# *Proceedings of Extreme Markup Languages®*



## Making CONCUR work

*Mirco Hilbert*
*Oliver Schonefeld*
*Andreas Witt*

### Abstract

The SGML feature CONCUR allowed for a document to be simultaneously marked up in multiple conflicting hierarchical tagsets but validated and interpreted in one tagset at a time. Alas, CONCUR was rarely implemented, and XML does not address the problem of conflicting hierarchies at all. The MuLaX document syntax is a non-XML syntax that enables multiply-encoded hierarchies by distinguishing different “layers” in the hierarchy by adding a layer ID as a prefix to the element names. The IDs tie all the elements in a single hierarchy together in an “annotation layer”. Extraction of a single annotation layer results in a well-formed XML document, and each annotation layer may be associated with an XML schema. The MuLaX processing model works on the nodes of one annotation layer at a time through Xpath-like navigation. CONCUR lives!

**Keywords:** Concurrent Markup/Overlap

### Table of Contents



### Mirco Hilbert

Mirco Hilbert studied Computer Science and Language Technology at Bielefeld University. He is now a member of the research unit for Applied and Computational Linguistics at the Justus-Liebig-University

Gießen. Parts of the paper deal with aspects of his Master's Thesis.

**Oliver Schonefeld**

Oliver Schonefeld studies Computer Science and Language Technology at Bielefeld University. Parts of the paper deal with aspects of his Master's Thesis.

**Andreas Witt**

Since 1996, Andreas Witt has taught at Bielefeld University, Germany in the field of 'text technology'. His research interests include the combination of computational linguistics and markup technologies, schema languages, and corpus annotation.

XML Source | PDF (for print) | Author Package | Typeset PDF

# Making CONCUR work

*Mirco Hilbert [Justus-Liebig-University Gießen]*
*Oliver Schonefeld [Bielefeld University]*
*Andreas Witt [Bielefeld University]*

**Extreme Markup Languages 2005® (Montréal, Québec)**



***Note:*** This paper contains W3C MathML, which is not equally well supported in all browsers. If you have reason to think that mathematical expressions are not displaying properly, consult the PDF version (or try a different browser).

## Introduction

This paper describes ways to approach the functionality of the SGML-feature CONCUR. This work is based on two Master's Theses. The origin of these theses was a paper originally given at Extreme Markup Languages 2004 **[Witt (2005)]** , where it was argued that the redundant encoding of information in multiple forms, as described by the TEI-Guidelines (see **[ACH/ACL/ALLC (1994)]** and **[Barnard et al. (1995)]** ), has a lot of advantages over other methods of encoding multiply structured text.

The multiple encoding of text allows to use all the available techniques and software products for XML documents. The MuLaX-Format was developed as an integrated format for editing. This format is strongly influenced by the SGML option CONCUR. The XML-conformance is achieved by the processing model: The processing is conservative because the concurrent annotations are kept separately. In other words, a non XML-syntax as a sequentialization of multiple incompatible hierarchies is used on the one hand, and on the other hand, all the processing is done on (hierarchical/XML conform) trees.

## Multiple Hierarchies and CONCUR

Some papers presented at Extreme 2004 described several possibilities of encoding multiple hierarchies with markup languages (e.g. **[DeRose (2004)]** ). **[Witt (2005)]** argued that - from a practical point of view - the technique of stand-off annotation is the best way for representating and storing documents with markup, which is used for annotating information from different levels. This allows to separate the different annotation layers. But it was also argued that stand-off annotation has several disadvantages:

- The layers, although separate, depend on each other. They can only be interpreted by reference to the layer(s) they point to.
- Although all information is included, the information is difficult to access using generic methods. As a consequence, standard parsing or editing software cannot be employed.
- Standard document grammars (e.g. the TEI Relax NG scheme, the XHTML-DTD, or the W3C Schema for DocBook) can only be used for levels containing both, markup and textual data.
- Linking to a sub-element range, or to textual data not annotated at all is difficult. The pointing mechanism defined by the TEI or by XPointer can be used, but requires another special software solution.

All of these points are related to the process of document exchange because stand-off annotation requires special purpose software. This contradicts the vision of a more sustainable storage of marked-up documents. Therefore, it was argued that multi-hierarchically marked up documents should be stored in different, separate, self contained XML-documents. It has been shown that if the textual data are exactly the same in all of the separate documents, the very text can serve as an (implicit) link. So all of the documents build a coherent unit.

As a simple example we use the following short dialogue between Peter and Paul, shown in Figure 1.

**Figure 1**

*Peter:* Hey Paul! Would you give me

*Paul:* the hammer?

Assuming the uttered texts as our primary data we can annotate the dialogue structure and the sentences of this dialogue in two separate primary data identical XML documents, shown in Figures 2 and 3.

**Figure 2: Separate annotation of the dialogue structure of the example dialogue 1**

```
<?xml version="1.0"?>
<!DOCTYPE div SYSTEM "tei/dtd/teispok2.dtd">
<div type="dialog" org="uniform">
   <u who="Peter">
     Hey Paul!
     Would you give me
   </u>
   <u who="Paul">
     the hammer?
   </u>
</div>
```

**Figure 3: Separate annotation of the sentence text structure of the example dialogue 1**

```
<?xml version="1.0"?>
<!DOCTYPE text SYSTEM "tei/dtd/teiana2.dtd">
<text>
   <s>Hey Paul!</s>
   <s>Would you give me
   the hammer?</s>
</text>
```

In this example the hierarchical structures are overlapping in that respect that Paul in his utterance completes the sentence begun by Peter.

Formally, a collection of implicitly linked documents with the same textual data can be transformed in a data structure, best described as multiple trees which share the same leaves. In other words, several trees

span over the same text. Such a structure is sometimes called a multi-rooted tree.

However, for the creation and for an integrated processing of such document collections, a theoretical approach and software solutions are necessary. [Witt et al. 2005] show a way to process and to unify these documents. As a technique for editing such documents, an old SGML-technique can be applied: The optional SGML-feature CONCUR YES.

The SGML standard [ISO 8879:1986] provides the optional feature CONCUR to annotate **concur**rent hierarchical structures in one SGML document. Therefore, it allows to use more than one DTD in one document at the same time. The elements are assigned to the DTD they belong to by preceding them with the corresponding document type surrounded by round brackets, the so-called document type specification. Elements that are equally defined in all applied DTDs can be used as shared elements without any prefix.

One possible SGML CONCUR representation that combines both of the above annotations in one document is shown in Figure 4.

**Figure 4: The representation of the dialogue and the sentence text structure of the example dialogue 1 as a SGML CONCUR document.**

```
<!DOCTYPE div SYSTEM "tei/dtd/teispok2.dtd">
<!DOCTYPE text SYSTEM "tei/dtd/teiana2.dtd">
<(div)div type="dialog" org="uniform">
  <(text)text>
     <(div)u who="Peter">
       <(text)s>Hey Paul!</(text)s>
       <(text)s>Would you give me
     </(div)u>
     <(div)u who="Paul">
       the hammer?</(text)s>
     </(div)u>
  </(text)text>
</(div)div>
```

As [DeRose (2004)] already pointed out, in CONCUR the several marked up hierarchies are completely independent of each other and there is no possibility to constrain relationships across the applied DTDs. Thus, when parsing CONCUR documents it is either possible to parse (and validate) each hierarchy separately ignoring elements of other DTDs, or to parse them simultaneously, for example using a multiple stack approach to check the wellformedness on each layer.

