



## Improving quality-critical XML workflows with XProc 3.0 pipelines

Achim Berndzen, `<xml-project />`

Thorsten Rohm, Thieme Compliance GmbH

Orchestrating complex XML pipelines has been a major topic of XML-related software development over the years. Comprehensive techniques have been developed to:

1. Deliver high-quality results
2. Ensure that the pipelines can be maintained
3. Allow the pipelines to be debugged for straightforward troubleshooting

The quality demands for the workflow and the produced results can vary: For example, you may find a very maintainable pipeline producing documents with very low quality demands, e.g. the system producing the static website for your local sports club. On the other hand you might find documents with very high quality demands produced by a pipeline that is not easy to maintain and debug. And, of course, the relationship between the quality of the documents and the maintainability of the pipeline producing these documents may change over time. Implementing new quality demands for the documents might have a negative impact on the pipelines quality. And sometimes in the history of developing a pipeline expected to produce documents with high quality demands, you might even decide to start over, as new quality demands for the documents threaten to impair the quality of your pipeline.

In this paper, we would like to report about a shared project of our two companies. We had to add new features to a well-established workflow producing documents in the medical sector that come with very high quality demands. As the existing workflow already had some pain points, we decided to start over and to refactor it. And we even decided to change the basic orchestrating technology: Since the existing workflow was based on a combination of Windows batch files calling different programs and some very elaborate XSLT stylesheets, we decided to use XProc 3.0 to orchestrate the workflow, thus doing away with as much shell scripting as possible while keeping the XSLT stylesheets to do the actual transformations.

As XProc 3.0 is a relatively new technology for orchestrating document workflows, we think our project might be of some interest to people developing and/or maintaining pipelines for documents with high quality demands. We will first provide some background context for the produced documents and their actual usage to elaborate the specific quality demands. This will be followed by an overview of the existing workflow and a discussion on its pain points and new demands. We will then give an overview of the new XProc 3.0 pipeline developed in the project and discuss some aspects of the used technology. The paper<sup>1</sup> concludes with the lessons learned in our project

---

<sup>1</sup>We would like to thank the reviewers of our abstract for their very helpful comments. A special thank goes out to Geert Bormans whose thoughtful remarks on the abstract helped to improved this paper significantly.

and the key takeaways of our project in a more general context of pipelines producing documents with high quality demands.

## 1. Introduction and background

### 1.1. About Thieme Compliance GmbH and patient education leaflets

Thieme Compliance GmbH, based in Erlangen, Germany, is a company that specialises in providing patient education solutions for healthcare facilities. These solutions include, among other things, information materials that educate patients about their illnesses, treatment options and possible risks. Patient education is an important aspect of healthcare as it helps patients make informed decisions about their health. It is also an important legal and ethical principle in medicine.

Thieme Compliance GmbH supports healthcare facilities in implementing this principle by providing customised information materials tailored to the specific needs of patients. The materials are developed in close cooperation with more than 400 experts from the medical community and tested for their comprehensibility and usefulness. Furthermore, a team of legal advisors ensures that the patient education content always corresponds to current case law. Professional societies and associations recommend the patient education leaflets from Thieme Compliance. In total, more than 2,000 patient education leaflets from more than 30 speciality areas are available in up to 31 languages. They are available in digital form as well as various print formats.

Figure 1. Patient education leaflet



The aim is to help ensure that patients are better informed and educated so that they can make decisions about their health in close cooperation with their doctors. Clinics and practices are supported in meeting legal requirements and minimising liability risks by using patient education leaflets. Under certain conditions, the clinics and practices can even receive a reduced insurance premium if they use the patient education leaflets from Thieme Compliance GmbH, as the risk of being sued by a patient is reduced.

The educational content is made available to clinics and practices via the self-developed software E-ConsentPro. E-ConsentPro is installed on-premise in clinics and practices and offers interfaces to, or is even embedded in, clinical information systems. It ships with Saxon-EE, Antenna House Formatter, XSLT as well as XSL-FO stylesheets, fonts, etc. and can thus be used to generate various media forms of the patient education leaflets, supplemented and personalised with data from the clinical information system.

**Figure 2.** “Anamnese mobil” app from E-ConsentPro



For parts of the patient education leaflets, such as the medical history, HL7's FHIR (Fast Healthcare Interoperability Resources) questionnaire resource is used for syntactic healthcare interoperability. The HL7 FHIR standard is based on a RESTful API architecture. This is an emerging standard for exchanging medical data between different systems and institutions and will replace the established HL7 V2 standard in the future. To ensure semantic healthcare interoperability as well, the questionnaire resource contains coding from SNOMED CT (Systematized Nomenclature of Medicine and Clinical Terms) or LOINC (Logical Observation Identifiers Names and Codes):

```
<fhir:item>
  <fhir:linkId
value="MF_Erkrankungen_Familie__Erkrankung_Blutsverwandschaft" />
  <fhir:text value="Among your blood relatives, are or were there
any diseases or indications of a disease?" />
  <fhir:type value="open-choice" />
  <fhir:required value="true" />
  <fhir:repeats value="true" />
  <fhir:answerOption
id="MF_Erkrankungen_Familie__Erkrankung_Blutsverwandschaft__nein">
    <fhir:extension url="http://hl7.org/fhir/StructureDefinition/
questionnaire-optionExclusive">
      <fhir:valueBoolean value="true" />
    </fhir:extension>
  </fhir:answerOption>
</fhir:item>
```

```

        <fhir:system value="http://snomed.info/sct" />
        <fhir:version value="http://snomed.info/sct/
9000000000000207008/version/20220430" />
        <fhir:code value="160266009" />
        <fhir:display value="No family history of clinical finding
(situation)" />
    </fhir:valueCoding>
</fhir:answerOption>
<fhir:answerOption
id="MF_Erkrankungen_Familie__Erkrankung_Blutsverwandtschaft__Krebs">
    <fhir:valueCoding>
        <fhir:system value="http://snomed.info/sct" />
        <fhir:version value="http://snomed.info/sct/
9000000000000207008/version/20210131" />
        <fhir:code value="275937001" />
        <fhir:display value="Family history of cancer
(situation)" />
    </fhir:valueCoding>
</fhir:answerOption>
<!-- ... -->
<fhir:answerOption
id="MF_Erkrankungen_Familie__Erkrankung_Blutsverwandtschaft__Erbkrank
heiten">
    <fhir:valueCoding>
        <fhir:system value="http://snomed.info/sct" />
        <fhir:version value="http://snomed.info/sct/
9000000000000207008/version/20220430" />
        <fhir:code value="429962007" />
        <fhir:display value="Family history of hereditary disease
(situation)" />
    </fhir:valueCoding>
</fhir:answerOption>
</fhir:item>

```

The patient education content is created and managed using the component content management system Content Lifecycle System (CLS) from Empolis Solutions GmbH. It consists of XML files with single-source-capability, modularised using XInclude. Because of the rich semantics, a self-developed data structure T<sub>0</sub> XSD is used.

The patient education leaflets are published in various media formats such as PDF, XHTML, HTML5, WordML, SpreadsheetML, among others. To facilitate the data exchange, especially with partners, the content is converted into a variety of XML dialects as well as JSON formats. Therefore, fully or at least highly automated publishing pipelines are being developed and maintained.

