the transformation pipeline
  - FOP is only the last piece in the puzzle
- Run each step manually
- Switch off parts of the XSLT to narrow down the search
- Use “-d” or a higher logging level

Conclusions

- Lots of variables in the equation
- Not just FOP defines the performance
- Measure yourself, measure early
- The FOP team can do better still. ☺
  We're working on it! And you can help!

Most Important Tips

- Know the difficult FO constructs
- Get an overview over the whole process
- Create an efficient data flow
- Learn to isolate the different steps

Stuck? Need help?

Contact us by subscribing to fop-users@xmlgraphics.apache.org

Questions?

ApacheCon Europe 05

FOP

Thank you!!!

Feedback? Comments? Suggestions?

Help wanted in the XML Graphics project! ☺