This independence of the marked up hierarchies has also some interesting side effects, as described in [Witt (2002)] . The order of two (or more) tags at one position of the primary data is arbitrary. Even if two (or more) elements of different DTDs span the same region, the order of their start and end tags is irrelevant as in the following example:

```
<(div)u><(text)s>Here you are.</(text)s></(div)u>

<(text)s><(div)u>Here you are.</(div)u></(text)s>

<(div)u><(text)s>Here you are.</(div)u></(text)s>

<(text)s><(div)u>Here you are.</(text)s></(div)u>
```

Because of these and other disadvantages or problems Charles Goldfarb, the main developer of the SGML standard, advised against using the CONCUR option already in 1990:

*I therefore recommend that CONCUR not be used to create multiple logical views of a document, such as verse-oriented and speech-oriented views of poetry.* ( [Goldfarb (1990)] )

Others do not share Goldfarb's critical position. Steven De Rose mentions that:

*The main advantages of CONCUR are that it is part of SGML, and that it is quite legible and maintainable.* ( **[DeRose (2004)]** )

Sperberg-McQueen and Huitfeldt conlude an comparison of CONCUR and MECS by stating:

*By using concur, the simplicity of interpretational rules found in MECS (and in the basic SGML model) can be combined with a powerful language for expressing constraints on document structures (the DTD). The theoretical and practical advantages outweigh the practical disadvantages and the humanities computing community should begin serious experimentation concur.* ( **[Sperberg-McQueen and Huitfeldt (1998)]** )

# The MuLaX Document Syntax

As mentioned above, the proposed solution dealing with multi-hierarchically structured documents is to store each hierarchy in a separate XML document. The documents differ only in the markup (including the values of attributes), the text content in all of these is identical. However, the main problem of this approach is to keep these multiple documents consistent when editing the redundant stored primary data or the annotations of one of the documents.

Thus, the purpose of this paper is not to propose yet another markup language to store and handle multi-hierarchically structured data, trying to solve the overlapping problem. [1] MuLaX, standing for **Mu**lti-**La**yered **X**ML, is intended as a document format which merges a collection of several primary data identical XML annotated documents into one document in a uniform way, which can easily be viewed and edited by a human user. Accordingly, MuLaX appears as a temporary processing and storage format of a source code editor which loads and stores sets of primary data identical XML documents. It is assumed that annotators using this editor are familiar with XML and with using XML source code editors. Therefore, not only the look and feel of the editor should be aligned to common XML editors, but also the document syntax should be as easy to understand as XML for the human reader.

## MuLaX and CONCUR

The MuLaX document syntax takes the syntax of SGML CONCUR as an example and is derived of it. In the same way it has its origin in the syntax of XML and thus it is subjected to some restrictions in contrast to SGML CONCUR. In order to distinguish the different annotation layers every tag is marked with a layer ID as a prefix to the element names and refers to the annotation layer which it belongs to.

As an example an annotation of the dialogue and the sentence layer of the dialogue in Figure 1 using the MuLaX document syntax is given in Figure 5. The similarity with CONCUR is unmistakable (see Figure 4).

**Figure 5: The representation of the dialogue and the sentence text structure of the example dialogue 1 as a MuLaX document.**

```
<?mlx version="1.0" encoding="iso-8859-1"?>
<!DOCTYPE (1)div SYSTEM "tei/dtd/teispok2.dtd">
<!DOCTYPE (2)text SYSTEM "tei/dtd/teiana2.dtd">
<(1)div type="dialog" org="uniform">
  <(2)text>
    <(1)u who="Peter">
      <(2)s>Hey Paul!</(2)s>
      <(2)s>Would you give me
    </(1)u>
    <(1)u who="Paul">
      the hammer?</(2)s>
    </(1)u>
  </(2)text>
```

```
</(1)div>
```

In order to describe the MuLaX document syntax we will use the terminology defined by the XML Recommendation as much as possible. In addition we define the following terms:

| | |
|---|---|
| Information Layer [2] | An information layer is a term which is used more conceptually. It refers to a special view on a document together with accessory meta information that can be represented by structure information. |
| Annotation Layer | In contrast to an information layer an annotation layer is a technical term which refers to the concrete syntactical realization of an information layer by annotations in a document. Here it is possible that one annotation layer represents more than one information layer at the same time. |

If a MuLaX document is viewed as a set of interwoven XML documents one annotation layer corresponds to exactly one XML document.

| | |
|---|---|
| Annotation Schema | An annotation schema defines elements, their relations and their possible attributes that can be used on one annotation layer. An annotation schema can be *implicit*ly existent or *explicit*ly declared. Implicit annotation schemes simply exist by the used vocabulary of elements and attributes. In XML it is for example possible to generate a DTD for one or a set of XML documents by the usage of DTD generators. Thus, an implicit annotation schema becomes an explicit one.<br><br>An explicit annotation schema is declared by an annotation schema declaration at the top of a MuLaX document. The advantage of an explicit annotation schema is that the correct usage of annotations is provable by a validation process. |

Which schema language is used for the annotation schema is not exactly regulated by the MuLaX document syntax. Thus, all common schema languages are possible, e.g. DTDs, XML Schema, and Relax NG.

| | |
|---|---|
| Annotation Schema Declaration | An annotation schema declaration declares the explicit annotation schema that belongs to one annotation layer. At the same time, it defines the annotation layer ID with which the elements of its layer refer to their corresponding schema declaration. In the current version the concrete syntax is not established for all possible schema languages. However, for DTDs or DTD fragments the following syntax is proposed to declare the annotation schema by a customized DOCTYPE declaration:<br><br>`<!DOCTYPE (1)div SYSTEM "dtd/teispok2.dtd">`<br><br>Here the annotation layer with the ID 1 and the root element `div` is assigned to the annotation schema that is stored in the local file dtd/teispok2.dtd. Of course, PUBLIC identifiers are possible as well as internal DTDs and DTD subsets as in XML.<br><br>If there is no existing annotation schema declaration for one annotation layer, the annotation schema is called "implicit" as mentioned above. |

The inversion of the above mentioned merging view can now be formulated as a mandatory requirement for every MuLaX document:

| | |
|---|---|
| Wellformedness Requirement | Every projection of a well-formed MuLaX document onto one annotation layer results in a well-formed XML document. |

Therefore, a *projection* onto an annotation layer is defined by the following process:

1. Delete all annotations that do not belong to the own annotation layer, i.e. all tags with foreign annotation layer ID prefixes, so called foreign tags, as well as possible existing foreign annotation schema declarations.
2. Delete all annotation layer ID prefixes from all annotations of the own layer as well as the annotation layer ID prefix declaration from a possible existing annotation schema declaration.

In our example document (Figure 5) the dialogue information layer is represented by the annotation layer with the ID 1, the sentence layer takes the ID 2. Deleting for example all tags with the layer ID prefix `(1)` it is imaginable that only the tags with the layer ID prefix `(2)` are left. We get a MuLaX document with only one annotation layer. If we now delete these prefixes from the remaining tags we get a well-formed XML document. Of course, the MuLaX declaration (in line 1) also must be changed to an equivalent XML declaration.

The wellformedness requirement implies and imputes some further features of MuLaX documents. In SGML CONCUR it is possible to use unmarked elements side by side with elements that are marked by a document type specification. If there are elements defined with the same name in all used annotation schemes these elements can be used without a document type specification in SGML CONCUR documents, and so they are interpreted as belonging to all annotation layers. The MuLaX document syntax is stricter in this respect. Here every element must be marked with an annotation layer ID prefix, which clearly assigns it to one annotation layer. Moreover, each annotation layer must have its own root element. So root elements may not be shared by more than one layer as it is possible in SGML CONCUR.

This restriction is an advantage for the processing model to be developed. Because unmarked elements must either be multiply represented by an object for each annotation layer which also has to be synchronized. Or they are represented by exactly one object, which increases the complexity of the model because a shared element -- unless it is the root element -- has more than one parent element. Therefore, by this restriction more simple processing models are possible for MuLaX documents.

Attributes are generally not marked with annotation layer ID prefixes, since they always belong to an element. [3] Thus,

```
<div type="dialog">...</div>
```

and

```
<div rend="bold">...</div>
```

will be merged to

```
<(1)div type="dialog"><(2)div rend="bold">...</(2)div></(1)div>
```

and not

```
<div (1)type="dialog" (2)rend="bold">...</div>
```

The latter would also be at odds with the stipulated absence of shared elements.

Comments and processing instructions are not yet provided by merging the XML documents. However,

they equally could be annotated using annotation layer ID prefixes. The syntax we would propose is

```
<!--(lid) no comment -->
```

and

```
<?(lid)php echo "Hello world!" ?>
```

respectively, while `lid` is the annotation layer ID.

Unlike SGML, in MuLaX the annotation layer declaration is optional. Hence, it is possible not to declare the annotations of one or more layers. Moreover, it is also possible that their layer ID prefix has no schema that it belongs to. MuLaX inherits this feature from XML. Since in MuLaX annotation layers with or without a corresponding annotation schema can co-exist, two kinds of validity are defined: (1) the validity on one annotation layer and (2) the universal validity.