## 1.2. About <xml-project /> and XProc

XProc 3.0 is a pipeline language with an XML syntax. It is based on XProc (1.0), which became a W3C recommendation in 2010. Based on user experience, a group of volunteers have worked together since 2017 as an W3C community group to improve and expand the original language. In September 2022, a community report on the core language specifications and the standard step library was published. While additional step libraries, e.g. for file processing, document validation, and paged media creation, are technically still under construction, we consider them to be very mature and in their final state. In fact, the project presented here relies heavily steps from the additional libraries, and they proved to be very useful and robust.

For those familiar with the original XProc, it might be interesting to mention some of the changes made for XProc 3.0. The most visible change is the expansion of the basic document model from XML only to a more realistic model for the latest processing: “Native” documents in XProc 3.0 are now XML, HTML, JSON, as well as text documents and binary documents (such as images or PDFs). The newly supported document types are accompanied by corresponding steps so that they can be used effectively in the pipelines. Further highlights of XProc 3.0 are the move to XPath 3.1 as the underlying processing language, XDM typing for options and variables along with a number of minor syntax tweaks that greatly improve the coding and debugging experience from the original XProc.

For those not familiar with XProc 1.0, or those who want to start over with XProc 3.0, there is now an improved learning base. Foremost, there is Erik Siegel's excellent book [Siegel:2020]. Erik also published a series of articles introducing XProc 3.0 on XML.com ([Siegel:2019], [Siegel:2020a], [Siegel:2020b]). For those who prefer videos, a series of six talks from Markup UK 2020 are available on the conference's YouTube channel. In addition to two talks on the basics of XProc 3.0, there are also talks on handling JSON documents, text documents and Zip archives.

Currently, two XProc 3.0 processors are known to be available: XML Calabash 3.0 is in its final phase as a successor to the well-known XML Calabash, both developed by Norman Tovey-Walsh. This paper is based on MorganaXProc-III, which is the successor to the now retired MorganaXProc. Also developed by <xml-project />, this is a Java (or JVM) based implementation that, in addition to the core specification and the standard step libraries, also implements the file step library and most of the validation library, while also supporting Extensible Validation Report Language (XVRL). It has been around as a public beta since February 2020, received a lot of useful bug reports from users and was released as version 1.0 in September 2022. Since then, it has received monthly updates with bug fixes and feature enhancements. MorganaXProc-IIIse is an open-source product released under GPL 3.0. Coming later this year is a second, commercial edition called MorganaXProc-IIIee (Extended Edition). It provides support for almost all optional features of XProc 3.0, with complete coverage of the proposed step libraries as well as processor-specific steps such as image processing.

## 2. Introduction to existing batches

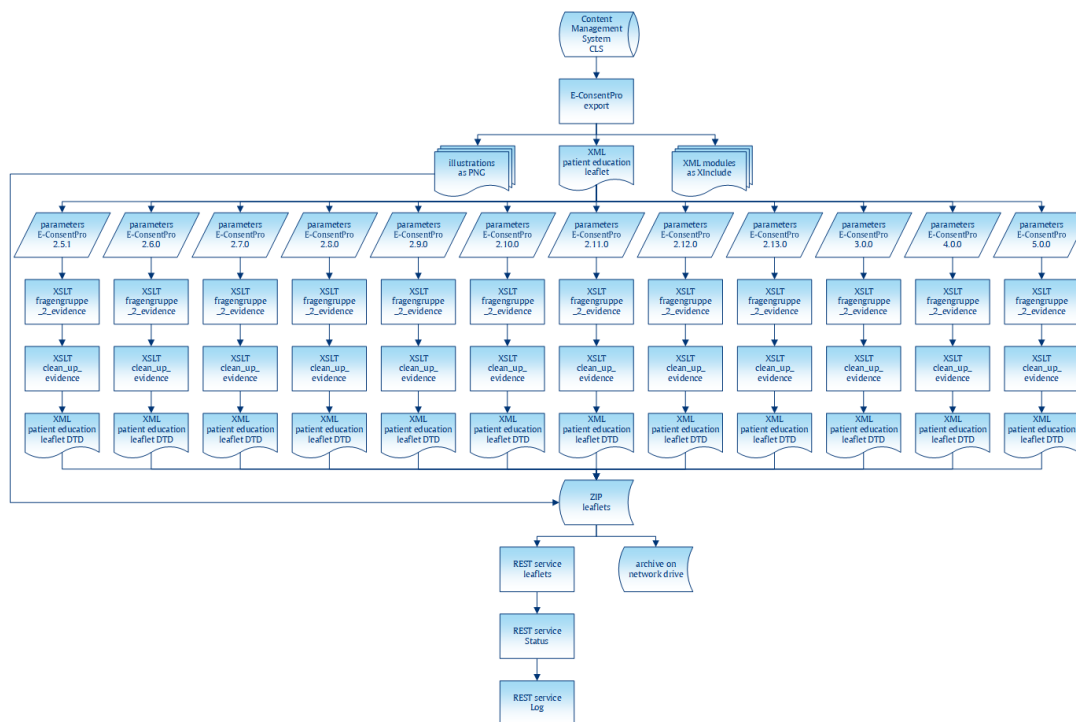
### 2.1. Batch “fragengruppe\_2\_evidence”

When it comes to the content delivery to Thieme Compliance's E-ConsentPro, the latest editions of the patient education leaflets are selected in the content management system CLS and an export is performed. The main XML file as well as all referenced XIncludes and images are exported from the database into a temporary folder on the CLS server. Here, the batch “fragengruppe\_2\_evidence” is used to

- ◆ Merge the main XML with the XIncludes
- ◆ Change the XML files from  $T_0$  XSD to  $T_0$  DTD by removing namespaces and inserting a Document Type Declaration
- ◆ Derive up to twelve different variants from the source XML file, each valid for the specific version of  $T_0$  DTD employed for the different versions of E-ConsentPro currently in use in the market

When the transformation is complete, Beyond Compare from Scooter Software is used to copy the images from the source to the result folder. The output is then zipped together with the referenced images and delivered via REST services using curl.

Figure 3. Batch “fragengruppe\_2\_evidence”



