Publishing and Output

We’re going to talk a bit now about how to situate TEI within the spectrum of practical uses that arise in digital projects: for instance, publishing, working with data as part of a project’s work flow, linking and exchanging data between projects. Because this is a huge topic, our approach here isn’t going to be to try to cover everything, but rather to provide a sort of conceptual framework and a few strategic insights, and then let your questions guide us towards more detail in the areas you want to know more about.

We’d like to focus on a few key topics in particular:

- the single-source model of XML publishing
- the different tools that are useful in working with TEI/XML within a project work flow
- the ways in which our TEI data may evolve during the course of a project (which will affect our publishing and output systems)
- the life cycle of project development, and the different TEI/XML tools that are appropriate and within reach at different stages

Build Once, Use Many

We can sum up the single-source approach to digital publishing with the slogan build once, use many: you create a single data source in XML (in our case, in TEI), and from that source you derive a variety of output formats through automated processes, leaving the source untouched. If you need to make changes to the output, you change the appropriate output process. If you want to make changes to the content, you change the source.

The single-source model of publishing has been considered a key advantage of descriptive markup systems since the birth of SGML in the late 1980s, for several reasons:

- it allows for a more efficient work flow, since you only make substantive changes to your content in one place (so no duplication of labor, diminished version control problems)
- it allows for quick updating of individual outputs (e.g. to adjust the formatting)
- it reduces dependencies of one output on any other (so you can change the way lists are handled in one kind of output without affecting the way they are handled in other kinds of output)
- most importantly, it makes your data independent from your publication tools

In a modern TEI/XML publishing environment, the kinds of tools we typically use to create our output include:

- XSLT (the master tool) to convert the source XML into other XML or non-XML formats
- possibly programming tools like Perl, Python (?), PHP (?) to perform additional conversions or other peripheral tasks
- XML databases and publishing frameworks (eXist, XTF, TEIPublisher, PhiloMine) to provide
comprehensive searching and display
• CSS as a formatting tool for HTML
• visualization tools that take derived formats and display them in interesting ways

And the kinds of outputs we typically see in play are:

• HTML (for web viewing)
• PDF (for printing)
• KML, GIS, JSON, other formats as needed for specific web tools
• other specific data formats to expose as needed: for instance, OAI metadata, MODS or METS (for ingestion into a repository), RDF
• less often, formats like RTF, tab-delimited data, formats intended to feed data to specific programs rather than for use in a web environment

Slide showing XML source and varied outputs, with routes: XSLT, CSS, FO, visualizations etc. (and also intermediate systems like databases, intermediate formats)

Work Flow

Within our TEI projects, there are many different kinds of output that are important at different points in the project work flow:

• error detection: proofreading, encoding review, finding inconsistencies in the markup: for these kinds of activities, first, we count on validation tools of various kinds (mention both schemas and schematron). But XSLT can also be incredibly helpful, by allowing us to extract specific pieces of our data and creating views of the data that are particularly designed for catching certain sorts of errors. For instance, if you’re encoding personal names and place names, one way to check the accuracy of the encoding is simply to create a big list of all the things that have been tagged as <persName>, and have someone run their eyes over it to catch things that don’t belong. XSLT makes this task easy. We can also write an XSLT stylesheet that identifies (for instance) all the poems marked as sonnets that have more or fewer than 14 lines. Similarly, we can create programs using languages like Perl that will look for particular patterns in the data that could be symptoms of errors (for instance, missing spaces around elements) and report them to us so that we can check them.
• Of course once a project is actually publishing its data, there are tools for TEI "publishing" proper: that is, creating formatted output for public consumption: XSLT, XQuery, XML databases, XML publishing systems, FO, CSS, HTML/JSON/KML/etc., outputs for visualization tools
• It’s also useful to consider another kind of publishing that is less visible to users but that has always been important to the TEI and is increasingly important in a linked-data universe: namely the extraction, generation, and sharing of data to feed other processes, internal or external. In this category we would include APIs and web services through which project data can be exposed for other projects to exploit, and also data that is generated with the goal of interchanging it or aggregating it with other projects’ data. For example, participants in the NINES project generate metadata which is ingested by NINES and which serves as a kind of proxy for the project within the NINES framework. Another example: the MONK project aggregated TEI data from a variety of sources and converted it all to a common TEI format, so that it could be used under a common suite of tools. XSLT is the key technology for handling these kinds of extractions and transformations.
Finally, we should note the increasing importance of archiving and data curation, as more and more projects reach maturity and need to think about where their data will go after the project’s conclusion. In this context, there are repository tools that house and manage TEI data (to varying degrees of functionality). We are less familiar with these tools, largely because we haven’t had to deal with them; Brown’s digital repository is built on Fedora, which is at the more powerful and flexible end of the spectrum, and it permits the repository developer to associate specific behaviors with a given kind of data: so for instance you can specify that a given set of TEI files will be viewed/processed using a specific set of XSLT stylesheets. Other repository systems will happily ingest and store TEI data, but without providing for any TEI-specific access or behavior.

Evolution of XML

One very important thing to bear in mind is that over the course of time, your project data will change. There are several reasons for this:

- changes to TEI: P4, P5, P6...the TEI is a research community and doesn’t stand still
- changes to project strategy: you may change your mind about what to encode, how to represent your data; your audience may make new demands.
- successive enrichment of XML during successive phases of the project’s development: this is often a smart way to build a project during successive funding cycles, by doing the encoding in layers.

Whatever the reason, there are some important consequences of these changes:

- When your data changes, your output systems will probably also need to be updated (although those updates may be very simple)
- Updating your data may in many cases be automatable (for instance, the upgrade from P4 to P5 was largely automatable)
- But if you are adding more information to your encoding (for instance, marking some new feature, or making a more fine-grained distinction) then it will involve hand work and human judgment. (This can be a good thing since it provides a rationale for continued funding.)
- It is a good idea to plan these changes in advance.

Developmental life cycle

Note that this table is an oversimplification: what is easy really depends on a lot of local and individual factors that determine the kinds of help you can get.

1. private prototyping/experimentation: one or two people, not much programming skill/cycles, no funding

- CSS, XSLT/CSS [eXist?]
- NB use of systems like TEI Publisher, XTF as well
- NB risks of matching your data to the system; better (but more challenging in some ways) to work directly with XSLT and similar "building block" tools, or to tweak/adapt the system
- NB crucial role of XSLT here to generate whatever tool-specific data is needed by specific tools, leaving the archival data untouched and pristine
What's easy about the tools in this box is that they don't require you to understand or configure or have access to the guts of a server.

2. funded development: XML databases, XML publishing systems: small group, access to project-level programming skills/cycles
   - XML publishing tools and systems: eXist, XTF
   - services like TAPAS which provide an infrastructure for publication

These tools do assume that you have access to a server and have someone to wrangle it (maybe that's you, or a member of the project team).

3. mature large-scale publication: larger team, more embedded in institutional systems and development teams; more resources, longer-term. As above, plus:
   - Fedora, MarkLogic, etc.
   - more embeddedness in/attention to dissemination and metadata systems like OAI, COINS, etc.