The validity on one annotation layer is reduced to the validity in XML as follows:

Definition 1
: A MuLaX document is called *valid on one annotation layer* iff there exists an explicit annotation schema for this layer **and** the projection of the MuLaX document onto this annotation layer is a valid XML document.

Universal validity is defined as follows:

Definition 2
: A MuLaX document is called *universally valid* iff it is valid on every annotation layer.

For universally valid MuLaX documents it is especially guaranteed that a suitable annotation schema declaration for each annotation layer exists.

## MuLaX and Namespaces [4]

At first sight the concept of annotation layers in MuLaX and namespaces in XML seem to be very similar. Both will distinguish and mark elements which belong to different annotation schemes. But since documents, which use markup from different namespaces are still hierarchically structured XML documents, the problem of overlapping structures persists.

One reason why we use the CONCUR syntax with parenthesised layer IDs to mark the different annotation layers instead of an extended namespace approach is, that it should even be possible to merge several XML documents, which may contain elements of different namespaces, to one MuLaX document. Although this seems to contradict the information layer metaphor, some annotators may have plausible reasons to mix annotations from different namespaces in one annotation layer and it would be an unnecessary limitation if the MuLaX document syntax would forbid it. An approach automatically splitting up XML documents using different namespaces may result in artificial annotation layers, disturbing the semantically motivated information layer view. Additionally, an artificially generated root element needs to be added in most cases, which has to be removed when restoring the original XML documents.

A further main difference between annotation layers and namespaces is the possibility of multiply used annotation schemes. Namespaces are always assigned to a definite namespace name, which, at least conceptually, refers to an annotation schema or tagset where the elements are defined. In XML a namespace name can only be used by exactly one namespace prefix. By way of contrast, in MuLaX an annotation schema can be used by more than one annotation layer. Thus, alternative annotations using the same tagset are possible. [5] Also using different DTD fragments, declared in the same DTD, i.e. using one TEI subset for different views or purposes.

# The MuLaX Processing Models

The description of the processing models is mainly based on a model developed and described in [Hilbert (2005)] . Moreover, aspects of the features and the characteristics of an alternative model are described.

## Features of the Processing Model

The MuLaX processing model defines a hierarchical view on the MuLaX document based on one annotation layer. Thus, a MuLaX document with $n$ annotation layers corresponds to $n$ instances of the MuLaX processing model because for each annotation layer one model instance is needed. However, for our purpose of the processing model, a user is acting only on one annotation layer at a time. Therefore, only one model instance is needed at a time and the expenditure of synchronizing $n$ model instances for each MuLaX document is dropped.

This commitment results in the MuLaX processing model having a hierarchical data structure which is analogue to the XML data structure of the projection of the actual annotation layer. In the MuLaX processing model we therefore distinguish between the *current annotation layer* and the so-called *foreign annotation layer*s. The current annotation layer as the preferred layer in the MuLaX model stipulates the hierarchical structure of the MuLaX model. Their elements are mapped onto elements in the MuLaX model. In contrast the elements of the foreign annotation layers are treated as milestone elements and are handled as child elements of the actual surrounding element of the current annotation layer.

Herein lies the primary difference and thus the substantial advantage of the MuLaX model compared to milestones or other fragmenting techniques. With those a commitment to a primary annotation layer was to be made already at the stage of designing a shared annotation schema and deciding on the specific document structure, while in the MuLaX document syntax all annotation layers are treated equally. Hence, it is possible not to select the currently interesting annotation layer before the stage of representing the document by a processing model instance and to change the view on the document arbitrarily. Also the author of a MuLaX document does not have the imposition to keep track of the correct and consistent artificial linking of element fragments or milestones, or of constructing virtual elements using ID reference techniques.

In some respects the processing model follows the ideas of the Just-In-Time-Trees (JITTs, [Durusau & Brook O'Donnell (2002a)] , [Durusau & Brook O'Donnell (2002b)] ).

### Structure of the Processing Model

With a basic idea of the concept of the MuLaX processing model we now want to look at its concrete structure. Similar to the DOM the MuLaX processing model uses the node metaphor. Aside the MuLaX model node, which forms the root of a MuLaX model instance, there are element nodes, attribute nodes, text nodes and so-called foreign tag nodes. Comment and processing instruction nodes are not represented in the current version of the model.

Except for the MuLaX model node every node has exactly one parent node which is either an element node or the model node. Above and beyond that all nodes in a MuLaX model but the text nodes belong to an annotation layer.

The MuLaX model has as its root exactly one *MuLaXModel* node which represents the whole MuLaX document from the view point of the current annotation layer. It refers to the current annotation layer and possesses a list of child nodes, which can consist of element nodes and foreign tag nodes. Since each annotation layer has to have its own root element according to the MuLaX document syntax only one element node is allowed in the list of child nodes which indeed may be surrounded by foreign tag nodes.

This can easily be understood considering our example MuLaX document. In the MuLaX model instance

with the dialogue layer (annotation layer 1) as the current annotation layer the `div`-element is the only child of the MuLaXModel node. If we treat the sentence layer (annotation layer 2) as the current one, the root element `text` is enclosed between the start and the end tag of a foreign layer. We will look at the entire model representation of our example document later in more detail.

Text nodes are not permitted as child nodes of the MuLaXModel node, since these would be outside of the root element node of the current annotation layer and thus also outside of the root element of the projection of the MuLaX document onto this annotation layer. Thus, either this projection document would not be a well-formed XML document or the demanded characteristic of primary data identity between the individual annotation layers would be violated.

Asides from the above mentioned parent node a *MuLaXElement* node possesses a possibly empty list of child nodes. Like its parent node an element node refers to the current annotation layer. The list of child nodes again can contain further element nodes, text nodes, or foreign tag nodes. If the list is empty the element node represents an empty element. Additionally, a MuLaXElement node has a name, which corresponds to the generic identifier of the element, and a possibly empty list of attribute nodes. Similar to the XML information set, the DOM, and other XML models, attribute nodes are not treated as child nodes of an element node.

A *MuLaXAttribute* node possesses an element node as its parent node and refers to the current annotation layer, too, whose reference it inherits from its parent node. Besides, it has a name and a value. The name of the attribute node corresponds to the attribute name and the value to the value of the attribute, which obeys the same restrictions as XML attribute values. Attribute types are not handled in this basic version of the MuLaX model. The information missing for this purpose is not available until evaluating the annotation schema and, thus, it should be supplemented in later versions of the MuLaX model.

A *MuLaXText* node is the only type of node which does not belong to any annotation layer. If required it can be assigned to an artificial layer 0 in a concrete implementation of the model. The parent node of a text node is always its superordinate element node of the current annotation layer. Thus, by changing the current layer the parent node of each text node is changed. The value of a MuLaXText node contains the appropriate primary data. Here as in the DOM no directly adjacent text nodes are allowed. Therefore, there is at least the delimiter of an element node (a tag) or a foreign tag node between two text nodes.

In analogy to the kinds of tags in XML there are three different kinds of *MuLaXForeignTag* nodes: the *MuLaXForeignStartTag* node, the *MuLaXForeignEndTag* node, and the *MuLaXForeignEmptyTag* node. A foreign tag node always refers to its associated foreign annotation layer. Since a tag of a foreign annotation layer is treated like a milestone element, a foreign tag node does not have any child nodes. Possibly existing attributes in foreign start or empty tags are not represented in the current version of the model. These can be supplemented in a later version if necessary.

