Storing, processing and transmitting linked chunks of structured text

Sindre Sørensen, Bergen, Norway

The current state of affairs

There is a vast amount of literature within computer science on how to create, how to process, using various algoritms, and how to transmit and data structures. This might be what computer science is all about. Nevertheless; creating, processing and transmitting data, such as nonlinear texts iusing XML is often not straightforward. Storing data structures in a linear or hierarchical form in an XML-document as well as validating and reconstructing data structures in memory from their serialised form is no easy task.

When data are produced in computer memory they are typically generated by a specially tailored application. The application may be specialised for assisting an author in creating structured texts, linear, hierarchical, or in other structures. Or to mention a completely different example, the data might be generated from environmental sensors, mapping values to a specific time etc. Or the data might be text typed by a human, using a tool to systematically reorganise an existing text, such as fragments from Wittgenstein's writings. Anyway, when we have an application that produces data structures in memory, we don´t have to worry about how the data are generated. Well written software would be able to natively handle any data structure, like sets, lists, trees, graphs or whatever is needed for the specific task. But the problems that I am trying to deal with in this paper arise when we want to store, share and transmit the data in a serialised form. Today, one of the standardised tools to store, transmit and retrieve text is XML. But XML does not by itself define how the structure of in-memory data structures are to be encoded out of their in-memory context. Document standard publishers, like The Text Encoding Initiative (TEI) and DocBook go one step further. They specify the semantics of the document and the structure of the final document, butstill confined by the hierarchical structure of XML.

The structure of XML documents is a tree:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<text>
  <text>
    Hello
      <text>
        world one!
      </text>
      <text>
        world two!
      </text>
  </text>
</text>
```

And because the inherent structure of XML is a tree, we can also use the inherent structure of XML to represent more general data structures, like lists and sets.

If all texts or all data were trees this would not be a problem. But I argue that this is not the case. A text might on one hand be considered an ordered list of a finite number of words. On the other extreme, the same text might be considered an intricate graph, where some elements repeat themselves; some elements overlap each other, elements point at each other unidirectionally or circularly. Consider a text talking about another text. It might be fruitful to both consider these two texts as two separate texts that together will form yet another text.

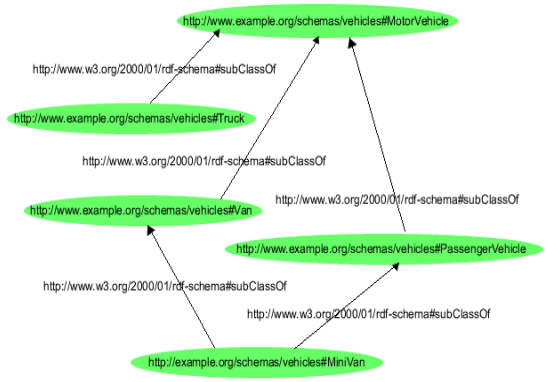
Encoding a text as a series of graphemes is easier: Just store it as a series of bytes in a file; a text file. Advancing to encode the text as an ordered list of words, that are contained in sentences, and thereafter in paragraphs etc, all in a hierarchical way would be solvable with for example XML.

But if the nature of the text or the data structure that we are trying to encode is not hierarchical we can not exploit the inherent structure of XML to encode our data structure. Still we can resort to a number of techniques to encode our data structure.

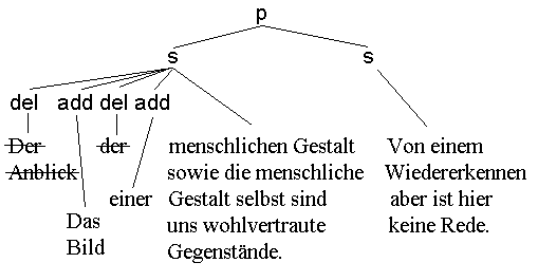
In standards for ontologies (in the computer science sense of ontologies) several such techniques are used. This following RDF/XML file is an example of this. The class "MiniVan" is a child of both "Van" and "PassengerVehicle". This makes the file describe a graph instead of a tree:

```
<?xml version="1.0"?>
<!DOCTYPE rdf:RDF [<!ENTITY xsd
"http://www.w3.org/2001/XMLSchema#">]>
<rdf:RDF
xmlns:rdf="http://www.w3.org/1999/02/2
2-rdf-syntax-ns#"
xmlns:rdfs="http://www.w3.org/2000/01/
rdf-schema#"
xml:base="http://example.org/schemas/v
ehicles">

