XML FLATTENED: THE LESSONS TO BE LEARNT FROM XBRL

Lucian Holland, DecisionSoft Limited

Think of an XML document, and the chances are that you will be thinking of a series of nested tags neatly indented to show the structure of the data being presented. The average XBRL (eXtensible Business Reporting Language [XBRL]) document, however, is almost entirely flat - a series of data items all at the same level. All the information that would normally be encoded into the hierarchical structure of a standard XML document has been stripped out, and resides in an XBRL "taxonomy" that defines the concepts being used and their relationships to one another. These taxonomies make heavy use of [XLink] in combination with XML Schema to define fluid and extensible frameworks for reporting different classes of business data.

In this paper I first of all give a brief description of how these mechanisms work. I then go on to examine how and why XBRL takes an approach that initially seems so counter-intuitive, looking at the advantages of disadvantages of its separation of structure from content from a number of angles, particularly validation, information reuse and versioning. While it is undoubtedly true that there are some areas of the specification that are very specific to the primarily financial domain that it addresses, and others that have proved to be unexpectedly complex to implement and work with, I believe that some of the core ideas in XBRL offer some interesting insights into many problems faced by the wider XML community. For example, extending existing content models whilst maintaining backward compatibility is an area that has exercised many minds; through its use of an extensible graph structure rather than a traditional hierarchy, XBRL offers a novel and interesting solution. By drawing out positive lessons like this, and negative ones where XBRL has taken routes that led it into difficulties, this paper aims to be of interest both to those looking to work with XBRL itself, and to those developing and using similar technologies that may be able to take advantage of some of the techniques it is based on.

WHAT DOES XBRL DO?

Reporting, not transactions. XML standards for business data have tended to focus on web service models; XML has been used as a common transmission medium to enable broad interoperability of applications. But XML can also be employed in longer-lived applications with less clearly defined consumers. This parallels traditional markup as employed in web pages and publishing: information is made available in a standard, annotated form, and is then processed by a wide variety of consuming applications over an extended period of time. XBRL is designed to be this sort of XML format, and provides a generic means of recording business data in a structured fashion.

Write once, read many. XBRL is fundamentally a publishing medium. Authors of XBRL documents make them available without much knowledge about how they will be used, or by whom. A given report might be consumed by a whole range of different users ranging from internal managers to external analysts and regulators. All of these users might be using different applications that seek to use the data in widely differing ways. This is a very different use case to a message passed between two carefully defined service interfaces! In addition, many XBRL reports will be archived and used for historical analysis over a period of many years. So there is quite a powerful requirement for semantic stability.

A complex and changing problem domain. To those of us with no training in it, accountancy can seem like a pretty black art. Much is open to interpretation and re-interpretation, and the same underlying information can be presented in a wide variety of different ways. Furthermore, practices are constantly changing as businesses find new ways to present their finances in the most favourable light allowed by laws which are themselves updated to impose ever-increasing transparency. While the presentation of accounting data is one of the primary use cases for XBRL, and it is an important design goal to improve transparency, it would be unrealistic to expect a new technology to completely revise the operation of an established profession overnight! So XBRL has to be able to cope with modelling a highly complex domain without first reducing it to its simplest presentation.

A domain that is also highly regulated. For all its fluidity, the world of accountancy and business reporting is also highly regulated. In the wake of the accountancy scandals of the past few years, financial authorities around the world are imposing increasingly detailed and stringent reporting requirements on businesses. A technology designed to act as a medium for such reporting needs to be capable of high degree of semantic precision to ensure that it reflects the appropriate accountancy frameworks without introducing further ambiguity.