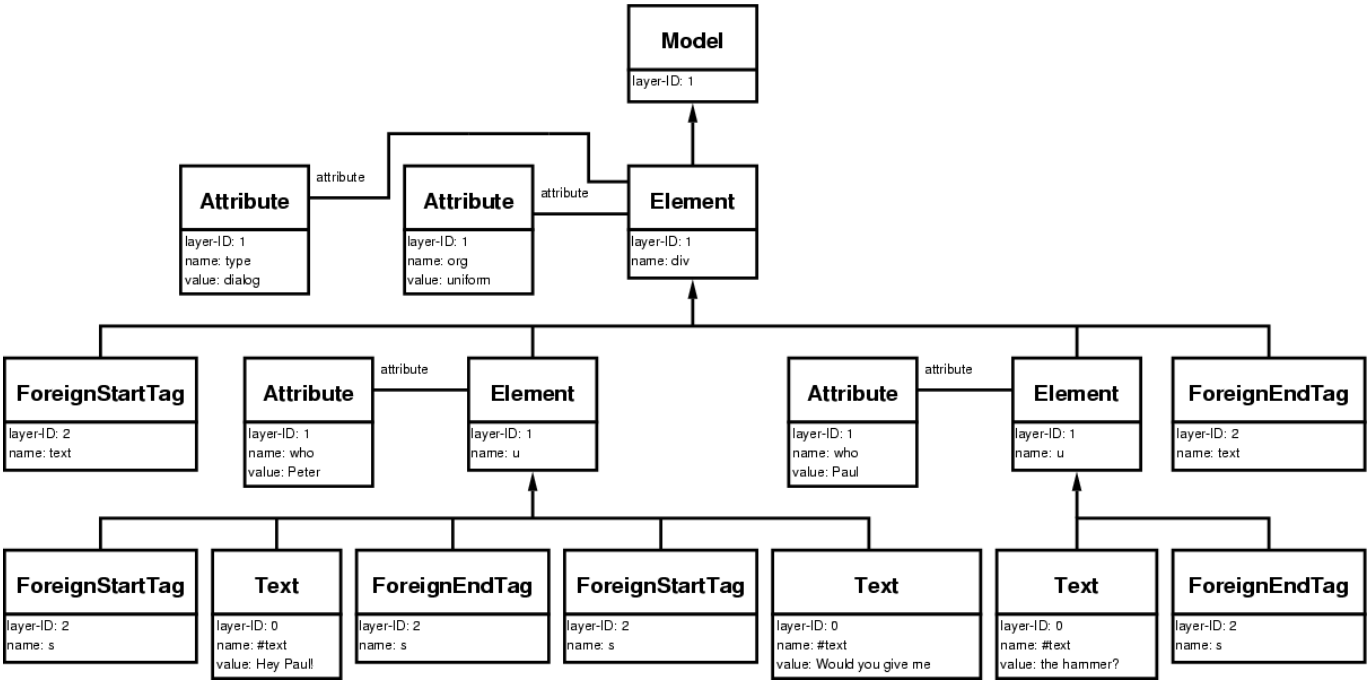
In order to represent the interrelation of a foreign start tag and a foreign end tag node these can optionally be provided with a reference to its corresponding foreign tag node. This reference is optional for the reason, since it is not demanded from a MuLaX parser to implement this feature. However, without any reference between foreign start and foreign end tag nodes, these may occur indiscriminately in principle anywhere in the model without any connection to each other. This means that there is no possibility to check whether the elements of a foreign layer are properly nested, even whether there is a corresponding foreign end tag to each foreign start tag at all. If a MuLaX parser implements these cross references between corresponding foreign tag nodes, its operating expense is much greater, indeed, but it thereby has the possibility to check the well-formedness on all annotation layers as well as to indicate to the user the relationship between foreign tags selected in a MuLaX editor.

**Example**

We would now like to look at our example document given above in Figure 5 in more detail. This consists of two annotation layers and can be represented by two different MuLaX models depending on which

view on the document one has and which annotation layer one selects as the current one.

**Figure 6: MuLaX model for annotation layer 1 of the example dialogue. To order to simplify matters, the parent and child relations are displayed as arrows in direction to the parent nodes**

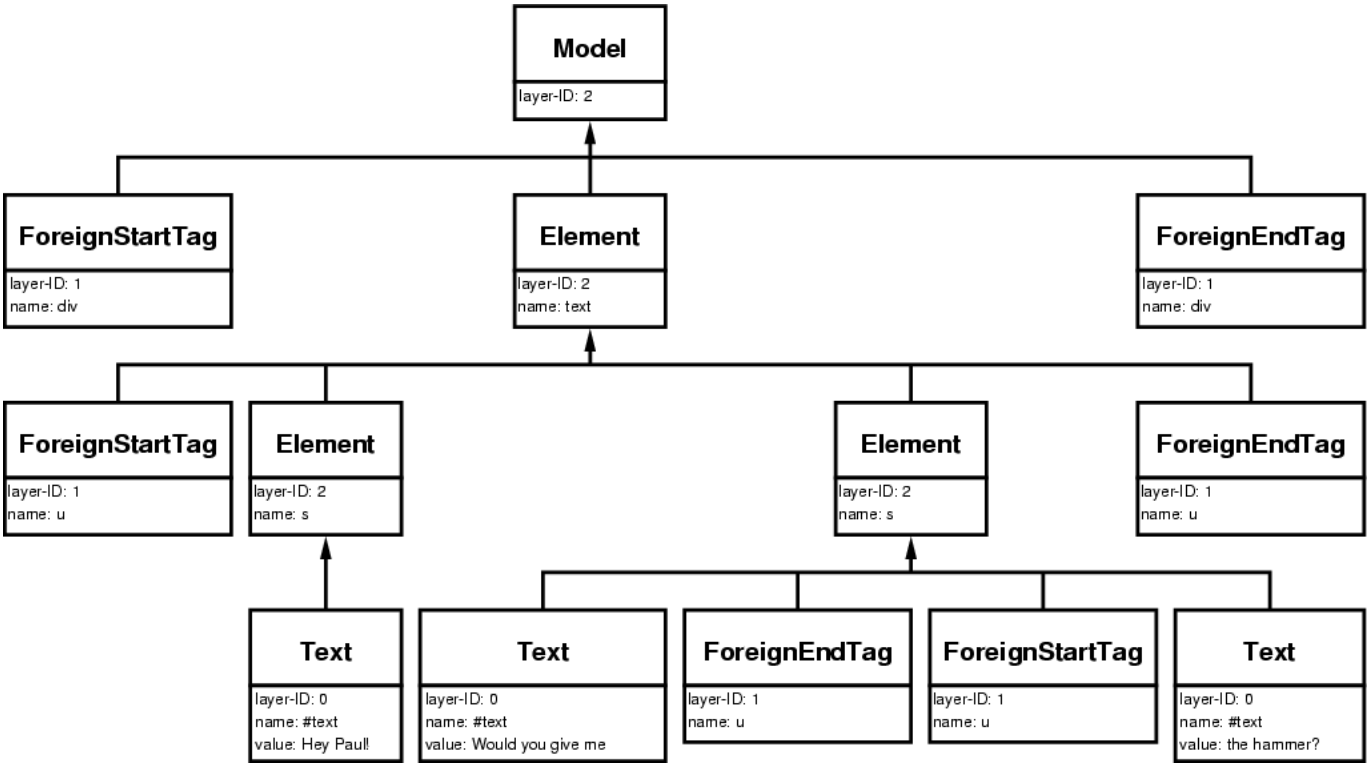



**[Link to open this graphic in a separate page]**

In the Figure 6 the MuLaX model is displayed from the view of annotation layer 1, the dialogue layer of the example dialogue. Every element of the current annotation layer, in this case the elements with the names `div` and `u`, are represented as MuLaXElement nodes. The tags `<(2)text>` and `<(2)s>` are "degraded" to MuLaXForeignStartTag nodes, `</(2)text>` and `</(2)s>` accordingly to MuLaXForeignEndTag nodes. These are understood in each case as child nodes of the actually surrounding element node of the current annotation layer. The text nodes are also in a child relationship to the element nodes of the current layer.

Figure 7 displays the MuLaX model based on annotation layer 2, the sentence layer.

**Figure 7: MuLaX model for annotation layer 2 of the example dialogue. In order to simplify matters, the parent and child relations are displayed as arrows in direction to the parent nodes.**




**[Link to open this graphic in a separate page]**

Here the elements `text` and `s` of annotation layer 2 are represented as MuLaXElement nodes. All other tags of the foreign annotation layer 1 are represented as appropriate foreign tag nodes and are subordinated to their surrounding element nodes of the current layer. Since in this case the root element of the current annotation layer is not the outermost element of the MuLaX document, it is not the only child node of the MuLaXModel node, but it is surrounded by the two foreign tag nodes of the root element of the foreign annotation layer. Furthermore, it becomes obvious here that corresponding foreign tag nodes do not always have to be on the same level of the MuLaX model tree. For example, for the first `u` element of the foreign annotation layer its ForeignStartTag node is subordinated to the `text` element on the second level of the tree, but its corresponding ForeignEndTag node is one level below subordinated to the second `s` element. For the second `u` element it is the other way round.