<rdfs:Class rdf:ID="MotorVehicle"/>

<rdfs:Class rdf:ID="PassengerVehicle">
  <rdfs:subClassOf
rdf:resource="#MotorVehicle"/>
</rdfs:Class>

<rdfs:Class rdf:ID="Truck">
  <rdfs:subClassOf
rdf:resource="#MotorVehicle"/>
</rdfs:Class>

<rdfs:Class rdf:ID="Van">
  <rdfs:subClassOf
rdf:resource="#MotorVehicle"/>
</rdfs:Class>

<rdfs:Class rdf:ID="MiniVan">
  <rdfs:subClassOf
rdf:resource="#Van"/>
  <rdfs:subClassOf
rdf:resource="#PassengerVehicle"/>
</rdfs:Class>

</rdf:RDF>
```

The XML/RDF fragment above is from Manola, Miller, McBride 2004



The above illustration is a faximile from Manola, Miller, McBride 2004



The above illustration is a facsimile from McQueen and Huitfeldt, 2000

```
<p>
  <s>
    <del>Der Anblick</del>
    <add>Das Bild</add>
    <del>der</del>
    <add>einer</add>
    menschlichen Gestalt sowie die
    menschliche Gestalt selbst sind uns
    wohlvertraute Gegenst&auml;nde.
  </s>
  <s>Von einem Wiedererkennen aber ist
     hier keine Rede.
  </s>
</p>
```

The XML fragment above is an XML encoded fragment from a Wittgenstein text, repeated from McQueen and Huitfeldt, 2000 but reindented here.

Trying to remodel the data structure into a plain text form that is completely different than its in-memory form, like in the above examples, might not be necessary:

A proposal for a new way and new tools

As mentioned, computer science provides much literature on how to deal with various data structures. If we have the right application, the problem of how to produce our data might already be solved. We already have the data in computer memory. Could we just keep the data in memory, and not try to linearise it? I suggest that we could. Let´s say that our data structure is stored in a block of memory. This block of memory does not contain anything else but our data structure.

The following is a schematical and simplified summary on how this structure could be stored in memory. For simplicity I am pointing to sequential numbers where words are atoms here, while in a real world implementation we might want to point to memory addresses.

Atom number	Atom	
1	Der	
2	Anblick	
3	Das	
4	Bild	
5	der	
6	einer	
7	menschlichen	
8	Gestalt	
9	sowie	
10	die	
11	menschliche	
12	Gestalt	
13	selbst	
14	sind	
15	uns	
16	wohlvertraute	
17	Gegenstände	
18	.	
19	Von	
20	einem	
21	Wiedererkennen	
22	aber	
23	ist	
24	hier	
25	keine	
26	rede	
27	.	
28	* p	29,34
29	* s	30-33, 11-17
30	* del	1-2
31	* add	3-4
32	* del	5
33	* add	6
34	* s	19-27
35	*signature	36
36	alvhwl1hwf8qdvosdihf	

In this rendition, all words from the paper copy are repeated initially, while the structure comes after. This order is enforced here for simplicity and readability.

The serialised format would then be the sequence of bytes in this memory block. In addition we could add some extra features to the serialisation. These features would assist in validation, consistency checking etc. I will now briefly describe some conceivable features:

Digital signatures, and authorship control

Digitally signing chunks of data would provide several benefits:

The authorship of the text can then be verified. In fact, the text could have one or more authors, each of whom could add their signature. In addition, the software could provide a signature of its own, to link the version and the exact build of the software to the text. In this way, one could identify candidate texts for scrutiny when software bugs etc. are discovered at a later point in time.

When another author wants to add to the work in the table above, the data structure could be loaded into a virtual machine. To preserve the original work, and also the signature, the software should allow modifying the structure without requiring the original work to be modified.

One of the current ways to verify the origin of an electronic document is verifying its physical origin. In case the document was retrieved from the internet, the server's IP number might be checked. If we trust that the server belongs to an institution or author that we trust, we will also trust that we have the correct document. When the document is signed, we might not need to check the origin of the document. Instead we can subject the text to harder scrutiny; through signature validation.

A side effect of having a digital signature is that it does not matter anymore from where we get the data, if we have access to a signature that we trust. This principle is used in peer-to-peer protocols like bittorrent (using hashes):

> In order to keep track of which peers have what, BitTorrent cuts files into pieces of fixed size, typically a quarter megabyte. Each downloader reports to all of its peers what pieces it has. To verify data integrity, the SHA1 hashes of all the pieces are included in the .torrent file, and peers don't report that they have a piece until they've checked the hash (Cohen, 2003)

Well-formedness checking