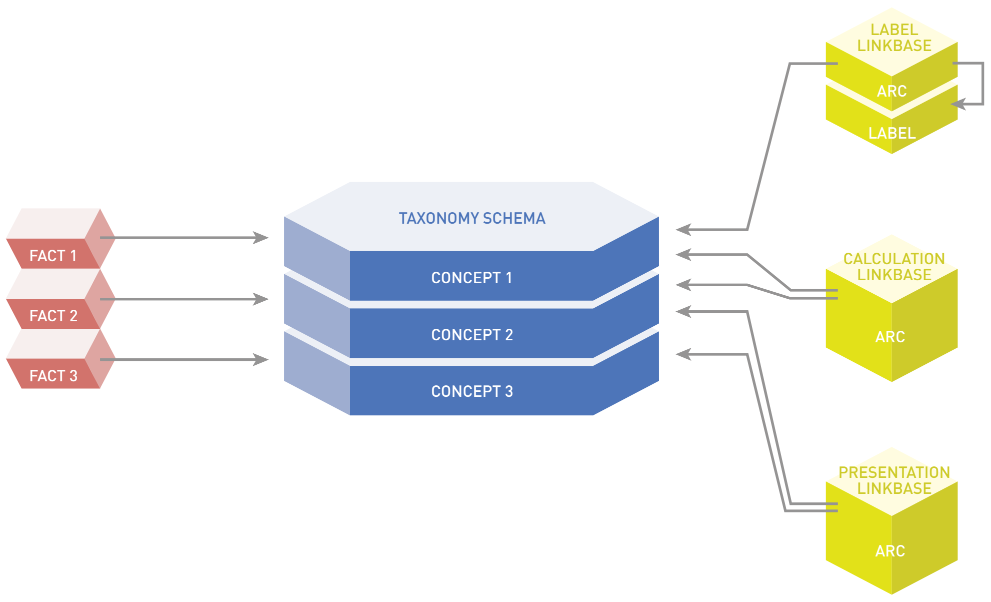

Figure 1.
A schematic view of the relationships between different XBRL documents.

HOW DOES IT DO IT?

Schema with minimal hierarchy. A Taxonomy Schema defines the content model of a corresponding XBRL instance. The elements that such a schema defines are known as Concepts and fall into two categories - Items and Tuples. An Item contains a single piece of data; with the exception of a special type designed to allow the representation of fractions, Items contain simple content. They must all derive from a common base type called itemType defined in one of the core XBRL schemas. By contrast, Tuples are used to group other concepts, and can thus contain only Items and other Tuples. Despite the availability of the Tuple construct, XBRL content models generally do not contain deep levels of nesting: for reasons that will be explained later in this paper, the generalisation implied by grouping items hierarchically in this way is only appropriate in a limited number of scenarios in the domains covered by XBRL.

Everything's global. A corollary of the limited hierarchy is the requirement that all concepts defined in an XBRL Taxonomy Schema must be defined globally. This results in a very different style of schema to that which most people are used to - an enormously long list of global element definitions, which often have long names reflecting their very precise level of applicability. As an example, the IFRS (International Financial Reporting Standards) taxonomy contains concepts with names like:
`AmountGainLossRecognisedFinancialAssetFairValueReliably MeasuredBeenOvercomeAssetBeenSold.`
Something of a mouthful! As a consequence, some taxonomy authors have already started to take the view that a Taxonomy Schema should not be treated (primarily, at any rate) as a human-readable document, and have used short codes as element names. In general, the overall effect of the construction of these schemas has the result that they are extremely easy to understand if you know exactly which concepts you are interested in, but immensely difficult to “browse”.

XBRL extensions to schema. XBRL takes advantage of the open content model of XML Schema to include a number of extra attributes that “decorate” the elements in the Taxonomy Schema itself. These include an attribute to indicate whether a monetary item should be considered a "debit" or a "credit" and an attribute to indicate whether an item should be used to report a fact valid at a single instant in time, or one that holds over an extended duration. These constitute an exception to the general rule in XBRL that semantic details about concepts are not expressed in the schema itself, but instead represented in the linkbases that accompany it.

Linkbases define relationships between concepts and resources.
A Linkbase is an [XLink] construct consisting of a series of `extended-type` links. Such links are made up of two sorts of definition. First, they define a number of end-points, which can be either resources local to the linkbase (such as a textual label), or URI (Universal Resource Identifier) references to external resources. In the case of XBRL, such external `locators` are generally used to point to concepts in the Taxonomy schema. Second, each link also defines a number of `arcs` that connect these endpoints together. In XBRL linkbases are used for two purposes. First they are used to associate metadata such as human-readable labels and references to relevant literature with the elements from the taxonomy schema. Second, they can be used to construct complex graph structures that express relationships between concepts.

Five standard linkbases. The XBRL specification defines five linkbases that provide core functionality:
Presentation Linkbase This defines a hierarchical organisation of the concepts in the taxonomy for display purposes.
Label Linkbase This defines labels that can be used to display concepts; each concept can have any number of labels associated with it for use in a variety of different circumstances. For example, a balance concept might have one label if positive, and another if negative.
Reference Linkbase This defines references to external sources of information such as documentation or authoritative literature.
Calculation Linkbase This defines calculation relationships between concepts. So `Total` might be defined as the sum of `Item1` and `Item2`.
Definition Linkbase This defines a miscellany of logical relationships between concepts, such as interdependency (i.e. the presence of a particular item requires the presence of another).