The relations between the corresponding foreign tag nodes, optionally representable in the MuLaX model, are not displayed in the two figures. How these are realized in a concrete implementation is a matter of design and therefore it is not explicitly specified by the MuLaX model. Extended path expressions similar to XPath are conceivable, for example. Thus, the ForeignStartTag node of the first `u` element then has a property `correspEndTagNode` with the value `/(2)text/(2)s/(1)u` and its corresponding ForeignEndTagNode has the property `correspStartTagNode` with the value `/(2)text/(1)u`. These two path expressions are unambiguous, since in the first case the first `s` element does not have a `(1)u` child node and below the second `s` element there is exactly one `(1)u` node, which is a foreign tag node at the same time. If some nodes may be not clearly locatable, the path expressions can be stated more precisely by augmenting the node names with indices [6], which indicate the exact position of the node in the current branch of the tree, as it is even customary in XPath. In the first case the path expression can be made more precisely: `/(2)text[1]/(2)s[2]/(1)u[1]`. Since foreign tag nodes can only occur as leaf nodes in a MuLaX model tree, the layer ID prefixes may also be omitted up to the latter.

It is important to note that these path expressions are not an instrument to give unique indications about the position of an element within a MuLaX document. They always depend on the current selected annotation layer and thus on the concrete MuLaX model instance. For example, the first `u` element of annotation layer 1 is accessible through the path expression `/(1)div/(1)u[1]` in the MuLaX model of annotation layer 1. But in the MuLaX model of annotation layer 2 it is represented by two foreign tag nodes with the path expressions `/(2)text/(1)u[1]` and `/(2)text/(2)s/(1)u[1]`.

A further possibility to express the relation between two foreign tag nodes in the MuLaX model is to augment each ForeignTag node with an automatically generated ID and to use an ID reference mechanism. In the concrete implementation of the MuLaX model in Java, object references are used as a further possibility.

## Multi-rooted Trees: An Alternative Model

Currently, an alternative data model for processing MuLaX documents is being developed. In contrast to the other approach a document is represented as a linear structure of element and text items. Additionally, a hierarchical DOM-like node structure will exist for each annotation layer. The other model is only capable of holding one hierarchical structure for all annotation layers using elements and foreign elements. If one wants to focus on a different annotation layer as primary layer a "structural recalculation" is needed. This is exactly what happens when multi-hierarchically structured data are processed by an JITTs-like approach. The new approach tries to avoid this computationally expensive operation. This can be achieved, since the hierarchical information for all layers is part of the model and will be kept in sync with the linear representation.

The datamodel implementation provides a function for giving access to the hierarchical structure of each annotation layer (e.g. doc->GetRootForAnnotationLayer(1)). The hierarchical structure will be implemented as a subset of the DOM-API **[DOM]** specified by the W3C and will allow to traverse and manipulate the structure.

### Structure of the Processing Model

As described above, the model consists of two parts: a linear structure and a hierarchical structure for each annotation layer. We distinguish between the terms *item* and *node*. Items are entities used in the linear part of the model, nodes are part of the hierarchical part.

The linear structure is an array of *text*, *start tag*, *end tag* and *empty tag* items. Each item stores an *offset* and a *length*. The *offset* is the offset in bytes from the beginning of the annotated document. The *length* is the length in bytes of the character sequence for a given item. The array storing the items is ordered by the item's offset. Additionally, element items may also store attributes in an array of attribute items.

The hierarchical structure is a tree structure of *element* and *text* nodes. An element node is connected to it's start and end item in the linear representation. For empty elements the start and end item are the same. Text nodes are connected to at least one text item. They may be connected to more text items, because a different annotation layer may slice a text item of another annotation layer:

```
<!-- ... -->
<(1)u who="Peter">
  <(2)s>Would you give me
</(1)u>
<(1)u who="Paul">
  the hammer?</(2)s>
</(1)u>
<!-- ... -->
```

The `s` element in annotation layer 2 is sliced by the two `u` elements in layer 1. In this case, the `s` node in tree structure would be connected to the two text segments "Would you give me" and the "the hammer?".

## Implementation

### Editor a

One example implementation of the MuLaX model shows how it is qualified as a fundament to develop a powerful editor for multiple XML-structured data. Hereby, the text editor infrastructure, which is provided

by the Eclipse framework [Object Technology International, Inc. (2003)] , is used to realize the editor as an Eclipse plug-in in Java. Thus, the MuLaX model is implemented by an object oriented approach.

By extending the abstract class AbstractDecoratedTextEditor, which Eclipse provides in the package org.eclipse.ui.texteditor, all basic features of common source code editors are already available. The plug-in structure of the editor can be divided into the following packages:

mulaxe.editors
: The package "editors" implements the essential structure of the editor and registers and configures its implemented editor characteristics.

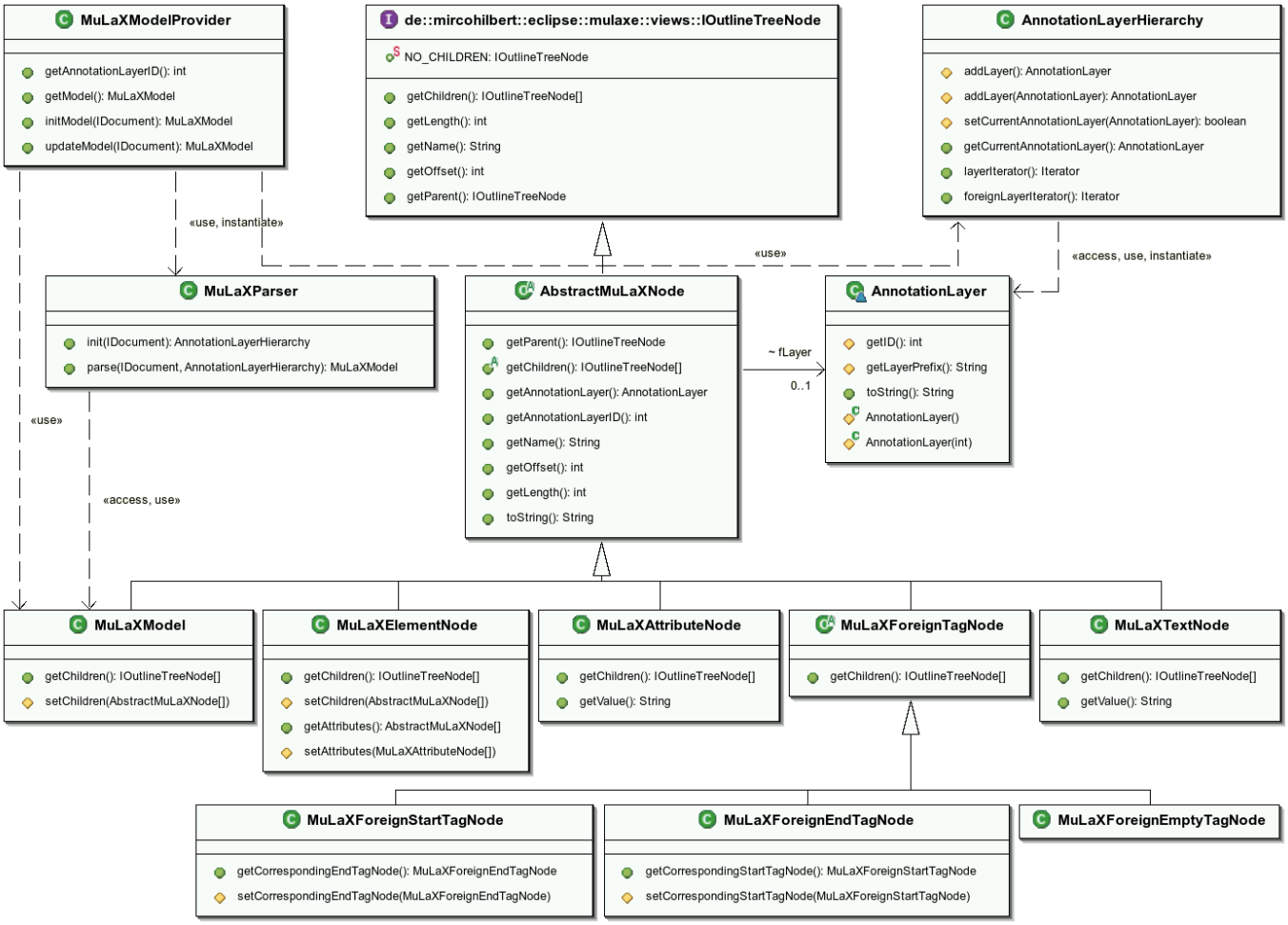
mulaxe.actions
: The package "actions" implements the application specific user actions to work with the editor, e.g. a function changing the current annotation layer view.

mulaxe.views
: The package "views" contains the implementation of an interactive document structure overview, where the tree structure of the current selected MuLaX model instance is shown in the so-called OutlinePage. The outline tree nodes are bound to the corresponding source text segments shown in the editor and can be used to navigate in the MuLaX document.

mulaxe.mulax
: The packages "mulax" implements the MuLaX processing model and, therefore, can be seen as the core of the MuLaX editor.

