Chapter 11

Automatically Rearranging Structured Data for Customised Special-Needs Presentations

S. S. Brown and P. Robinson

Introduction

Computers are most frequently used for processing text, or for processing signal-based data such as graphics, audio and video. However, there is also a significant amount of non-textual symbol-based data, such as musical scores, mathematics, schematics, and scientific models, that are encoded in such a way that they can be manipulated by algorithms at a higher level than simply processing the signal. Such data can be used for complex search queries, for example.

In order to present such data to people, it must be sent to an output device, such as a screen, a printer, a Braille embosser, or a speech synthesiser. This usually involves one or more *transformations*, in which the data is changed by an algorithm. If an alternative form of presentation is required, perhaps for a special need, then this algorithm often needs to be customised, modified, or re-written altogether. Such modifications can be beyond a developer's resources, and this limits the ability of software to adapt the presentation to the needs of a diverse range of people. It also limits the extent to which end-users can customise the presentation to their own individual needs.

This paper proposes a different paradigm for representing and transforming structured data, which can help developers (particularly print-disabled developers) to build systems that automatically transcribe data into different formats. This benefits universal access because the methods of presenting data to print-disabled people are far more diverse than one might think, and the need cannot be completely addressed simply by writing a limited number of conversion programs or by adopting a standard "accessible" presentation.

Related Work

Diversity of Special Needs

Special needs are diverse and can be difficult to anticipate. A person's ability to use data in printed form may be hampered by a print disability, such as blindness, low vision, or dyslexia. It may also be hampered by educational and cultural differences — the person may have learned a notation system that is different from the one being used in the presentation.

The diversity of print disabilities is rarely acknowledged in the literature. Many papers use the phrase "blind and visually impaired" when discussing designs for blind people, implying that people with low vision work in the same way; in fact, many with low vision use their residual sight as much as they can. Other papers (such as Hermsdorf *et al.*, 1998) assume that the needs of all partially-sighted people are much the same. Jacko *et al.* (1998, 1999) show that this is not true, and suggest that a user's needs can be determined by clinical assessment. Even this can be difficult in the case of some eye conditions, such as nystagmus, which can vary over time and can produce different perceptual results for different people (Taylor & Harris, 1999). Some users need to be given control over the presentation themselves, as Gregor and Newell explain for some cases of dyslexia (2000).

Many industry-standard applications already allow the customisation of fonts, sizes and colours, although it can be difficult and is not always reliable. Not all of these applications allow the layout of structured data to be changed to compensate for the reduced viewing area to text size ratio (a problem that is also prevalent in the mobile telephone industry). Also, people who have difficulty fixing their gaze can sometimes be helped by different layouts, and this requires even more customisation.

Educational background can also contribute to the requirement for an alternative presentation. A good example of this is in music — besides Western staff notation, musicians use various tablature and instrument-specific notations, as well as *sol-fa*, Chinese *Jianpu*, and others, and it is often possible to transcribe a piece of music from one notation into another in order to make it accessible to a greater number of musicians. Braille music also has numerous different versions across the world, and most existing transcription software, such as Goodfeel (McCann), cannot generate all of them. And sometimes an educational establishment will customise a notation for pedagogical purposes — while students are learning a complex notation, they need documents that only use the subset of it that they already know (Ofsted, 2000).

Although it is common practice to focus on one need at a time, it is possible to envisage individuals who have a combination of several of the above-mentioned special needs, and this gives rise to even more diversity.

Existing Transformation Systems

There is no shortage of books and websites on XML (W3C, 2000), and on related tools for presenting it (such as the Apache Cocoon project), many of which make use of the XSLT transformation language (W3C, 1999). It is often claimed that this language is Turing-powerful and is therefore adequate for any transformation task.

However, some common transformation tasks are relatively difficult to achieve in XSLT, particularly for people with print disabilities, because so much code needs to be written. Consider, for example, matrix transposition. This is frequently encountered when processing musical scores, which can be structured either with many bars (measures) within each part or with many parts within each bar:

```
<SCORE>                            <SCORE>
  <PART>                             <BAR>
    <BAR> a </BAR>                     <PART> a </PART>
    <BAR> b </BAR>                     <PART> c </PART>
  </PART>                            </BAR>
                       ↔
  <PART>                             <BAR>
    <BAR> c </BAR>                     <PART> b </PART>
    <BAR> d </BAR>                     <PART> d </PART>
  </PART>                            </BAR>
</SCORE>                           </SCORE>
```

While this transformation can be done in XSLT, the resulting code is long and complex.