## 2.2. Batch “fragengruppe\_2\_FHIR-Questionnaire”

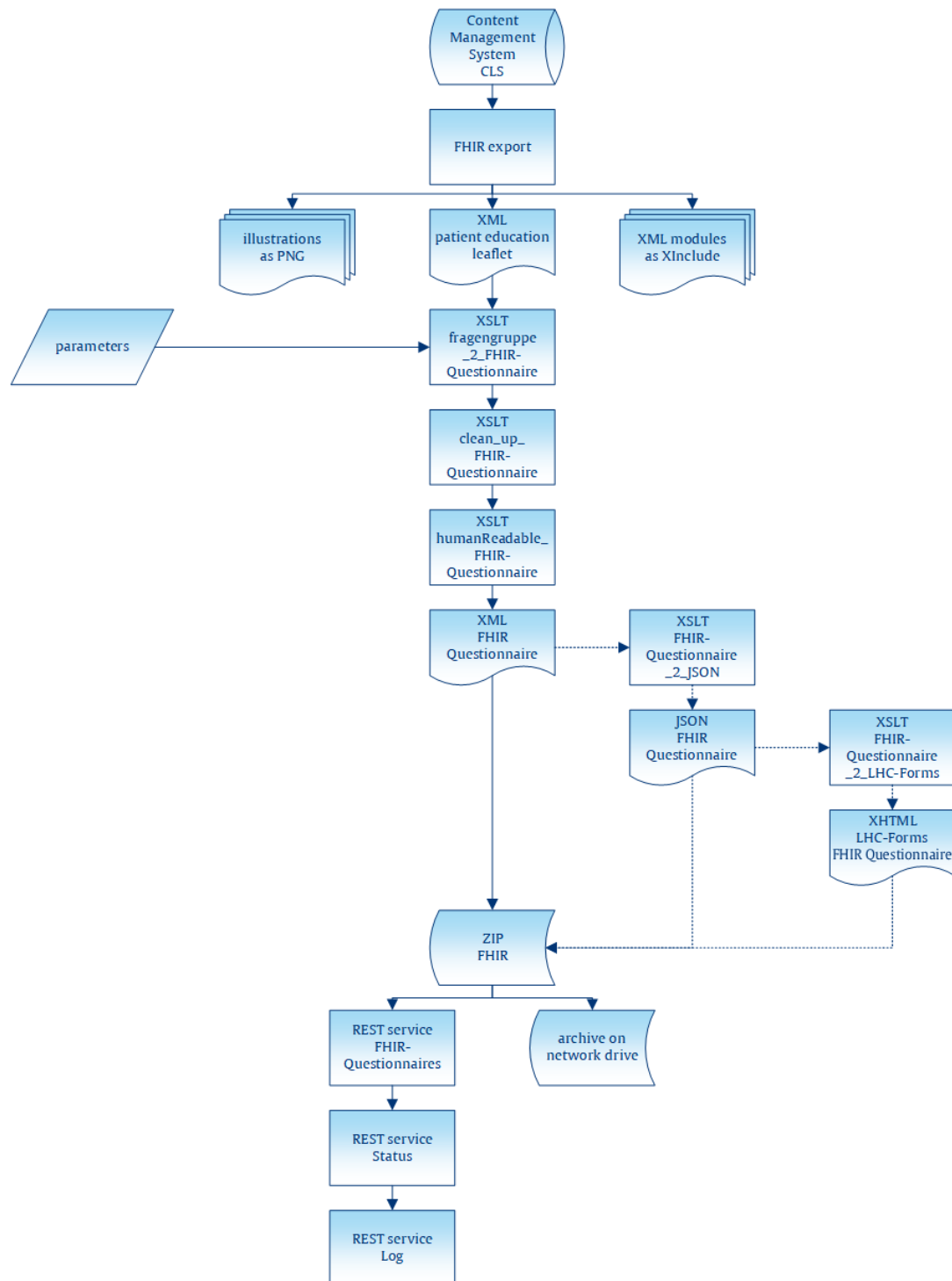
This second batch is used to deliver the medical history part of the patient education leaflet additionally as an HL7 FHIR questionnaire resource. Therefore, the main XML file as well as all referenced XIncludes are exported from the database into a temporary folder on the CLS server. A three-step transformation is then performed to:

- ◆ Merge the main XML with the XIncludes
- ◆ Generate the FHIR questionnaire resource and then clean it up (e.g. whitespace handling)
- ◆ Derive the human-readable part from the previously generated FHIR questionnaire resource and merge it back in
- ◆ Optionally transform the previously generated XML FHIR questionnaire resource into JSON
- ◆ Further optionally merge the just generated JSON FHIR questionnaire resource into an XHTML template for an output based on LHC-Forms

Afterwards, the generated results are zipped and delivered via REST services using curl. The images are not needed for this output and are therefore discarded.



Figure 4. Batch “fragengruppe\_2\_FHIR-Questionnaire”



### 3. Pain points of the existing batches

#### 3.1. Lacking of flexibility for inserting additional XSLT steps (in between)

Regarding the batch “fragengruppe\_2\_evidence”, it all started about a dozen years ago with just one single XSLT stylesheet. It was called with an initial template and used `fn:collection()` to process the entire source folder.

Over the years, more and more requirements have been added as well as, and new E-ConsentPro versions were released that had to be supported with the matching version of the XML content. As a result, the original XSLT stylesheet became more and more complex. In addition, some of the new requirements could no longer be sensibly implemented within a single XSLT stylesheet – for example, additional whitespace handling following a transformation performed on the original stylesheet. The simplest way to extend the existing transformation, without having to adjust anything else, was by adding another XSLT stylesheet using `@saxon:next-in-chain`.

Using `@saxon:next-in-chain` was an incredibly easy and effective solution, but it also comes with some downsides. Each XSLT stylesheet is orchestrated from the previous one. Inserting an additional stylesheet is inflexible and requires an unrelated XSLT to be changed.

Saxonica has since deprecated `@saxon:next-in-chain` and suggests using `fn:transform()` instead. While we are convinced that this function is a suitable replacement for `@saxon:next-in-chain`, but we have not looked into this simply due to the additional pain points as well as the requested improvements for existing pipelines.

### 3.2. No easy way to debug the intermediate results of each XSLT step

Debugging a multi-step XSLT pipeline can be an arduous process. A no longer matching `<xsl:template>` in a later XSLT stylesheet, because an earlier stylesheet has already changed the node(), occurs frequently. Storing intermediate results from each step therefore is necessary but is not as easy as it could be, based on the existing batches. XSLT provides `<xsl:result-document>` for this purpose. Until now, however, this is only inserted manually if needed and not added as a general rule. The XProc 3.0 pipeline needs to take this into account from the outset and simply offer this functionality by adding an invocation parameter.

### 3.3. Too many tools means too many dependencies

The main problem with the existing batches is the stability and maintainability of the pipelines. That is because the batches had to use different tools for specific tasks, e.g.:

- ◆ Beyond Compare for synchronising the XML files and images in the source folder and in the result folders
- ◆ 7-Zip to create Zip archives
- ◆ curl for transferring results to REST endpoints

The more tools are involved, the greater the dependencies and the greater the risk of breaking changes with future updates of these tools – of which there were quite a few in that dozen years. It also makes it harder to run the batches from different machines. That is because it has to be ensured that each machine provides these specific tools in their specific version. XProc 3.0 offers the necessary functionalities out of the box, which provided the opportunity to significantly reduce dependencies by minimising the tools involved in the pipelines.

## 4. New requirements for next version

In addition to the pain points mentioned above, there were also some requested improvements.

### 4.1. Future-proof approach and improved maintainability by adding a separate orchestration layer

As stated above, the existing batches are not only based on XSLT but also call other tools to fulfil different tasks. Using the command line interface of these tools has the



disadvantage of a strong connection between a logical task (e.g. create Zip archives) and a specific software implementation (e.g. call 7-Zip with the following parameters). Adapting to a new software version with changes in the command line interface involves changes in the batch. Using other software to fulfil the task is often associated with great costs. Languages such as ANT etc. introduce an extra level of abstraction that models the logical structure of the task and uncouples it from a specific implementation. Using such a level of abstractions makes workflows more robust against changes in the implementing software. Changing the task's implementation can be as easy as changing one configuration file instead of changing every workflow document that uses the task.