In Figure 8 the class hierarchy and associations of the classes are shown as a UML diagram. The class AbstractMuLaXNode implements the basic structure of a MuLaX tree node with all attributes and methods all node types have in common. The MuLaX model instance for the current annotation layer is constructed by a MuLaXParser object, which is called by the MuLaXModelProvider, which operates as an interface that provides the desired model instances for the current MuLaX document.

**Figure 8: The UML class and association diagram of the Java package mulaxe.mulax which includes the implementation of the MuLaX processing model. The object attributes are hidden for lack of space.**



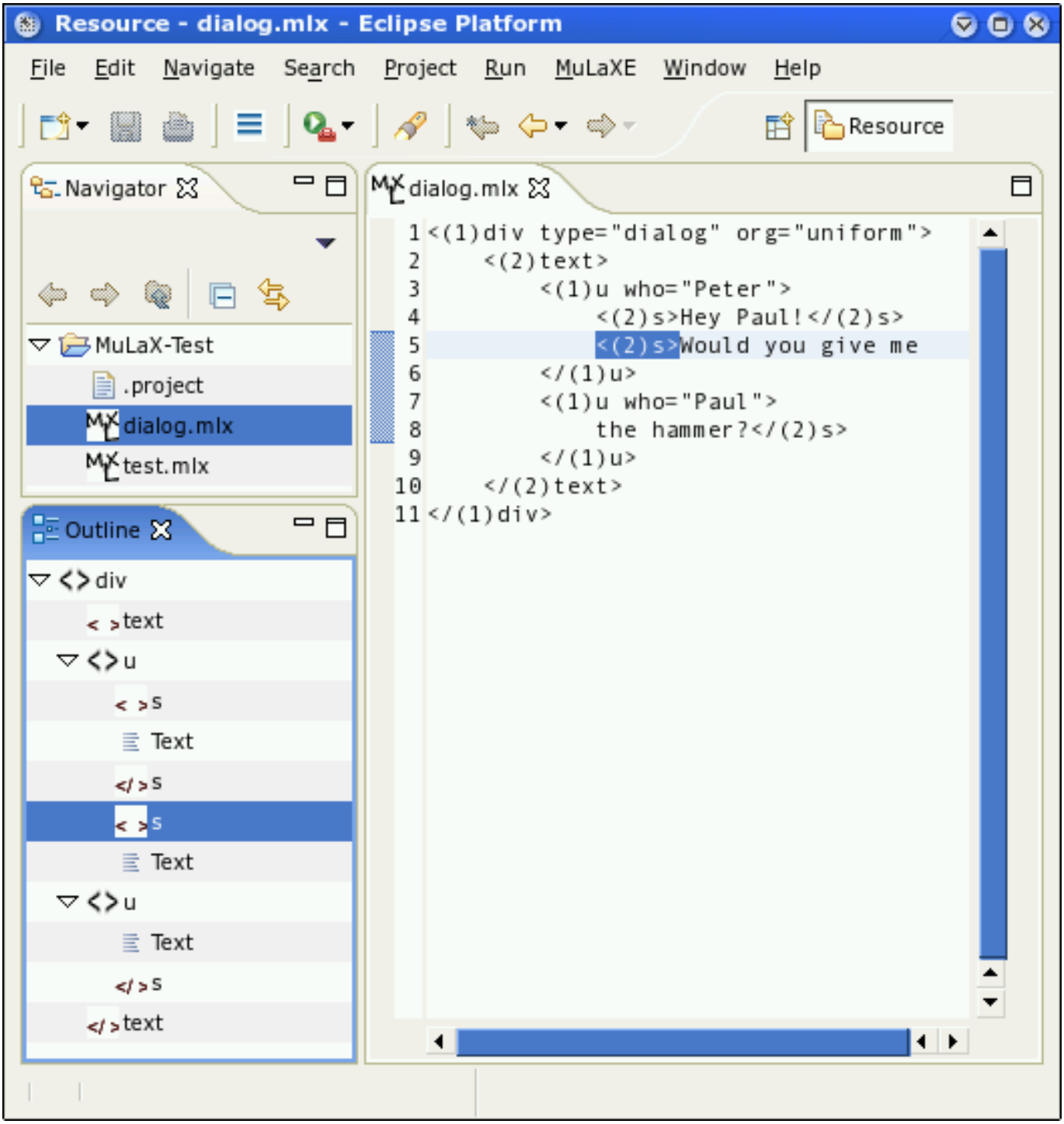

**[Link to open this graphic in a separate page]**

As the wellformedness check and the validation of a MuLaX document can easily be done using native XML techniques, while acting on projection documents as described in "The MuLaX Document Syntax". The main task of the MuLaX parser is to construct a hierarchical view of the current annotation layer to be shown in the OutlinePage.

Thus, the parsing strategy is similar to parsing XML using a simple push down automaton approach which cooperates with a tokenizer which will differentiate between current and foreign tags. Current ones are used as in XML to detect the boundary of one element node, and foreign ones are simply subordinated as foreign tag nodes to the surrounding current element node. If the correlation of corresponding foreign tags should be detected by the parser, the parsing strategy can be extended using not only one pushdown stack for the current layer but one for each layer. Thus, the different layers can be parsed simultanously.

The following Figures 9 and 10 show the current basic version of the editor in action. In the first figure the dialogue layer (layer 1) is selected as the current one (compare the structure view at the left hand side with the MuLaX model instance in Figure 6). By selecting the second `s` foreign start tag in the outline page, in the editor the range to its corresponding end tag is shown by the blue bar at the left border of the editor frame. Additionally, the corresponding text region is marked.