In an earlier project (Brown, 2000), the first author represented musical scores in an unstructured tuple space. Each tuple was a list of attribute/value pairs that represented an isolated event in the music, such as:

```
(type=note, letter=c, octave=2, barNumber=5, part=1,
time=5, notatedLength=2, realLength=1/2,
accidental=sharp, staccato=1)
```

A scripting language was developed for transforming this into the output; it contained a construct called `foreach`, which sorted the tuple space using a given attribute as the sort key, divided the result into subspaces (one for each unique value of the given attribute), and executed the enclosed code on each of these subspaces. Thus it was easy to write code that structured the output without having to consider how the input was structured, such as:

```
foreach bar
  foreach part
    foreach time
      …
    end foreach time
  end foreach part
end foreach bar
```

However, this model does not adequately deal with cases where the original hierarchy *does* need to be preserved, such as mathematical expressions (which are recursive). For these, XSLT is much simpler. A general system must be able to deal with documents that contain a mixture of text, music, mathematics and other notations (consider an encyclopaedia for example), which suggests that some sort of hybrid system is needed, which gives the benefits of both XSLT and the tuple space paradigm.

A Four-Dimensional Markup Language

Markup languages, such as XML, can be used for describing hierarchical structures over documents and data. A piece of data can be enclosed within an "element", which can in turn be a member of a higher-level element, and so on. As described above, this model becomes more difficult to work with when the data can be divided in more than one manner.

The proposed four-dimensional markup language (4DML) allows the scope of an element to be non-contiguous; this makes it easier to represent and address multiple, overlapping sets of markup. The document is represented as a set of points in a discrete four-dimensional space (fourspace) with the following dimensions:

- Element name;
- Element position (it is possible for elements of different names to share the same position);
- Element 'depth', indicating how deeply nested the element is in the tree (this is required for disambiguating between identical elements at different levels of a recursive structure);
- Scope. Each value on this axis corresponds to a unique instance of a symbol (a string of any length) in the document. No ordering is implied — that is defined by element positions. Symbols can be empty placeholders to represent elements that have no data.

An element has one point for each symbol in its scope (including those in the scope of its children). Thus there are many points for each symbol. To identify an element in the fourspace, it is sufficient to specify any one of that element's points.

It is not intended for the fourspace to be directly visible to the user; it is interacted with by converting to and from XML or another language (this paper uses XML without trailing closing tags). When representing XML, elements with empty names are added around XML cdata (to preserve the position information when mixing cdata with other child elements), and XML attributes are represented as children of a child element named `!attributes` (not valid in XML) which does not disturb the position numbers of the other children.

Transformation by Model

This operation takes as inputs two fourspaces — the input and the model (or template). The return value is the result of transforming the structure of the input to match that of the model. For example, the model to effect the transformation in the example above is `<SCORE> <BAR> <PART>`. Note that the models are written using the structure of the output only — the user does not have to know what (if any) transpositions should occur to achieve the desired structure.

The model fourspace is recursively traversed top-down (if two or more elements share the same position number then they are processed in an undefined order). For each element *e* in the model, the *input* fourspace is searched for elements with the same name as *e*, and only the least deeply-nested of those elements are considered. The input is divided into subspaces, one for each of these elements, with the elements themselves removed. The part of the model enclosed within *e* is then applied to each of these subspaces, and each result is “covered” with an element named after *e*. If *e* is a leaf node in the model, then each result is a symbol representing all the data in that subspace, in the order given by top-down traversal.

Reporting lost data. If the model does not specify every element, it is possible that some data will be lost. The algorithm can return this data, as a fourspace containing points that were in the input fourspace but were not used in generating the output. This will contain just enough of the input’s structure to show where the lost data was. If this is presented to the user in some way, it can be used for debugging the model, or at least for summarising what had to be lost in the conversion.

Parameters in the Model

Parameters to the model’s elements are stored in the same way as XML attributes. The following parameters are defined, and the design is extensible with new parameters. When doing this, one should not be afraid of redundancy, because different users approach tasks in different ways.

rename and nomarkup. `rename` specifies a new name for the element, to be used in the output. This is useful if the nomenclature is different (e.g. `<BAR rename="MEASURE">`). Element names that are empty (`rename=""`) are accepted; when converting to XML, no markup is outputted around such elements. For convenience, the parameter **nomarkup**, if present, is equivalent to setting `rename=""` for this and all child elements.

before, after, between, begin, end, and arbitrary text. `<TD before="a" after="b" between="c"/>` specifies that, if there are any `TD` elements at all, then *a* should be added as cdata *before* the first element, *b after* the last element, and *c between* each. Parameters `begin` and `end` are also provided for adding cdata *within* each element (at its beginning and end respectively). Additionally, any text in the model is copied to the output whenever it is encountered. Thus the model:

```
<TABLE> begin table
  <TR> <TD/> end tr </TR>
end table
```

will insert the text "begin table", "end table" and "end tr" at the appropriate places in the output; this could be used to output a structure in a non-XML language, particularly if `nomarkup` is used. However, the effect of `<TD> text </TD>` may not be obvious – the `TD` is no longer a leaf node in the model, so its contents are completely replaced by "text". The use of parameters `begin` and `end` is recommended instead.