#### **4.2. Increased quality through validation of XML sources using T<sub>0</sub> XSD as well as validation of XML results using specific versions of T<sub>0</sub> DTD**

While the existing pipelines validate all XML source documents, XML results were not validated. Simply validating the XML results manually during stylesheet development and subsequently applying correct XSLT transformations while relying on valid results was sufficient. The nature of the source-to-result transformation for the XML documents is not one-to-one, but one-to-many – producing up to twelve versions of XML result documents for one source document. With regard to the quality of the results, none of these twelve XML results should appear in the Zip archive if at least one result document is not valid with respect to its specific version of T<sub>0</sub> DTD.

#### **4.3. Increased quality by additional validation of XML results using Schematron**

Schematron validations (of XML sources) are performed by Medical Editors and Content Managers during editing in <oxygen /> XML Editor. There is even a batch based pipeline integrated in the content management system to perform a Schematron validation using Skeleton that generates an easy to understand PDF from the SVRL report. Unfortunately this Skeleton implementation isn't integrated in the existing batches for the exports. Therefore the XProc 3.0 pipeline should perform a additional Schematron validation when the DTD validation mentioned above is done.

#### **4.4. Summarised, formatted and easily comprehensible log files**

The target audience of the content management system is Medical Editors, i.e. non-technical users. For every new edition of their patient education leaflet, they have to perform an export so that an integration test of their leaflet can be done in E-ConsentPro for quality assurance purposes. After moving to XProc 3.0, the pipelines will produce a lot more logging messages, e.g. because of the newly performed additional validations. The existing batches simply stored Saxon's error messages to different text files. For the non-technical audience, there should be just one single HTML file that serves as a summarised log. There should also be some basic CSS and the use of <details> to show/hide additional information as well as the possibility to filter log messages by error category.

#### **4.5. Performance improvement by omitting unnecessary images from the Zip archive**

The XML sources contain links to images supplied in the same folder as the XML sources. The improved pipeline has to make sure that all references are valid, i.e. every link goes to an existing image. The batch simply copied all images from the source folder to the result folders. The XProc 3.0 pipeline should be able to copy just the referenced images, but only the images referenced in the XML results, not in the XML sources. That is because there are cases when the pipeline omits the XML result because of special XPath conditions in the source or because the generated XML result is not valid against its specific version of T<sub>0</sub> DTD. Although these additional unnecessary images are subsequently ignored by the REST services, it would be better to just omit them. This would result in smaller Zip archives and therefore faster transmission to the REST service and faster processing.

#### 4.6. Limiting processing to specific sources from the source folder

When a new requirement is developed in the XSLT stylesheets, special test XML sources are created. Normally, the source folder in the file system contains all source XML files, which the new test XML sources are added to. Processing the whole source folder is necessary to avoid regressions. But this takes a couple of hours and, during stylesheet development, only the new test XML sources are relevant. So that the source folder does not have to be changed manually, an easy way to use another source folder and/or a specific file filter is needed. The XProc 3.0 pipeline should be able to use a custom source folder and/or a specific file filter as invocation parameters.

### 5. New system based on XProc 3.0

Having discussed the original batches and the requirements for the new system, we can now move on to the new pipeline system using XProc 3.0. If we take another look at Figure 3 [7] and Figure 4 [8], it is easy to see that the first one is a good deal more complex than the second one. Our coverage will therefore concentrate primarily on the pipeline replacing the first batch since the basic concepts and strategies for developing the two pipelines are fundamentally the same. The difference between the two pipelines is discussed further below.

The general approach in our move to XProc 3.0 was to replace the double control flow of the original solution with a single control flow in a pipeline. As Figure 3 [7] shows, the original batch incorporated twelve sub-batches, one for each parameter group. Inside each of these batches, an XSLT stylesheet was called to collect and process all documents in the input folder with the given parameters. As we wanted to eliminate as much batch scripting as possible, the first or outer control flow had to be replaced by an XProc 3.0 pipeline. Given the new requirements, we had to remove the inner, XSLT-based control flow as well, thus giving us one iteration over all documents in the input folder instead of twelve iterations (and potentially more in the future).

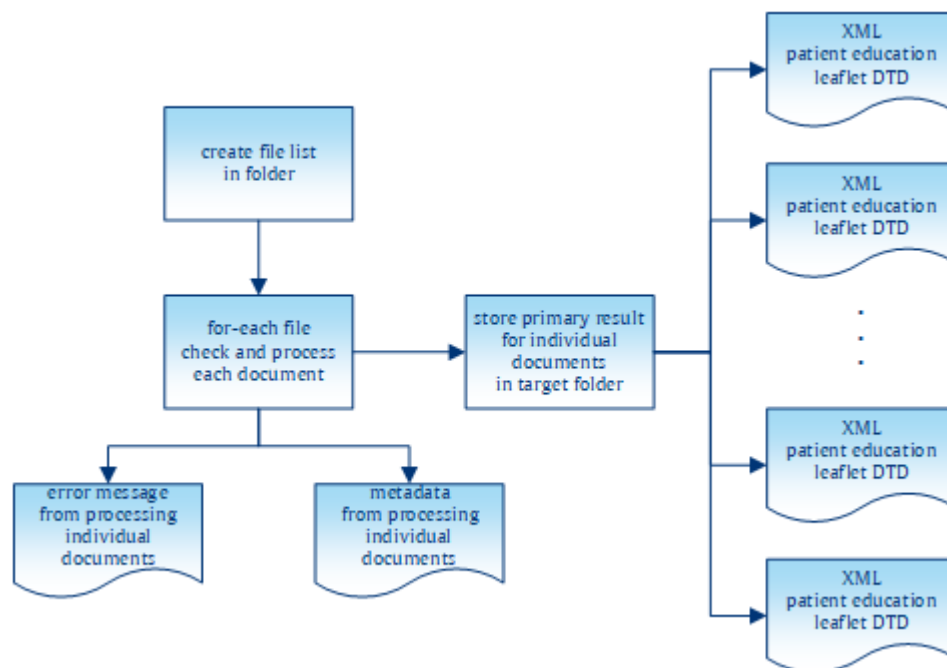
The basic reason for this was the new requirement to check the picture references: A document in the input folder should only be processed if all references to pictures in the document are valid, i.e. point to an existing file. As this is a property of an input document, it makes sense to address this once and for all, and hence start the twelve XSLT transformations only after all picture references have been checked.

Doing away with batch scripting also necessitates two other additions to the pipeline: As we no longer call XSLT via Saxon's command line, we can no longer use its powerful command line interface to perform an XInclude and to validate the source documents. As XProc 3.0 has the steps `<p:xinclude>` and `<p:validate-with-xml-schema>`, these two tasks can be easily performed before the stylesheet transformations begin.

The well-established and tested XSLT stylesheets are of course reused in the pipeline. However, this first stylesheet is not called from a batch anymore but via the `<p:xslt>` step instead. Additionally, the XProc pipeline uses a different entry point to the stylesheet than the batch: The latter called a named template that creates a collection of documents in a given folder matching a given pattern of file names. For each document in this collection, a template is then called by matching its root element.