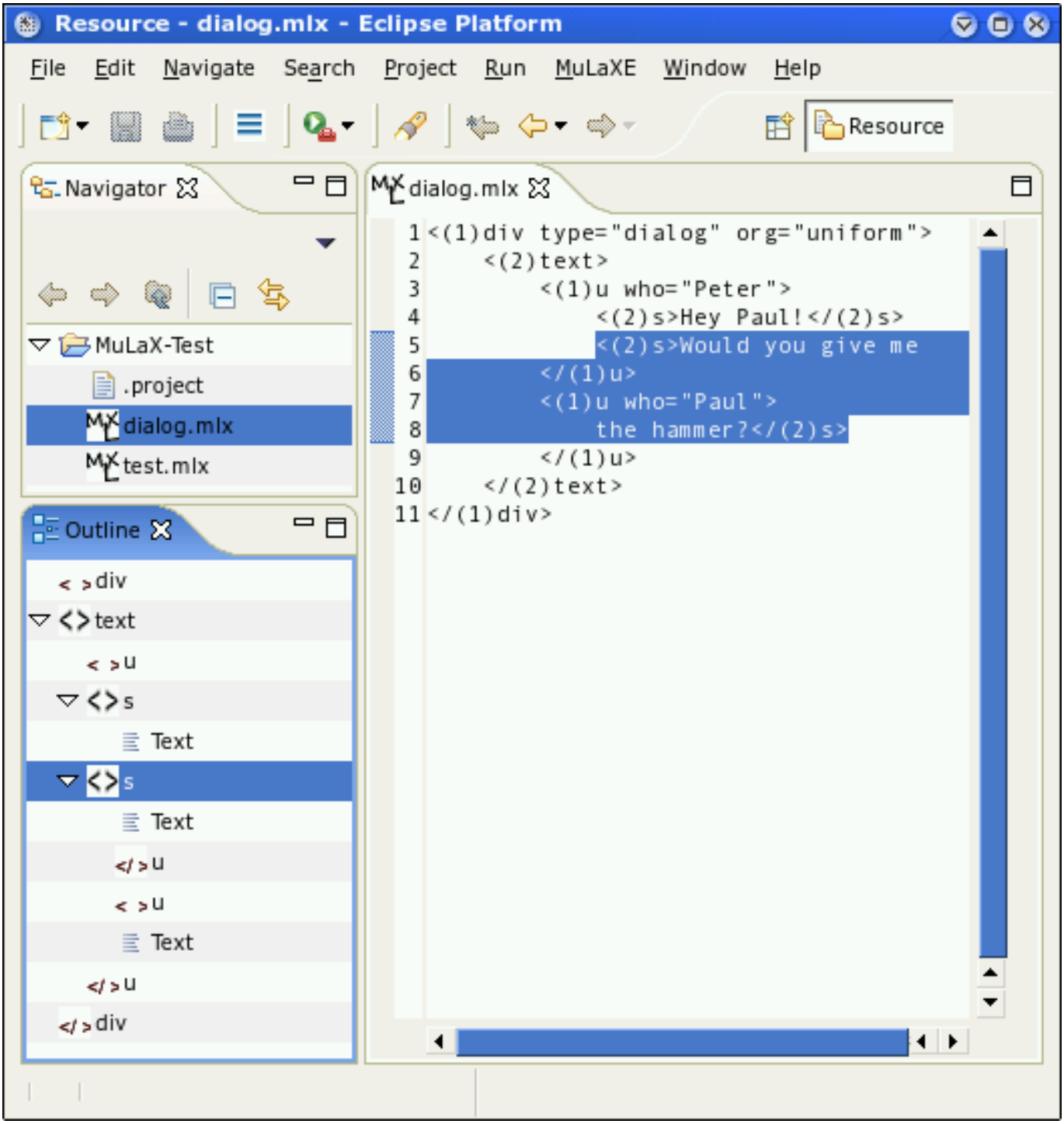
**Figure 9: A screenshot of the MuLaX editor MuLaXE. In this figure the view is displayed if annotation layer 1 is selected as the current one.**




**[Link to open this graphic in a separate page]**

In the second figure the sentence layer (layer 2) is the current one (here the structure view corresponds to the model instance displayed in Figure 7). Here, the whole element region is marked by selecting the `s` element in the outline page.

**Figure 10: A screenshot of the MuLaX editor MuLaXE. In this figure the view is displayed if annotation layer 2 is selected as the current one.**




**[Link to open this graphic in a separate page]**

### Editor b

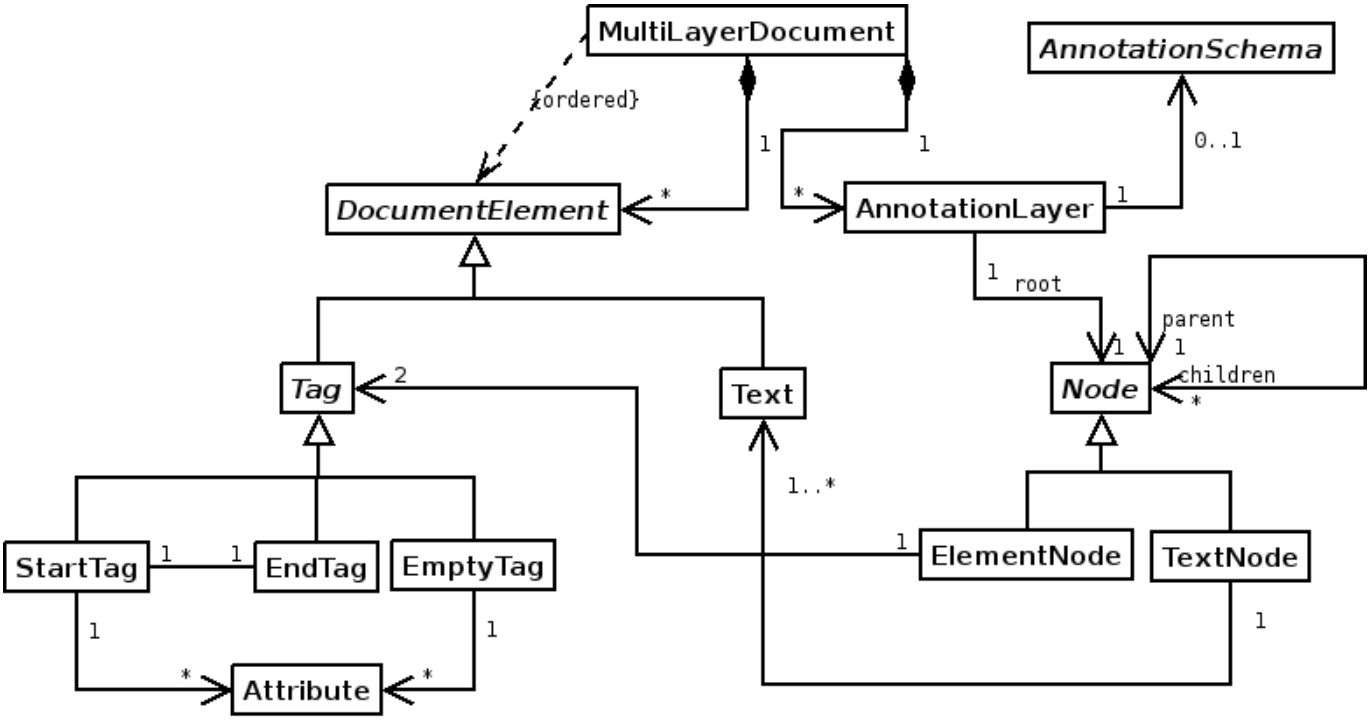
The alternative data model and editor application will be developed using C++ and the portable wxWidgets library. The software will be available on Windows, Linux and possibly Max OS X. Porting to different Unix flavors should not be too complicated. Using C++ allows the editor to be fast and small. There is no need for a Java runtime environment or the Eclipse platform to be installed.

The editor will feature the usual editing options like redo, undo, cut, copy, and paste. The user will be able to specify an annotation schema for each annotation layer, e.g. DTD or XML-Schema. The editor will assist the user by providing automatic validation, context sensitive completion of elements, and syntax highlighting.

A plug-in interface for annotation schema implementations and import and export filters are planned and

will allow users to extend the editor.