A powerful extensibility mechanism. Building on the basic [XLink] functionality, XBRL adds a mechanism for overriding arcs to allow taxonomies to be extended by adding new linkbases. This allows extension taxonomies to cancel the effects of relationships defined in base taxonomies; as will be discussed in more detail later in this paper, this is important for achieving maximum flexibility and extensibility.

WHY?

A counter-intuitive approach? A well constructed XML schema can be a complex document, but in general schemas are comprehensible to an appropriately knowledgeable reader. The same frequently cannot be said of an XBRL taxonomy. The removal of all structure from the schema and its placement in [XLink] linkbases makes for a schema that gives little idea of the sense of the data it is designed to constrain. The linkbases themselves are equally hard to read, precisely because the concepts they structure are in the schema! In short, without software designed to process it, XBRL is very difficult to work with. So why on earth would anyone want to use a format like this?

Requirements for flexibility. The answer to this question lies in the stringent requirements that I mentioned earlier. The "X" in XBRL is more than just a moniker stolen from its parent technology - XBRL strives to provide an unusual degree of flexibility. At the heart of this lies an interesting approach to a very general problem of data modelling in XML (and indeed data modelling more generally). What is the best way to define a format such that it captures data with the maximum possible precision, but at the same time can be extended/rearranged with the minimum impact on backwards compatibility? In the following sections I will compare three different approaches to this problem, ending with that chosen by XBRL.

GENERIC SELF-DESCRIBING ELEMENTS

The problem of static typing. This approach aims to ensure maximum flexibility and extensibility by dispensing with standard "static typing". By the term "static typing", I refer to the practice of using specific elements for specific purposes; these elements are defined in a schema complete with detailed content models. In general, the syntax and semantics of a given element can thus be understood simply be examining its name. Such an approach can be limiting, however, since it ties the semantics of a given element to some precise syntactic constraints. For example, lets say I have an element `Expenses`, whose content model is defined as a sequence of `PostageCosts`, `TravelExpenses` and `FoodExpenses`. The general semantics of the Expenses concept are quite clear and well defined - this is an element used for reporting business expenses; and applications looking for such information would always expect to be able to find them in an Expenses element. The details of what is reported therein, and how, are much more likely to change. In particular, I might want to introduce a new tag `OutOfOfficeExpenses` under which to group `TravelExpenses` and `FoodExpenses`. But to do this involves a radical change to the content model of the element, and would break backwards compatibility.

Using dynamic typing in XML. Many highly configurable applications get round this problem by moving to a greater degree of genericism. The example we looked at above could be restructured to be much less rigid. We might replace all of the different elements with a single one, `Expense`. This might have a "total" attribute, a "type" attribute and a content model consisting of zero or more child Expense tags.

```
<Expense type="TotalExpenses" total="400">
  <Expense type="PostageCosts" total="10"/>
  <Expense type="OutOfOffice" total="390">
    <Expense type="TravelExpenses" total="10"/>
    <Expense type="FoodExpenses" total="380"/>
  </Expense>
</Expense>
```

As you can see, it becomes a lot easier to extend the content model using this method. Naturally, processing applications will have to be coded reasonably defensively to ensure that they are able to cope with the extra flexibility, but at least the constraint on extensibility is now the capabilities of a particular version of the code rather than the definition of the data format itself. When a later version of the expense reporting structure is used with an earlier version of the software, the result should hopefully be a certain loss of detail rather than a complete failure to process.

Poor schema validation. The major disadvantage of such a system arises precisely because it does eschew the standard “static” typing system of XML, namely the use of discrete element names. Most popular XML validation technologies use element names as the primary means of locating content definitions against which to validate. By using a single element and substantially deregulating its content model, one is essentially bypassing most of the checking that grammar-based validation can offer. Of course, it is possible to use rule-based systems like Schematron to replace much of this lost capability, but this misses the point in two ways. First of all, Schematron doesn’t provide typing - it just asserts the truth of a series of statements; much of the power of grammar-based validation lies in its ability to impose a conceptual framework of types onto the supplied data. Secondly, simply adding back the validation in another form creates much the same problems for extensibility as we saw earlier. The fundamental issue here is that imposing detailed constraints on a content model immediately creates potential problems for extensibility, since it may be necessary to change that very model.

