Rick Jelliffe://From Grammars to...

The Schematron

Information is Molecular!

When one atom of information
requires another piece to give it meaning,
we say there is **cohesion** between these atoms.

For example, to know some *size* information,
we need some *number* and some *unit* of measurement.

Data is Structural!

When we represent information as data
in a computer, we create structures to represent it.
These structures **couple** data together in various ways.

For example,
in an XML document,
two atoms of data may be coupled
as an element with an attribute,
or as two elements under the same parent element,
or as parent and child elements.

Data Stuctures are Models

The way that data is **coupled** may not reflect the actual **cohesion** of the information.

For example, in a relational database data must be put in tables, in well-formed XML data must fit into trees. This is useful, because adopting an appropriate modeling **discipline** may have nice implementation properties.

But we are no longer representing the information; we are trying to model it within the constraints of the tools we are using.

Kinds of Cohesion

- **Strong** cohesion is where one piece of information is necessary for another to be useful.
- **Weak** cohesion is where one piece of information may relate to onother, but it is not necessary.
- **Neutral** cohesion is where one piece of information does not effect another.
- **Repulsion** is where one piece of information diminishes the usefulness of another.

Note that cohesion may be equally strong in both directions.

Kinds of Coupling

- **Strong** coupling is where one piece of data is structurally tied to another.
- **Weak** coupling is where one piece of data may be structurally tied to another.
- **Rejection** is where one piece of data must not be coupled to another.

What is a Structural Schema?

A structural schema
is an attempt to model
the kinds of information **cohesion** using
the kinds of data **coupling** available in
a particular *paradigm*.

DTDs

A DTD uses the paradigm of a *regular grammar.*

The kinds of coupling available are:

- *parent−>child* **strong** coupling: a required subelement;
- *parent−>child* **weak** coupling: an optional subelement or group; an element declared ANY is a shorthand for weak coupling to all other elements in the schema;
- *sibling−>next sibling* **strong** coupling, in the context of a particular group in a particular parent: sequences;
- *sibling−>sibling* **rejection** though alternatives (i.e., |, when not directly inside a group with a ⋆ or +);
- *sibling−>next sibling* **weak** coupling, in the context of a particular group in a particular parent: repetition;
- *parent−>child* **rejection**: an element declared EMPTY rejects all other elements.

Limits to Grammars

All kinds of information cohesion must be represented through these available kinds of data couplings.

A strong cohesion between two pieces of information that are modeled as grandparent and grandchild elements requires *grandparent–>grandhild* coupling. In DTDs, this must be expressed through a strong *parent–>child* coupling of from the grandparent, and then a strong *parent–>child* coupling to the child.

In particular, rejection and neutral coupling are only model implicitly, as the result of modeling **strong** and **weak** *parent–>child* coupling.

Tree Patterns

The Schematron uses the paradigm of *tree–patterns*. These can model:

- *x*–>*x* **strong** coupling, in the context of a pattern: the asssert statement ;
- *x*–>*x* **rejection**, in the context of a pattern: the report statement;

where *x*–>*x* can include any relationship expressible by starting from a location given by an XPath *path* in a document and evaluating an XPath *expression* from that location. This includes *ancestor–>descendent*, *cousin–>cousin*, and even graph relationships expressed using IDs or keys.

Grammars versus The Schematron

- Tree-patterns allow more one-to-one relationships of data coupling to be directly specified; in turn this means that more kinds of information cohesion can be directly modelled;
- Grammars allow, through the grouping mechanism, more kinds of complex linear patterns to be specified;
- Grammars cannot express any relationships that cannot be reduced to chains of parent-child or sibling->nextSibling relations;

So we can say that a grammar based system is mainly interested in defining an element and the things that it can contain: the subject is the **element** and the vocabulary is the **content model**: groups, optionality, repitition.

A Schematron schema is interested in how an element fits into a pattern, which may include coupling or rejection from any other part of the data structure: the subject is the **pattern** and the vocabulary is the **path** and the **expression**.

Converting a DTD to a Schematron Schema

First, create a list of all element types used in the DTD.

Then for each element type in the DTD:

- create a single rule in your Schematron schema;
- add an assert statement for each required element type in the content model and each required element: this just tests the child axis;
- if the element is EMPTY or only (#PCDATA), generate a report element that tests is there is any element content;
- for each element type in the content model add the name of the parent element to a list of allowed parents;
- add a report statement that tests the parent axis for any element that is in the list of all elements but is not in the list of allowed parents.

The effect is that we test for all **strongly** coupled children and all **rejected** parents and all explicity **rejected** children. (Note that this procedure can be performed automatically.)

Improving the Conversion

There are several simple improvements that can be made:

- keep track if there is a fixed number or range of occurrences of an element type in a content model, and make a test for this;
- create assert statements testing the element type against a fixed position, for each required and non-repeating elements in the content model in the DTD, starting from the start and ending when some repeating or optional element type occurs;
- keep track of each element type that can possibly follow another element type in the content model, and make assert statements: this will have to be in a separate pattern, the context of each rule must specify the parent/child where the parent is the element type being defined and the child is the element that may be followed--there would be one assert statement testing all possible next siblings.

Result

The first conversion in effect gives us the equivalent of a weaker validation of the DTD's content models. It is the same as disallowing groups and replacing ? with $\star$, adding + everywhere else, and replacing , with SGML's &. Note however that we have gained openness: an element from outside the DTD will not generate an error, except in an EMPTY element or one with data content only.

The improvements gives us much more of the power of regular grammars, remembering that content models for XML tend to not have complex sequence requirements; the rules about when #PCDATA can be used and the lack of the & indicator are disincentives for intricate content models compared to SGML.

(These steps can also be automated.)

Finally

As a last step, you can add by hand the kind of information that a DTD cannot express: notably which elements are allowed to be at the root of a document, which elements should not contain themselves (SGML global exclusions). If the DTD was an SGML DTD, many global inclusions can be modeled by asserting a **rejection** on the element if the required ancestor is not present. You may find that many of these DTD-derived rules hide more expressive patterns.

A Schematron schema is based on a different *paradigm* than grammars: each is useful, but each can only incompletely simulate the other. For the simplest data, database records with fixed structures, this data only requires **strong** *parent->child* coupling: both grammars and Schematron schemas will be equally powerful.

The aim of a structural schema is to model information cohesion in the data coupling. While there is a lot over overlap, both grammars and tree-pattern schema languages excell at different structures. Is it possible that in the world of namespace, open content models and data islands, the tree-grammar may be a more useful paradigm than the regualr grammar?