The XProc 3.0 pipeline calls the same template by matching the document's root element, but creates the sequence of the documents to be processed itself: XProc's `<p:directory-list>` produces a document reflecting the content of a given directory, possibly using include and exclude filters. The pipeline then iterates over each `<c:file>` element to subject the respective document to processing.

Figure 5. A bird's eye view of the new system



The second change to the existing XSLT stylesheet is to unroll the `@saxon:next-in-chain` concatenation of stylesheets to a sequence of explicit calls of `<p:xslt>`. In Saxon, you can use `@saxon:next-in-chain` inside a `<xsl:output>` to direct the stylesheet's output to another stylesheet. This is a convenient way to chain stylesheets and thus decompose complex processing into a set of smaller stylesheets. This approach helps to improve the quality of code by breaking down a complex task into smaller pieces that are easier to manage. The downside is that `@saxon:next-in-chain` is a Saxon-only extension attribute, which is not supported by other processors and is not guaranteed to be a Saxon feature in the future. With the XPath 3.1 function `fn:transform()`, we have a way to chain stylesheet execution together in a standard-compliant manner. It would therefore be easy to rewrite the existing stylesheets to get rid of `@saxon:next-in-chain` and replace it with a cascade of `fn:transform()`. Hence, there is no need to change technology from XSLT to XProc if you are looking for a standard-compliant way to develop decomposed stylesheets and then chain them together to perform the general transformation.

In our case, using XProc 3.0 provides an additional advantage for developing decomposed stylesheets: With `<p:store>`, XProc 3.0 has a step for storing documents that can easily be used to improve debugging pipelines and – in our case – decomposed stylesheets. Unlike in XProc 1.0, `<p:store>` is fully transparent, meaning that the document on the input port is stored *and* delivered on the output port. Therefore adding a `<p:store>` (almost) anywhere in your pipeline does not break the “normal” flow of documents, but provides great debugging opportunities. Switching off debugging is also pretty easy in XProc 3.0: The attribute `@use-when` associated with a Boolean expression can be used on (almost) any step in XProc. If the expression is evaluated as `false`, the step (and all its descendants) are effectively excluded from the pipeline. Therefore, by using `<p:store use-when="expr" />`, we can easily switch the generation of debugging information on and off. Since XProc 3.0 also introduced static options that could be used in XPath expressions, switching debugging on and off directly from the pipeline invocation in the command line is the way to go. To sum this point up: Unrolling `@saxon:next-in-chain` concatenation into an XProc pipeline helps to improve the quality of code by splitting larger tasks into smaller pieces.

What we have discussed so far might be considered a general blueprint for embedding complex XSLT stylesheets into an XProc pipeline. Running this pipeline is, apart from the improvements in debugging, largely equivalent to invoking the original stylesheet. Let us now have a look at the advancements added to the overall process with the XProc pipeline. As you can see in Figure 5 [12], the general process can be described in three main parts:

1. Obtaining the URIs of the documents to be processed
2. Processing any individual documents and (possibly) storing the results to disk
3. Collecting the information from the individual processing to
  - a. Create the Zip archive
  - b. Generate a report about the processing

In the overall design, the `<p:for-each>` step has two purposes: (1) It processes every selected document, i.e. applying the XSLT stylesheets and, if the processing was successful, storing the produced documents to disk. (2) It produces meta information about the document processing: For every processed source document, a report document is created. The latter contains success or error reports for each stage of the source document processing, references to the image files in the source document, and a list of URIs pointing to the created result documents. We will look at these report documents later. For now, let us focus on the first purpose of the block inside the `<p:for-each>` step.

When unrolling the `@saxon:next-in-chain` sequence from the original batch, we first get a sequence of two to six `<p:xslt>` steps, each followed by a `<p:store>` for debugging purposes. The first improvement to be added to the pipeline was the validation of the result document(s). Until then the quality of the result document(s) relied on the validity of the source document and the correctness of the XSLT stylesheet(s) to produce the result documents. One requirement for the XProc pipeline was to add a validation for the result document(s) and store only those documents that prove to be valid. As XProc 3.0 defines steps for validation with RELAX NG, XML Schema and Schematron (as well as NVDL and JSON schemas), this could be achieved quite easily. The result of the last XSLT transformation is connected to the input port of a validation step. If the validation is successful, i.e. the document is valid, it appears on the step's result port and can be stored. If the document proves to be invalid, the validation step raises an error, which is then caught to create an element for the report document.

For the “fragengruppe\_2\_FHIR-Questionnaire” the validation takes the form of an XML schema and a Schematron document. To validate the produced FHIR document, you simply chain a `<p:validate-with-xml-schema>` and `<p:validate-with-schematron>` after each other to obtain a complete validation of the document. For the other pipeline, called “fragengruppe\_2\_evidence” the resulting documents are in-house documents for which a DTD is the authoritative grammar. This poses some problems to XProc 3.0 since, as you might have established from the aforementioned list of supported validation languages, there is no step for DTD validation included. With `<p:load>`, however, pipeline authors can ask for a DTD validation of the document to be loaded. As with the “normal” validation steps, this either returns a valid document or an error is thrown. Unlike with the normal validation steps, however, the document to be validated does not appear on the input port but has to be loaded from a secondary store via URI. To circumvent this restriction, we switched the steps around: Instead of validating the (in-memory) document first and then storing if the validation succeeds, we first store the document and then validate it via reloading. As a consequence, invalid artifacts are written to disk, which does not occur in the other pipelines where we can validate before the document is stored. This was a deliberate decision, because it would be easy to delete the stored document from disk once we have discovered its invalidity.

Another aspect of the quality improvements for the produced results is related to image referencing. Along with the source XML documents, the source folder also contains image files. Some of the image files are referenced in the documents, but not all. On the

other hand, documents might have a reference to a non-existing image file. To improve the overall quality of the results, the XProc pipeline was required to check the source documents for image references and to raise an error if an image file is referenced that is not present in the source folder. In addition, the pipeline has to keep track of the referenced image files, so that only referenced images are included in the resulting Zip archive.

With XProc 3.0, both requirements are pretty easy to fulfill: The pipeline iterates over all image references in the selected source documents and obtains their URI. With the `<p:file-info>` step, you can then obtain information about the referenced image: If the referenced resource exists, a `<c:file>` document appears on the `result` port. If the document does not exist, a `<c:error>` document will appear on this port. In the first case, an entry for the archive manifest is generated, In the second case, an error report element is created to flag the error for the report to be generated.

The outlined algorithm solves both shortcomings in the existing batch system: Since only images with a corresponding `<c:entry>` element are included in the Zip archive, the latter will only contain images actually referenced in at least one source document. And since every image reference in a source document is tested, the resulting report will include every error for invalid image references.

However, some post-processing of the image-related `<c:entry>` elements proved to be necessary: For efficiency reasons, the image references were tested for the source documents, but not for the produced documents. This is possible since the XSLT transformations do not add any image references. In general, there will be a 1:m relationship between the source documents and resulting documents, but since the produced documents are validated, there may be a 1:0 relationship if the validation for all produced documents fails. In this special case, there might be a `<c:entry>` element for an image document that is not referenced by any resulting document. Before actually generating the Zip archive, some cleanup has to take place. Every entry for an image documents records the URI of the source document it was derived from as well as any resulting XML document. For every image-related `<c:entry>`, the cleanup has to check whether there is a corresponding `<c:entry>` from a produced document. If not, the image-related entry is simply deleted, meaning that the resulting archive will only contain images actually referenced in a contained XML document.