For simplicity, we here assume that all our data structures are intact in memory, i.e. that all pointers point to the correct place in memory and that all data structures are consistent in memory. Our software then gives the text a signature. Let's assume that we have a signature mechanism that verifies that only one exact and unmodified version of a software package may have stored the data structure. Let´s also assume that we trust this software package to provide well-formed data. I argue that in this case signature checking may replace well-formedness checking. We may even trust the software that made the signature as much as, or even more than our locally running software. Using XML we would have had to parse the file, check for well-formedness and validity. Here we could potentially just load the file into memory, bit by bit, to reproduce the data structure that was in machine A into machine B.

Validation

I have now described a way to avoid restructuring, linearisation and parsing of a text. An important part of an XML workflow is validation. An external document, such as a DTD, a schema or some other mechanism is used to verify that a text is valid according to a set of rules.

As mentioned, in the system proposed here, signing might remove the need to validate data more than once. But we might in many cases still want a method to restrict the structure of content. For XML we have various solutions, like DTDs, XML Schema and RELAX NG. These are all well documented standards enabling us to define document types, and thereby validate instances to check that they are proper instances of the document type that is referred to.

I suggest that using the system proposed here we could store the rules needed for validating a document type in a similar way to the way that the document instances are stored. In principle we could store all data structures known to computer science in memory. One way to restrict this and to define document types could be to store a graph that contains all possible relations. I.e. the document definition graph could contain information on global document traits for our specific document type, such as whether the document must satisfy the criterions for being a list, a tree or a graph, or maybe a forest of graphs. In addition it could contain information about whether elements are allowed to have relations, and which relations each element would be allowed to have.

In-place markup versus stand-off markup

There has been a long debate on whether in-place or stand-off markup is the best mean to mark up text.

At the moment the in-place proponents seem to have grabbed the longest straw. XML and its relatives HTML and SGML are all basically in-place. When one needs to talk about something outside of the new text, three are several solutions:

In the system I am proposing here, we inherit a little bit from both of these worlds. When creating a new text, we might start from scratch, and the markup is actually a part of the new work, not something external to it.

A brick wall principle

What happens when we want to publish new comments and link them to an existing text? Presumably we can do this in a stand-off kind of way, where we do not touch the existing data. Instead we will point to places in the original data, at fragments of the original text etc. When we want changes to the original structure, we will form a new structure, but we will do it outside while pointing into the original text. In this way the new text depends on the existence of the original text, while the original text still exists as its own entity.

Machine independency

When the Java language was conceived one of the main ideas was that programs should be able to run on any hardware. This was achieved by specifying the compiled version of programs to be run in a virtual machine. The compiled code would then run on any system that implements such a virtual machine. For the system that is pro-

posed here I suggest that a similar technique would be used. But we don't have to worry about

An end user scenario, a brief walk through of a possible web publishing scenario

A researcher on Wittgenstein's philosophy would like to digitise a text written by Wittgenstein. After the text is digitised, the researcher would like to publish it, and make it available to other researchers for them to correct any errors, to discuss, make their own interpretations and comment on textual and philosophical issues, and to link places in the text to other texts. Researchers should also be able to make their own versions of the digitised text, where a common version can not be agreed upon.

The text is digitised in a specialised text editor, which allows for marking deletions, additions, corrections and margin notes. User friendly tools to do these kinds of digitisation should be available without having to resort to editing the machine readable encoding itself. The text is then published on a web site, where anyone comment In both the content and the structure of the text by adding extensions that point into existing work.

Conclusion

Stand-off markup and most of the ideas presented here are of course not a new idea. But hopefully the combination of tools presented here would be worth a test implementation.

Literature

Cohen, Bram, 2003 *Incentives Build Robustness in BitTorrent,* (http://www.bittorrent.org/bittorrentecon.pdf).

Manola, Miller, McBride, 2004 *RDF Primer, W3C Recommendation*, (http://www.w3.org/TR/REC-rdf-syntax/)

McQueen, Huitfeldt 2000 *GODDAG: A Data Structure for Overlapping Hierarchies*.

Email: sindre.sorensen@uib.no