In all cases, if XML is used to enter the model then there should be some means of representing characters like `<` or `"` in the text. This could be done by expanding XML entities (such as `&#60;`).

start-at, end-at and number. `start-at` and `end-at` can be used to restrict which elements are processed. For example, `start-at="4" end-at="5"` causes only the 4th and 5th elements that are found to be processed. For convenience, `number="`*n*`"` sets both values to *n*.

Reserved element names and wildcard. It is sometimes necessary to combine the above with a way of matching any element name; for example, the two parameters of a MathML `msub` element may need to be treated differently (`number` required) but each can be one of a number of possible element types. It would be useful to reserve a "wildcard" element name that, when encountered in the model, causes that model element to match on all top-level input elements, and not to remove the said input elements from the subsets it generates.

However, reserving special element names introduces complications when an element in the input has the reserved name. One way around this is to make the reserved name customisable (with no default). This can be done in the model — the `wildcard` parameter specifies the name of the wildcard elements, and applies within the scope of the current model element (perhaps the top-level element).

Namespaces. XML namespaces are a useful alternative to reserved words, and namespaces can also be defined which are equivalent to setting certain attributes in a model element. This can lead to more concise models; for example, if the namespace `seq` pointed to `http://ssb22.cam.ac.uk/set/sequential/1`, then `<seq:P>` would be equivalent to `<P sequential="1">`. However, the use of namespaces should not be enforced, since not all editors support them, and they can detract from the conciseness of small models.

children-only and sequential. The `children-only` parameter, if present, causes only elements that are (direct or indirect) children of the given element to be considered. For example, consider the effect of the model `<B><A>` on the input `<A`$_1$`><B><A`$_2$`/><A`$_3$`/>` (the subscripts are added for annotation). As it is, the model's `<A/>` will apply to the input's A$_1$, but if `<B children-only="1">` were used, it would instead apply to A$_2$ and A$_3$ (A$_1$ would be ignored).

`sequential` causes each input element's immediate children to be processed sequentially, rather than being grouped by element name as would normally happen. This will often be used when processing documents that contain a mixture of different types of object in any order (as is the case with XHTML). `sequential` implies `children-only`, and also that only an element's *immediate* children are considered by the next level of the model. As a special case, `sequential="cdata"` additionally causes any cdata elements (at that level) to be copied from input to output.

call. This is a means of adding recursion to the model. For example, in:

```
<math>
  <mrow call="math"/>
  <mi>…</mi>
  …
```

each `mrow` element is treated as though it were another **math** element. It is also possible to call elements non-recursively, as in:

```
<math>…</math>
<p sequential="1">
  <math call="math"/>
  …
```

external. Consider the effect of the model `<SCORE><PART><TITLE/>` … on the input:

```
<SCORE>
  <TITLE> … </TITLE>
  <PART> … <PART>
  …
```

The intention here is that each `PART` bears the `TITLE` of the `SCORE`. However, `TITLE`'s data is outside the scope of the `PART` element and will therefore not be present in the corresponding subset of the input when `PART` is processed. It can be reached by searching a display stack of fourspaces that contain all points not present in any of the subspaces generated by the model. Hence, if no `TITLE`s are found in the `PART`, then the `SCORE` will be searched (this search excluding all of the `PART`s); if no `TITLE`s are found there then the next level up will be searched and so on.

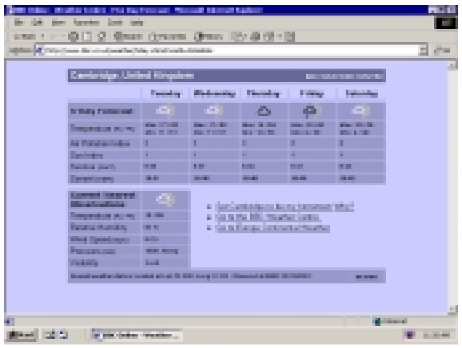
This behaviour can be overridden with the `external` parameter; `external="never"` would cause only the `PART` to be searched; `external="always"` would cause only the `SCORE` and higher-level elements to be searched.

Evaluation

The transformation by model algorithm has been implemented in Python. The model and input are read from XML files, and the output is generated as XML (or as plain text if **nomarkup** is used at the top level).

The prototype can be used from the command line or as a filter in a pipe. Additionally, a graphical user interface was constructed that displays the model, output, and lost data as tree controls (similar to those used in Windows Explorer), and allows the user to edit the model tree while observing the effects of these changes on the other two trees.