The last aspect of the new XProc pipeline system to mention here is the improvement of logging or reporting. One new feature of XProc 3.0 that very much supports this requirement is the addition of `<p:catch>` with specified error codes. In XProc 1.0, we just had a `<p:try>/<p:catch>` where every error raised in the `<p:try>` block had to be handled in a single `<p:catch>` block. The ability to write different `<p:catch>` blocks for specific errors results in a more readable pipeline. This is also facilitated by the definition of more fine-grained error conditions for each individual step in XProc 3.0. This makes it very easy to identify what exactly went wrong inside an XProc pipeline, and to react to or report on the exact problems that occurred.

Our pipeline makes extensive use of `<p:try>/<p:catch>` to improve reporting, which is an important requirement as stated above. If you have a look at Figure 5 [12] again, the pipeline does not only produce XML documents representing the transformation results (shown on the right side of the code block), but also reports about the pipeline process (failing out at the bottom of the code block). For (almost) every document found in the source directory, the pipeline produces a report document. Here is an excerpt of such a document:

```
<tcg:report file="reference-to-source-doc">
  <tcg:report-done phase="validation">
    <successfully-validated />
  </tcg:report-done>
</tcg:report>
```

```

</tcg:report-done>
<tcg:report-done phase="processFileRefs">
  <c:entry name="name-of-zip-entry-for-pic1"
    href="path-to-pic1"
    found-in="reference-to-source-doc" />
</tcg:report-done>
<tcg:report-done phase="2.5.1">
  <c:entry name="name-of-zip-entry-for-doc1"
    href="path-to-doc1"
    derived-from="reference-to-source-doc" />
</tcg:report-done>
<tcg:report-error phase="2.6.0">
  <c:errors><!-- detailed error report here --></c:errors>
</tcg:report-error>
</tcg:report>

```

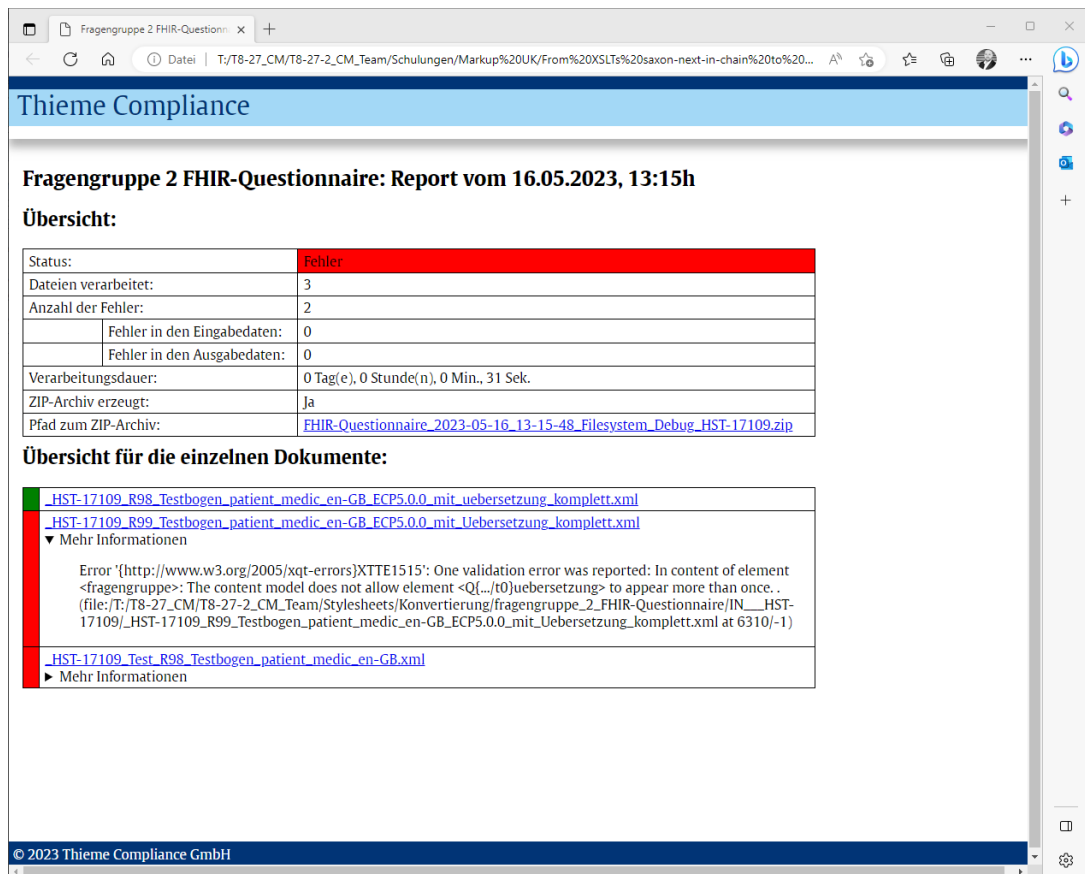
This snippet shows a processing report for a source document the URI of which is reported in the `@file` attribute. The document passes the source validation successfully, and it contains a valid reference to an image file. It is transformed in phase "2.5.1", thus producing a result stored at the recorded position. However, the transformation in phase "2.6.0" was not successful, as a result of which a `<tcg:report-error>` is included to flag the error and to report corresponding details. The `<c:entry>` elements are used to create the Zip archive. The attributes `@found-in` and `@derived-from` are used to make sure that an image file is included in the archive only if the source document containing the image references also passes transformation.

Based on the collected processing reports, the last two tasks of the pipeline are performed: creating the Zip archive and then creating the final report. Creating the archive is pretty straightforward. XProc 3.0's `<p:archive>` step has an input port `manifest`. The expected document has a `<c:manifest>` root element with `<c:entry>` elements denoting the expected archive entries. As we already created `<c:entry>` elements in our document reports, we just need to extract those elements from the report, wrap it in the expected element root and then call `<p:archive>` to create the Zip archive.

The final report is also based on the collected report elements: The reports contains information about every single processed XML document in the source folder, to which we add aggregated data such as the number of processed files, the number of detected errors or the complete processing duration. Finally, this report is processed with an XSLT stylesheet to create an HTML document and thus provide the improved logs requested in the initial requirements.