Reliant on correct high-level analysis. The other problem with this approach is that it only solves part of the problem. In our example, we analysed the different elements being used and decided that there was a generic class to which they all belonged, namely “Expenses”. But in a real domain, things are likely to be a lot more complicated than this, and, inevitably, some of the analysis will be wrong. And at that point, one is back to square one - one needs to change a content model that is built into the data format.

THE XML SCHEMA APPROACH

Using object orientation. The above approach is employed in many existing XML formats that require a high degree of flexibility, such as configuration files for complex application frameworks. Nevertheless, readers will no doubt have noted that it is a somewhat simplistic approach that takes no real advantage of objected oriented techniques such as polymorphism. A more sophisticated alternative can be achieved using type derivation and substitution groups in XML Schema. With a carefully constructed schema, it is possible to ensure that a data format is highly extensible whilst at the same time providing for complete and detailed validation of the elements it contains.

Under such a scheme, one would define a content model very similar to the one we’ve just seen: an Expense element that contains zero or more child Expense elements. But the Expense element would be `abstract` - one would never actually use the Expense element itself. Instead, one would use subtypes of the Expense element that were defined to be substitutable for it - so one could again return to an XML instance that looked something like this:

```
<TotalExpenses total=”400”>
   <PostageCosts total=”10”/>
   <OutOfOffice total=”390”>
      <TravelExpenses total=”10”/>
      <FoodExpenses total=”380”/>
   </OutOfOffice>
 </TotalExpenses>
```

The definitions of the specific elements could then be given with as loose or as rigid a content model as was deemed appropriate.

Greater semantic rigour. The primary advantage of such a solution is that it lets the data structure do a bit more of the work - a processing application is no longer responsible for ensuring that the basic format of the data is correct, since this can be handled at the level of the schema. At the same time, programs capable of working with the schema and the PSVI (Post Schema Validation Infoset) will have access to a wealth of information about the type relationships between the elements used. This is important because it gives them something to fall back on if they don't recognise a particular element - they can walk back up the type hierarchy until they find something that they *do* recognise.

Content model extensibility still problematic. Exploiting the schema typing system to the full goes a long way towards solving the problem of combining semantic precision with extensibility. Unfortunately there are still significant limits to the flexibility of what can be achieved. The type derivation rules in schema are complex and comparatively restrictive; an instance of a derived type must always be a valid instance of its base type, either without modification in the case of derivation by restriction, or after a straightforward truncation in the case of derivation by extension. Flexibility is achieved by postponing the specification of detailed syntactic constraints, rather than employing constraints that are inherently flexible in themselves [1].

HIERARCHIES: THE SOURCE OF THE PROBLEM?

The weakness of trees. At this point it may seem like there is something of a logical contradiction in my arguments; it is almost as though I am looking for a rigid format that is also flexible! In reality, the rigidity I'm looking for is of a different sort to the flexibility that I am trying to blend it with. Semantic rigidity, with precise names and fixed typing, is important to the kinds of applications I am considering; but *relational* rigidity is not. The way that one concept connects with another is typically expressed in terms of a hierarchy in XML. Often, this is the most direct and concise way to structure one's data. But in quite a number of cases, hierarchies can turn out to be unnecessarily constraining. In Object Oriented software design, many of the most powerful design patterns eschew simple inheritance hierarchies in favour of more complex and dynamic networks of associations between objects. An important reason for this is that tree structures based on strong parent-child relationships can be difficult to change; the impact of any alteration increases exponentially as one moves away from the leaves of a tree towards its root. As tree-like structures, XML documents suffer a similar problem - an alteration in the content model near to the root of a document can affect the interpretation of a huge portion of the document, even if it appears comparatively trivial.