Test cases. The British Broadcasting Corporation (BBC) has a website that publishes local weather forecasts; each forecast is some 30K of HTML. After this was converted to XHTML, the prototype was used to transform it into a one-line textual summary suitable for being read by a speech synthesiser, sent as an SMS message, or being displayed in a login script or similar:



$\longrightarrow$

BBC Weather Centre. Tuesday: Sunny Intervals. Wednesday: Sunny Intervals. Thursday: Cloudy. Friday: Light Showers. Saturday: Sunny Intervals

Although the transformation is easily invalidated by changes in the BBC's house style, it does demonstrate the relative simplicity of the prototype's models:

```
<TABLE nomarkup="1">
  <TABLE children-only="1">
    <TD start-at="4">
      <TR start-at="2" end-at="7">
        <B before=". " after=": "/>
        <ALT/>
```

The prototype was also used to parse a MathML document (which was generated from LaTeX by the utility tex4ht), and to output it as English text. For example, the expression:

$$\sum_{n=0}^{k} \frac{f^n a}{n}$$

became "sigma from n equals 0 to k of f to the n a over n". Since tex4ht outputs MathML in "presentation" format and uses Unicode entities for such things as , it was necessary to extend the model language with a `value` attribute, so that entries like `<mo value="&#x2211;"> sigma </mo>` could be used to translate the symbols.

Conclusion

A system was developed that allows the user to specify the structure of the desired output in a relatively concise manner. The concept described can be used to prototype more quickly systems for presenting structured data in alternative formats for people with special needs. The system can be extended for use with a wider range of transformation tasks. It will probably be more useful when dealing with multi-dimensional data such as musical scores.

The method of inputting the desired structure is left open; it need not be XML. Future work may include allowing the user to describe the desired structure by creating an annotated example document. It might also be possible to develop a parser generator that takes similar specifications, so that there is greater conceptual similarity between parsing data and formatting it.

The prototype tends to generate large numbers of points in the fourspace, and this is not efficient in space or time, even though the points are stored in a data structure that allows for faster queries. There is scope for developing a more efficient model for use in realtime or embedded applications.

Acknowledgements

The first author is supported by a studentship from the UK Engineering and Physical Sciences Research Council.

References

Silas Brown. An extensible system for conversion of musical-notation data to braille musical notation. *Computing in Musicology*, 12:45-74, 2001. The original was an undergraduate dissertation entitled "A Representation and Conversion System for Musical Notation", Cambridge University Computer Laboratory, 2000.

World Wide Web Consortium. *XSL Transformations (XSLT) Version 1.0, W3C Recommendation*, Nov 1999. http://www.w3.org/TR/1999/REC-xslt-19991116.

World Wide Web Consortium. *Extensible Markup Language (XML) Version 1.0 (Second Edition)*, Oct 2000. http://www.w3c.org/TR/2000/REC-xml-20001006.

Office for Standards in Education. *Inspection Report - RNIB New College, Worcester*, page 37. Oct 2000. Inspection number 223644.

Peter Gregor and Alan F. Newell. An emperical investigation of ways in which some of the problems encountered by some dyslexics may be alleviated using computer techniques. In *Proceedings of the Fourth International ACM Conference on Assistive Technologies ASSETS 2000*, pages 85-91, Nov 2000.

Dirk Hermsdorf, Henrike Gappa, and Michael Pieper. Webadapter: A prototype of a WWW-browser with new special needs adaptations. In *Proceedings of the 4th ERCIM Workshop on 'User Interfaces for All'*, number 8 in Long Papers: WWW Browsers for All, page 15. ERCIM, 1998.

Julie A. Jacko, Max A. Dixon, Robert H. Rosa, Jr., Ingrid U. Scott, and Charles J. Pappas. Visual profiles: A critical component of universal access. In *Proceedings of ACM CHI 99 Conference on Human Factors in Computing Systems*, volume 1 of *Profiles, Notes, and Surfaces*, pages 330-337, 1999.

Julie A. Jacko and Andrew Sears. Designing interfaces for an overlooked user group: Considering the visual profiles of partially sighted users. In *Third Annual ACM Conference on Assistive Technologies*, pages 75-77, 1998.

Bill McCann. *GOODFEEL Braille Music Translator*. Dancing Dots Braille Music Technology, 1754 Quarry Lane, P.O. Box 927, Valley Forge, PA 19482, USA. http://www.dancingdots.com/.

David Taylor and Christopher Harris. About nystagmus. Technical report, Nystagmus Network, 108c Warner Road, Camberwell, London, SE5 9HQ, UK, Sep 1999. http://www.btinternet.com/~lynest/nystag.pdf.