Figure 6. Summarised HTML log



The screenshot shows a web browser window with the address bar displaying a URL related to a FHIR questionnaire. The page title is 'Thieme Compliance'. The main heading is 'Fragengruppe 2 FHIR-Questionnaire: Report vom 16.05.2023, 13:15h'. Below this, there is a section titled 'Übersicht:' which contains a table with the following data:

Status:	Fehler
Dateien verarbeitet:	3
Anzahl der Fehler:	2
Fehler in den Eingabedaten:	0
Fehler in den Ausgabedaten:	0
Verarbeitungsdauer:	0 Tag(e), 0 Stunde(n), 0 Min., 31 Sek.
ZIP-Archiv erzeugt:	Ja
Pfad zum ZIP-Archiv:	<a href="#">FHIR-Questionnaire_2023-05-16_13-15-48_Filesystem_Debug_HST-17109.zip</a>

Below the table, there is a section titled 'Übersicht für die einzelnen Dokumente:' which lists two documents with their respective error messages. The first document is '\_HST-17109\_R98\_Testbogen\_patient\_medec\_en-GB\_ECP5.0.0\_mit\_uebersetzung\_komplett.xml' and the second is '\_HST-17109\_R99\_Testbogen\_patient\_medec\_en-GB\_ECP5.0.0\_mit\_Uebersetzung\_komplett.xml'. The error message for the second document is: 'Error '[http://www.w3.org/2005/xqt-errors]XTTE1515': One validation error was reported: In content of element <fragengruppe>: The content model does not allow element <Q[.../t0]uebersetzung> to appear more than once. (file:/T:/T8-27\_CM/T8-27-2\_CM\_Team/Stylesheets/Konvertierung/fragengruppe\_2\_FHIR-Questionnaire/IN\_\_HST-17109/\_HST-17109\_R99\_Testbogen\_patient\_medec\_en-GB\_ECP5.0.0\_mit\_Uebersetzung\_komplett.xml at 6310/-1)'. The footer of the page reads '© 2023 Thieme Compliance GmbH'.

As stated above, since the two batches were similar in their basic concepts, the two pipelines will also resemble each other. There are two basic differences to the pipeline already discussed:

1. While the first pipeline is only a cascade of two XSLT stylesheet, the FHIR has a total of five XSLTs. The implementation is pretty straightforward as the result of the first `<p:xs:lt>` step serves as the input for the next one. As a consequence, instead of two consecutive calls to `<p:xs:lt>`, we have a sequence of five interconnected calls in the FHIR pipeline.
2. The second, more conceptual difference is the way that the pipeline results are validated. While the pipeline discussed above uses a DTD validation, FHIR documents employ a different concept. The standard defines an XML schema and a Schematron schema to validate the documents. As both of these validation technologies are supported in XProc 3.0 by individual steps, incorporating FHIR validation simply requires a pair of `<p:validate-with-xml-schema>` and `<p:validate-with-schematron>` with the produced document on the default input port.

## 6. Takeaways

What are the takeaways from our project?

### 6.1. Smooth transition to XProc 3.0

First of all, we have to say that the move from batch scripts to XProc 3.0 was a very smooth experience. Compared to developing pipelines with XProc 1.0, the process is a

lot faster. This is thanks both to the wealth of syntactic sugar and to the much cleaner concept of some frequently used steps. The type system for variables and options provides a new level of security. The availability of XPath 3.0 and the respective functions improves programming a lot. And we are pleased to say that we did not miss parameter ports for a second.

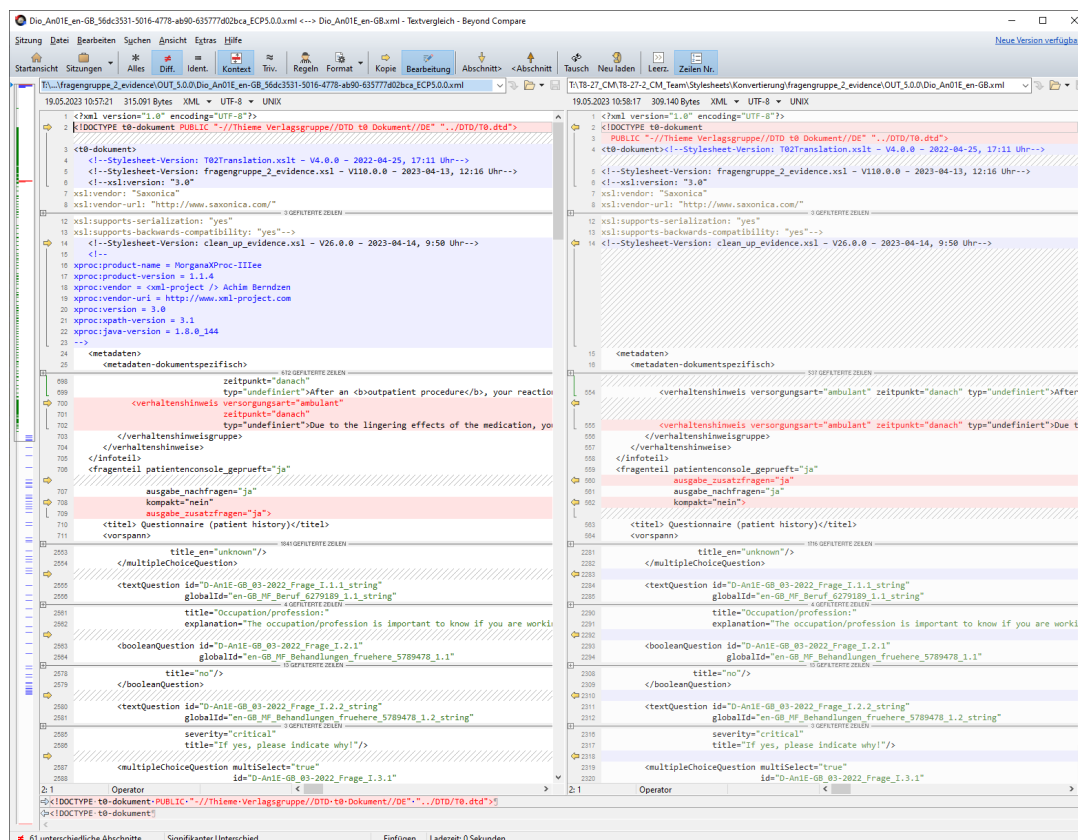
## 6.2. MorganaXProc-IIIse worked well and could even be improved over the course of the project

MorganaXProc-IIIse turned out to be a reasonable tool. The complete task could be fulfilled without any custom additions. Having to develop a complex pipeline system with a lot of documents to process helped a lot in improving MorganaXProc. During pipeline development, some bugs were found in the software and most of them have already been fixed. Additionally, we identified pain points for optimisation from using MorganaXProc in a large real-world project, which will be reflected in new features to be added in the future.

## 6.3. Serialisation is now done by MorganaXProc and no longer by Saxon

One thing to note is that with XProc 3.0 the serialisation is done by MorganaXProc even though the transformation is still performed by Saxon. Since the XSLT transformation is now orchestrated by XProc 3.0, a `<xsl:output>` declaration in the stylesheet is now ignored. In particular, the serialisation differs with regard to the order of attributes and whether or not they are presented in a new line. In addition, Also Saxon provides powerful serialisation features which MorganaXProc currently lacks. For instance, there is `@saxon:line-length`, `@saxon:attribute-order` or `@saxon:double-space`, which were used with the batch pipelines. These are useful features for (further) increasing the human readability of the XML result documents.

Figure 7. Serialisation done by MorganaXProc (left) and Saxon (right)



We had to ensure that the XML results of the XProc 3.0 pipelines are similar to the results of the batch pipelines. But there were two main issues:

1. There is some significant whitespace handling in the XSLT stylesheets.
2. Some transformations are now done in XProc itself using `<p:insert>` and no longer within the XSLT stylesheets.