Hierarchies provide context. The meaning of the `FoodExpenses` element changes subtly depending on whether it is used inside `OutOfOfficeExpenses` or `CorporateHospitality`. One can reuse the same structure at different points in the hierarchy, relying on context to distinguish them. This is especially important when the nature of that context is itself dependent on the data being expressed. A classic example of this is the concept of a record:

[1]Schema does provide the `redefine` mechanism, but even this permits only limited modification: types must be redefined with themselves as a base type.

```
<CustomerRecord>
   <Name>Joe Bloggs</Name>
   <TelephoneNo>123456789</TelephoneNo>
   <FaxNo>24681012</FaxNo>
</CustomerRecord>
<CustomerRecord>
   <Name>Jane Bloggs</Name>
   <TelephoneNo>987654321</TelephoneNo>
   <FaxNo>1357911</FaxNo>
</CustomerRecord>
```

Where one has an open-ended repeat of a single structure like this, hierarchies are vital; one certainly wouldn't want to work with something like this:

```
<JoeBloggsTelephoneNo>123456789</JoeBloggsTelephoneNo>
<JoeBloggsFaxNo>24681012</JoeBloggsFaxNo>
<JaneBloggsTelephoneNo>987654321</JaneBloggsTelephoneNo>
<JaneBloggsFaxNo>1357911</JaneBloggsFaxNo>
```

That's because `JoeBloggsTelephoneNo` could only ever have one value, so it's not a terribly useful element!

Context is only important in some situations. Removal of hierarchy doesn't always produce such unworkable results:

```
   <OutOfOfficeFoodExpenses>123456789</OutOfOfficeFoodExpenses>
   <CorporateHospitalityFoodExpenses>
24681012
   </CorporateHospitalityFoodExpenses>
```

In this example the elements remain *conceptual*: they are not tied to a particular item of data, and thus have applicability beyond the scope of a single document. On the other hand, they have been made much more specific, so they are likely to be unique across a much broader context: a single company creating a report for a single time period might list numerous `FoodExpenses` elements, but they're likely only to list a single `CorporateHospitalityFoodExpenses`. So what do we lose in this sort of case by removing the hierarchy? Well, for a start, we lose some self-description; having removed all the obvious structure and flattened everything onto one level, we have lost information about how one piece of information relates to another. There is nothing beyond our choice of verbose element names that makes clear the close relationship between `OutOfOfficeFoodExpenses` and `CorporateHospitalityFoodExpenses`, nor presumably, between these two and `OutOfOfficeExpenses` and `CorporateHospitalityExpenses`. A corollary of this is that in a large document, it can be hard to find the information one is looking for, because all of the options are presented on one level.

For the purposes of browsing, exploration, and presentation, then, a "flattened" XML structure has disadvantages. But is this really so important? Providing that one is not dealing with the sort of record-based xml that simply doesn't work without a hierarchy, I don't believe that the loss of structure from the data is as significant as it might at first seem.

Once the concepts are made sufficiently precise to compensate for the lack of context, one can accommodate all of the data that one was able to to with a hierarchical structure. And one can then re-express separately all of the lost relationships in terms of the *definitions* of the elements being used, rather than within the data itself. So we could record the fact that `OutOfOfficeFoodExpenses` is a subtype of `FoodExpenses`. Of course, this is exactly what XBRL does with linkbases.

Semantic precision and flexibility combined. By taking this approach, XBRL succeeds in decoupling information about *type* from information about *relationships*. This means that the data can be expressed with great semantic precision without tying it to a particular structure which may need to be changed or extended in the future. Inevitably, the achievement of this sort of careful balance involves trade-offs, assumptions and compromises; in the remainder of this paper, I will look more specifically at the strengths and weaknesses of XBRL in the three areas of extensibility, information reuse and validation.

EXTENSIBILITY

Easy to override and extend. It is easy to change the relationships between concepts in XBRL without breaking backwards compatibility. To pick up the example that we used earlier, changing the breakdown of one's expenses would be trivial. Hierarchical relationships for the purposes of presentation are expressed in a presentation linkbase.
The starting structure was a tree as shown below:

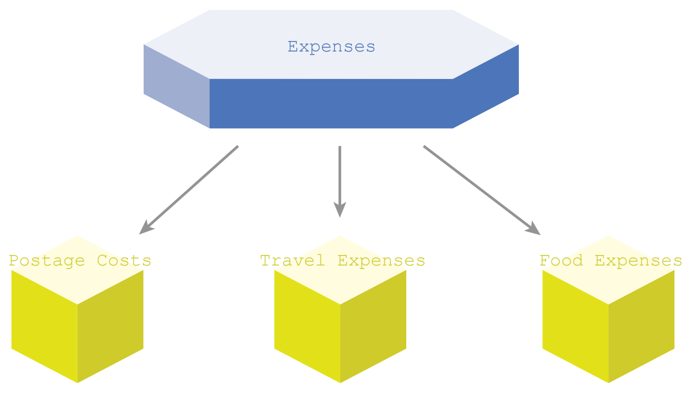

Figure 2.
Starting Expense Tree

Without changing the original linkbase, an extension linkbase could prohibit the relationships `Expenses-FoodExpenses` and `Expenses-TravelExpenses`. It could then define a relationship between `Expenses` and a new concept expressed in an extension schema, `OutOfOfficeExpenses`. Finally, the two relationships that were prohibited could be replaced with the relationships `OutOfOfficeExpenses-FoodExpenses` and `OutOfOffice-Expenses-TravelExpenses`. The result would be a new tree as shown below, where the dotted lines represent the old, prohibited relationships.

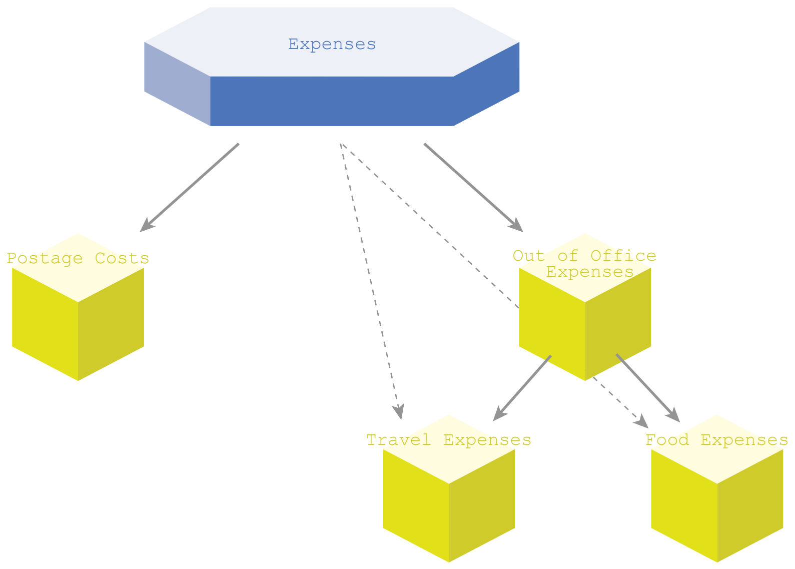

Figure 3.
Updated Expense Tree

A parallel extension could be made for the calculation linkbase, to make `OutOfOfficeExpenses` into a subtotal for the items it grouped. The important thing to note is that in the case of both the presentation and the calculation extensions, data that is valid according to the extension will remain valid according to the original taxonomy, because applications using the original taxonomy will simply ignore any value for `OutOfOfficeExpenses`.

Modularised extensions - original remains unchanged. The XBRL linkbase mechanism provides authors of extension taxonomies with extremely fine-grained control over the extension and redefinition of structures from the taxonomies they are based upon. It is true that something fairly similar can be achieved with `redefine` in XML Schema, but the XBRL system allows one to make changes at the level of individual relationships of specific types between specific concepts, rather than being forced to completely redefine a whole type in order to make a change; it also allows considerably more wide-reaching changes since it is not constrained by the rules of schema type derivation[2].

XBRL framework itself designed for extensibility. Factoring out different relational information into individual linkbases doesn't just make data formats defined with XBRL more flexible - it also makes XBRL itself more flexible. New linkbases can be added to capture information about new types of relationships between concepts without having any negative impact on existing applications and taxonomies.

EXTENSIBILITY - LIMITATIONS

Complex webs of documents. An area of the XBRL 2.1 specification that has proved quite problematic to define and implement has been the DTS (Discoverable Taxonomy Set). A side-effect of the power of the redefinition mechanism provided by XBRL is that a taxonomy that has been extended to any great extent can become a very large web of inter-referring XML documents. Processors must follow all of

[2]It is also worth noting that schema `redefine` alters components within their original namespace, which limits its usefulness as a robust extensibility mechanism.

the different types of links in all of these documents, to find the limits of the DTS, and then combine all of the data from the discovered documents into a single, consolidated taxonomy view. This is obviously a complex enough task for an application to perform, but it can be even harder for a human reader to follow[3].

XBRL taxonomies are verbose. People have come to accept that there is bandwidth price to pay for the transparency and flexibility of XML formats - markup tags are generally a verbose method of storing data. The actual data as reported in XBRL instances carries a comparatively light overhead by XML standards thanks to the lack of deep nesting, but this is only achieved at the expense of significant additional weight in the taxonomies that describe them. [XLink] is not a lightweight technology at the best of times, and when one has 5 separate linkbases all describing the same concept set, one tends to get very repetitive, verbose data structures; and this also worsens the problem of readability. It should be borne in mind, however, that in a wide variety of situations, the verbosity of the taxonomy never becomes an issue: those using it view it through the mediation of specialist software, and the large taxonomies themselves generally do not have to be transmitted as part of the submission or distribution of an XBRL report.

INFORMATION REUSE

Presentation neutral. An XBRL instance is a sequence of global elements, not laid out with any presuppositions about how it will be presented; it effectively acts as a straightforward set of named values. It is thus very easy to tailor the presentation of the data to the purpose at hand. By swapping in a new label linkbase, one can translate a report into another language; different presentation linkbases could draw on the same underlying data to produce a profit and loss report or a balance sheet. Since it contains just the core facts reported as individual, disconnected items, an XBRL instance can act as the basic data source for a very wide variety of different applications, representations and analyses.

Semantic stability. Another requirement that I identified at the start of this paper was for semantic stability of reported data over time. XBRL taxonomies satisfy this requirement because every item of data, whether completely independent, a specialisation of a more general type of item, or a sub-item in a particular category, always has its own unique element in which it is reported. Thus items in an XBRL instance do not depend for their interpretation on other items, providing some degree of insulation from change.

INFORMATION REUSE - LIMITATIONS.

Only applicable to some types of data structure As discussed earlier, as a solution to the problem of extensibility, the model used by XBRL only works well for some types of data. In particular, it is extremely poor at capturing transactional and record-based data which require lots of repeating structures to be meaningful[4].

[3]An added complication is the relationship of schemas discovered in the DTS to the set of grammars available for validating other documents in the DTS; the current XBRL specification does not discuss this matter in any detail.
[4]Naturally, such structures do appear to a limited degree even in documents which are predominantly report-based. To represent these, XBRL provides `Tuples`, which allow hierarchical structures to be created. Their use anywhere that they are not strictly necessary is discouraged, however, since it makes identifying the target of arcs in linkbases considerably more complicated, and can potentially diminish the expressive power of the linkbase constructs.

XBRL works best in fields like financial reporting where it is possible to identify precise, narrowly defined concepts that can be translated into global elements that repeat only for different reporting contexts[5] rather than within a single reporting context.

Requires specialist software. As will by now be apparent, XBRL's flexibility is achieved at the cost of a considerable increase in the complexity and size of the taxonomies. Apart from the immediate problems that this raises, it also effectively imposes the requirement for specialist software to do any significant processing of XBRL data, since traditional XML tools and technologies turn out to be of somewhat limited use. For example, while it is possible to parse an XBRL taxonomy into a series of DOM (Document Object Model) trees, this gives one very little leverage on the XBRL constructs, since it doesn't really simplify the task of aggregating all of the various links across the different linkbases. For this, one needs an [XLink] processor. [XLink] processors, however, will be unaware of the XML Schema significance of the concepts being linked; more importantly, the overriding prohibiting mechanism in XBRL is an extension to the standard [XLink] graph model, and will not be handled by a standard processor. When it comes to extracting and transforming XBRL data, one might be tempted to use XSL (eXtensible Stylesheet Language). But it doesn't take much time battling with multitudes of interdependent `xsl:keys` built from 5 different documents to realise that this approach is only really viable if one is prepared to ignore (or recreate in XSL) the information contained in the taxonomy itself[6] . Despite this, XBRL is still an XML format, and a heavily standards-based one at that; as a consequence it fits well into existing XML processing pipelines and is easy to integrate into standard web services.

VALIDATION

Modularised validation. As we have seen, XBRL factors out many of the details of constraints and relationships into a number of linkbases outside the core schema for a taxonomy. Furthermore, even within these linkbases, it is possible to partition and group the graph structures in completely configurable ways using the [XLink] mechanisms of `arcroles` and `roles`. With suitably flexible software, or with some fairly standard XML manipulation, it is possible to be highly selective about which relationships one wishes to make use of; this is obviously of particular interest in the field of validation, where the ability to perform different checks at different times can be extremely helpful both for those authoring reports and taxonomies, and those accepting submissions of XBRL data.