We were not aware of the serialisation behaviour at first and, aesthetic considerations aside, these shortcomings in MorganaXProc-IIIse made it difficult to compare the results of the original batches with the pipeline results. Comparing the XML results therefore took more time, and the XMLs had to be pretty-printed first, with the downside that this falsified the whitespace handling.

#### 6.4. Performance problems with FHIR XML schema

The low performance of the FHIR pipeline gave us some headaches. It was significantly slower than the other pipeline although being very similar in its structure. Turned out that the result validations with the official FHIR schema was the culprit. Initially we used “fhir-all” in `<p:validate-with-xml-schema>`, but that was extremely slow. Turns out that schema document just imported about 140 other schema documents, and each of them importing the same schema. Luckily there is an other official schema document “fhir-single” which contains a complete schema document. Using this schema document did the trick and we are now happy with the pipeline's performance.

#### 6.5. XProc pipeline optimisation by loading stylesheets only once at the beginning

The pipeline development was not without drawbacks: The first (and natural) approach to processing the documents in the input folder would be:

```
<p:for-each>
  <p:xslt>
    <p:with-input port="stylesheet" href="the-stylesheet.xml" />
  </p:xslt>
</p:for-each>
```

However, if you process around 15,000 source documents, the stylesheet document is loaded each time too. Since this is completely inefficient, we changed it to:

```
<p:load href="the-stylesheet.xml" name="stylesheet" />
<!-- ... -->
<p:for-each>
  <p:xslt>
    <p:with-input port="stylesheet" pipe="@stylesheet" />
  </p:xslt>
</p:for-each>
```

This is a bit better performance-wise, but makes the pipeline more difficult to read. Moreover, it only partially resolves the inefficiency issue because the stylesheet has to be compiled every time the `<p:for-each>` is executed. The same applies to the XML schemas or the Schematron. They have to be prepared every time to be usable, although they do not change in our case. Using an XML catalog does not help here, because it only caches the document. The most effective solution would be to cache the ready-to-use stylesheet etc. But simply caching them as default processor behaviour is not feasible in XProc 3.0. Looking at the specifications, it is perfectly possible to rewrite documents or stylesheets etc. within `<p:for-each>` so that a later iteration depends on documents produced in an earlier one.

A processor could perform some optimisation here by checking whether a certain document is written inside a `<p:for-each>`. However, since catalog resolution etc. frequently takes place in XProc, there is no simple and reliable way to do this. From our perspective, some more investigation of this problem is necessary: One solution could be an (extension) attribute on `<p:with-input>` allowing a pipeline author to declare a stylesheet, schema etc. to be cacheable. Whether this is implemented in XProc at language level or takes the form of a vendor-specific extension is a discussion for the future.

## 6.6. Feature request for XProc: please add `<p:validate-with-dtd>`

XProc 3.0 offers a whole range of possibilities for validating documents using:

- ◆ `<p:validate-with-json-schema>`
- ◆ `<p:validate-with-nvdl>`
- ◆ `<p:validate-with-relax-ng>`
- ◆ `<p:validate-with-schematron>`
- ◆ `<p:validate-with-xml-schema>`

However, there is no equivalent for validating XML documents with DTD.

Even though XML Schema became a W3C recommendation already in 2001, and RELAX NG was also defined then, there are still legacy systems that only support DTD. In rare cases, it is possible to enhance these systems by switching to XML Schema, but this is generally not an option and these systems have to be supported nevertheless.

The lack of DTD validation in XProc 3.0 means that the pipelines have to use a workaround. The XML result must first be stored to the file system and then loaded immediately from there again using `<p:load>`. This is because, surprisingly, DTD validation is possible with XProc 3.0 but only while loading a document. Finally, the previously stored and validated XML result has to be deleted, because its no longer needed. While this workaround does its job, the ability to do this in memory using something like `<p:validate-with-dtd>` would be much more effective.

There is perhaps an argument for adding DTD validation to the pipeline using `<p:declare-step>`. However, this requires the XProc processor to realise that only a memory stream is needed, causing it to perform an optimisation and avoid storing the result to the file system and reloading it. Even if this would work, anybody who wants to support DTD has to add the following step to their pipelines instead of just using `<p:validate-with-dtd>`.

```
<p:declare-step type="tcg:validate-with-dtd">
  <p:input port="source" sequence="false" content-types="xml" />
  <p:output port="result" sequence="true" content-types="xml" />

  <p:pipe step="load" />

  <p:store href="foo.xml" name="store" />

  <p:load href="foo.xml" name="load" parameters="map{'dtd-
validate' : true()}" depends-on="store" />
  <p:file-delete href="foo.xml" depends-on="load" />
</p:declare-step>
```

So please, XProc working group, support DTD validation natively and add `<p:validate-with-dtd>` and Bob's your uncle.

## Bibliography

[Siegel:2019] Erik Siegel. *An introduction to XProc 3.0*. XML.com. <https://www.xml.com/articles/2019/11/05/introduction-xproc-30/>. 2019.

[Siegel:2020] Erik Siegel. *XProc 3.0. Programmer Reference*. XML Press. Laguna Hills, CA. 2020.

[Siegel:2020a] Erik Siegel. *XProc 3.0 - Connecting steps using ports*. XML.com. <https://www.xml.com/articles/2020/01/23/xproc-30-connecting-steps-using-ports/>. 2020.

[Siegel:2020b] Erik Siegel. *XProc 3.0 - Strategies for merging documents*. XML.com. <https://www.xml.com/articles/2020/11/16/xproc-30-strategies-merging-documents/>. 2020.

[7-Zip] is a file archiver with a high compression ratio. <https://www.7-zip.org>

[Beyond Compare] is a data comparison program developed by Scooter Software. In addition to comparing files, the program can also compare entire directories as well as FTP directories and archives. The program's particular strengths include its scripting capability and its usage of built-in and extendable comparison rules for different file formats. <https://www.scootersoftware.com>

[CLS] Content Lifecycle System (CLS) is a component content management system from Empolis Solutions GmbH. <https://www.empolis.com/>

[curl] is used in command lines or scripts to transfer data. <https://curl.se>

[E-ConsentPro] is a software solution for digital patient education developed by Thieme Compliance GmbH <https://thieme-compliance.de/en/products/e-consentpro-software/>

[HL7 FHIR] The Fast Healthcare Interoperability Resources (FHIR, pronounced "fire") standard is a set of rules and specifications for exchanging electronic healthcare data. It was created by the Health Level Seven International (HL7) healthcare standards organisation. <https://www.hl7.org/fhir>

[LHC-Forms] is a widget that renders input forms for web-based applications based on FHIR questionnaires provided by the National Library of Medicine. <https://lhcfirms.nlm.nih.gov/>

[LOINC] (Logical Observation Identifiers Names and Codes) is an international system published by the Regenstrief Institute for the unique identification and coding of medical observations, especially laboratory tests. <https://loinc.org>

[SNOMED CT] (Systematized Nomenclature of Medicine and Clinical Terms) is currently the most comprehensive health terminology in the world, a steadily growing ontology of preferred terms and their synonyms. It is maintained and distributed by SNOMED International. <https://www.snomed.org>

[XProc] is an XML based programming language for processing documents in pipelines, which involves chaining conversions and other steps together to achieve the desired results. The current version is 3.0. <https://xproc.org/>