Fine-grained validation. The use of linkbases also makes it feasible to express data in very narrowly defined elements, since it removes the need to use common elements to express common purpose; instead one can simply link the relevant concepts appropriately to indicate their relationships to one another. An important consequence of this is that the validation constraints on individual items of data can be highly specific, thanks to their very narrow context of applicability.

[5]XBRL defines an element context which effectively allows reports to be partitioned by time or perspective (e.g. projected or actual); the most common example of a `context` is a particular time period, such as "The year ending April 2004". As can be imagined, there are many XBRL reports that only use one or two contexts, because they report factual information about a single time period.
[6]This is not to say it is not possible. In fact, DecisionSoft and others *have* produced XSL to process XBRL in a taxonomy aware fashion; it's just that having done so, we'd rather not do it again...

VALIDATION - LIMITATIONS

Constrictions of linkbases. The key to XBRL's flexibility lies in its use of [XLink] linkbases to express relationships between the concepts defined in the taxonomy schemas, but to do this in a graph structure independent of the schemas themselves. Unfortunately, this does impose some restrictions, since it means that the relationships being captured must be susceptible to modelling as a graph connecting the taxonomy concepts. For many purposes, such as providing a hierarchical presentation view, this causes no problems at all; but there are other situations where graphs become problematic. For example, while simple summations can easily be expressed as a tree of subtotals and contributing items, more involved calculations become increasingly complicated to express, particularly when they involve multiple ways of calculating the same total. More importantly, it becomes extremely difficult to determine whether a given network of calculation relationships expressed in a taxonomy is actually meaningful, or whether it ultimately simplifies to a contradiction of some form.

Overriding hard to control. Generally speaking, increased flexibility is a good thing; but inevitably there are situations in which one wishes to impose rigid controls on data rather than allowing a format to be extended to fit individual circumstances. Since XBRL encourages the use of very specific elements for reporting, and this strategy is made possibly to a large extent because of the ease with which taxonomies can be extended and specialised by individual users, these situations where tight control is required can become problematic. Disallowing extension taxonomies altogether is frequently not an option, since it may well prevent report authors from expressing all of the information they need to; but on the other hand, the enormous power of the XBRL overriding mechanisms mean that any extension could potentially rewrite the majority of the base taxonomy! This is a difficult area, but there are a number of possible solutions. The simplest is to validate the data against both the extended and non-extended versions of the taxonomy, to ensure that the extension is not redefining core relationships. Slightly more sophisticated would be to configure an XBRL validator to prevent overriding of certain specific types of relationship considered to be critical. Finally, at the most complex end of the spectrum, there is no reason why tools could not be written to check for the backwards compatibility of extensions in a more generic fashion.

CONCLUSIONS

XBRL takes an unusual approach to data modelling in XML that at first sight appears to be highly counter-intuitive. But if one looks beyond the complexity, I believe that this system is a valuable addition to one's XML toolbox. Nevertheless, it must be remembered that it is a tool. A good tool used for the right job can produce better results with less expenditure of effort; misuse it, however, and you may find yourself with a few less fingers than when you started! This is as true of XBRL's linkbase and schema combination as of any other tool.[7] As we have seen, the strength of this system lies in its ability to define a rich conceptual framework for recording data without requiring instances of that data to reflect the complexity of the framework in their structure. For this to work effectively, it must be possible (and sensible) to define the majority of the relationships that exist between items of data in abstraction from the actual data itself; in other words, one must be able to provide a complete and fixed description of the shape of the data independently

[7]Not literally, of course. There aren't yet any reports of bodily harm caused by XBRL.

of the data. Furthermore, XBRL is complex and heavyweight as XML technologies go - for it be worth using, the domain being modelled must be large enough to warrant the extra overhead. But if these criteria are met, the approach we have been examining has the potential to provide significant advantages in the areas of extensibility, semantic stability and validation.

BIBLIOGRAPHY

[XBRL] XBRL is the eXtensible Business Reporting Language. It is defined by XBRL International, a not-for-profit consortium of companies including many large accountancy firms. For more details see the XBRL International website.
http://www.xbrl.org

[XLink] XLink is an XML linking technology created by the World Wide Web Consortium. Details are available on their website.
http://www.w3.org/XML/Linking

[XML Schema] XML Schema has become the de facto standard for defining XML data formats. It is defined by the World Wide Web Consortium. Details are available from the schema homepage.
http://www.w3.org/XML/Schema