**Figure 11: The UML class and association diagram of the alternative processing model.**

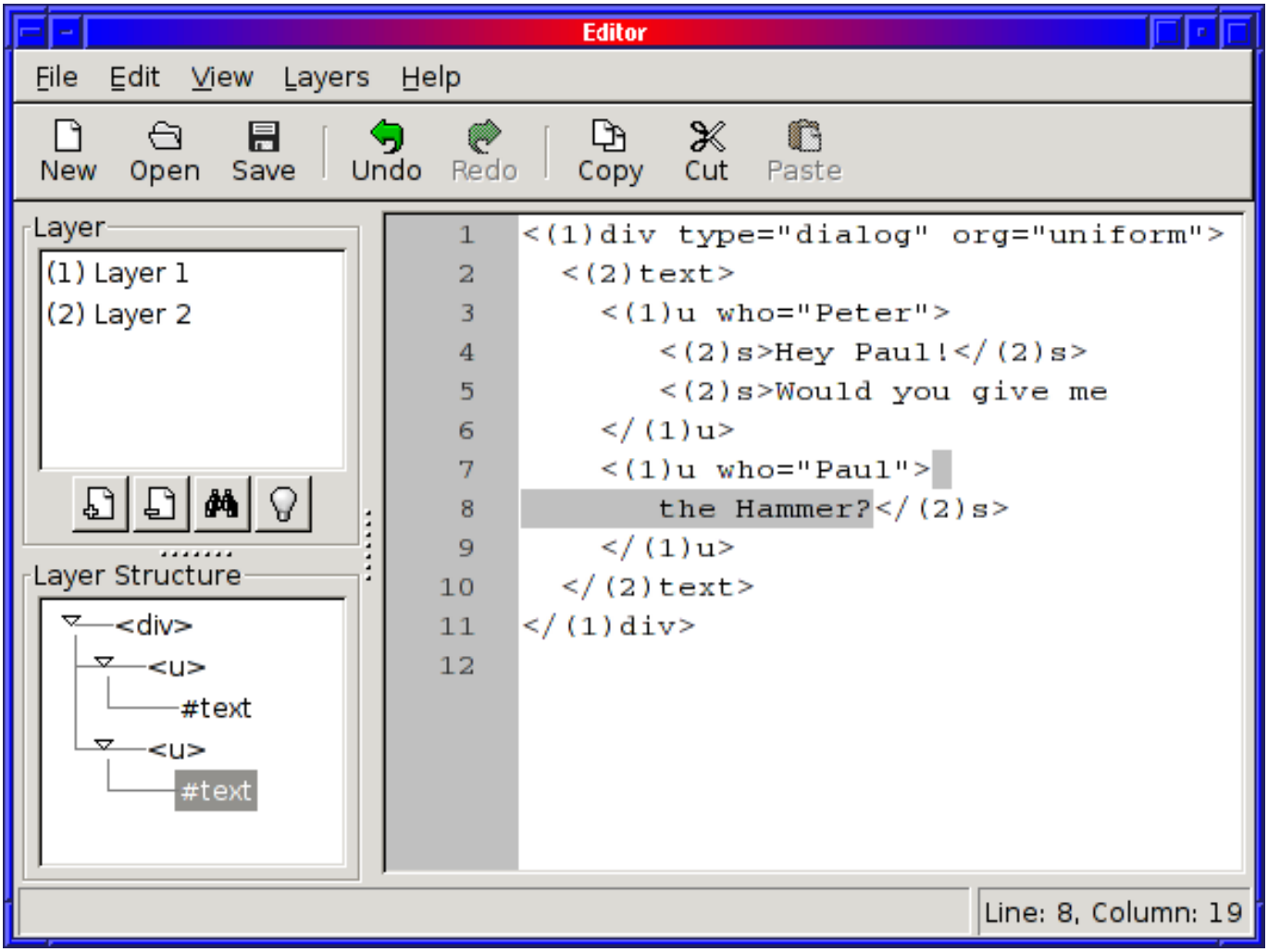



**[Link to open this graphic in a separate page]**

Figure 11 shows the UML class and association diagram of the alternative data model implementation. The *MultiLayerDocument* class is a container for the linear and the hierarchical structures. The linear structure is an array of type *DocumentElement*. This abstract class is the generalization for the *Text* and abstract *Tag* classes. *StartTag*, *EndTag* and *EmptyTag* are the concrete classes implementing tags. All but the EndTag may carry *Attribute*s. The linear structure is modified depending on the actions the user performs in the editor.

The *AnnotationLayer* class implements the annotation layers. The hierarchical structure is made up from *Node* objects. The abstract class *Node* is the generalization for the classes *ElementNode* and *TextNode*. Both classes link onto the appropriate object in the linear structure. Each *AnnotationLayer* may link onto an *AnnotationSchema*, which is an implementation of an annotation scheme (e.g. DTD, Schema, RelaxNG). The hierarchical structure will be build by processing the linear structure while the user is not typing. Currently the editor is in development and not all features have been implemented, yet.

**Figure 12: A screenshot of the prototype of the alternative editor implementation.**




**[Link to open this graphic in a separate page]**

## Notes

1. With the Layered Markup and Annotation Language (LMNL, [Tennison and Piez (2002)] ) a general multi-purpose markup language exists. The set of structures which can be modelled with LMNL is a superset of the set of structures which can be modelled with CONCUR.

2. A similar concept was introduced by [Bayerl et al. 2003] under the name Annotation Level.

3. This is one of the differences between annotation layers and namespaces.

4. More aspects on the relation between multiple-annotated documents and namespace techniques can be found in [Witt (2005)] .

5. We would have the same difficulties in SGML CONCUR since the document type specifications correspond to the declared document types defined by the used DTDs.

6. More precisely these indices are an abbreviated form of an XPath predicate, which uses the XPath function `position()`. The full predicate notation for `/(2)text[1]` is

`/(2)text[position() = 1]`. The annotation layer may also be represented by an additional function: `/text[annotationLayer() = '2' and position() = 1]`.

---


### *Acknowledgments*

We would like to thank the members of the collaborative research group on linguistic modelling of information, funded be the DFG, and Vivian Raithel and Thorsten Trippel.


---

### *Bibliography*


**[ACH/ACL/ALLC (1994)]** Association for Computers and the Humanities, Association for Computational Linguistics, and Association for Literary and Linguistic Computing. 1994. *Guidelines for Electronic Text Encoding and Interchange (TEI P3)*. Ed. C. M. Sperberg-McQueen and Lou Burnard. Chicago, Oxford: Text Encoding Initiative, 1994.

**[Barnard et al. (1995)]** Barnard, David; Burnard, Lou; Gaspart, Jean-Pierre; Price, Lynne A.; Sperberg-McQueen, C. M.; Varile, Giovanni Battista. "Hierarchical Encoding of Text: Technical Problems and SGML Solutions." The Text Encoding Initiative: Background and Contents, Guest Editors Nancy Ide and Jean Vèronis = Computers and the Humanities 29/3 (1995) 211-231.

**[Bayerl et al. 2003]** Bayerl, Petra Saskia, Harald Lüngen, Daniela Goecke, Andreas Witt, and Daniel Naber: Methods for the semantic analysis of document markup. In: Roisin, C., E. Munson and C. Vanoirbeek (Ed.): Proceedings of the ACM Symposium on Document Engineering (DocEng 2003). pp. 161 - 170

**[DeRose (2004)]** Steven DeRose. *Markup overlap: a review and a horse.* Paper presented at Extreme Markup Languages 2004, August, Montreal, Quebec. August 2004.

**[DOM]** Arnaud Le Hors, Philippe Le Hégaret, Gavin Nicol, Lauren Wood, Mike Champion, Steve Byrne (Ed.,2005). Document Object Model Core.http://www.w3.org/TR/2004/REC-DOM-Level-3-Core-20040407/core.html

**[Durusau & Brook O'Donnell (2002a)]** Patrick Durusau and Matthew Brook O'Donnell. "Coming down from the trees: Next step in the evolution of markup?" Late breaking paper presented at Extreme Markup, Montreal, 2002.

**[Durusau & Brook O'Donnell (2002b)]** Patrick Durusau and Matthew Brook O'Donnell. "Concurrent Markup for XML Documents." Presented at XML Europe 2002. http://www.idealliance.org/papers/xmle02/dx_xmle02/papers/03-03-07/03-03-07.html

**[Goldfarb (1990)]** Charles F. Goldfarb (1990). The SGML handbook. Oxford: Clarendon Press.

**[Hilbert (2005)]** Hilbert, Mirco (2005). MuLaX - ein Modell zur Verarbeitung mehrfach XML-strukturierter Daten. Diploma Thesis, Universität Bielefeld. http://www.Mirco-Hilbert.de/MuLaX/

**[ISO 8879:1986]** ISO 8879:1986. Information processing - Text office systems - Standard Generalized Markup Language (SGML).

**[Object Technology International, Inc. (2003)]** Object Technology International, Inc. (Ed.,2003). *Eclipse Platform Technical Overview (Eclipse White Paper)*, IBM Corporation et al. http://www.eclipse.org

**[Sperberg-McQueen and Huitfeldt (1998)]** Sperberg-McQueen, C. M. and Claus Huitfeldt (1998).

Concurrent Document Hierarchies in MECS and SGML, ALLC-ACH98, Joint Conference of the ALLC and ACH, Debrecen, Hungary

**[Tennison and Piez (2002)]** Tennison, Jeni and W. Piez (2002) The Layered Markup and Annotation Language. LMNL-ORG. http://www.lmnl.net

**[Witt (2002)]** Andreas Witt (2002). Multiple Informationsstrukturierung mit Auszeichnungssprachen. Dissertation. Universität Bielefeld.

**[Witt (2005)]** Andreas Witt (2005). Multiple Hierarchies: New Aspects of an Old Solution. In: Stefanie Dipper, Michael Götze, and Manfred Stede (eds.). 2005. "Heterogeneity in Focus: Creating and and Using Linguistic Databases", volume 2 of "Interdisciplinary Studies on Information Structure (ISIS), Working Papers of the SFB 632". University of Potsdam, Germany. (Corrected reprint of an Extreme Markup 2004 paper) http://www.sfb632.uni-potsdam.de/isis.php

**[Witt et al. 2005]** Andreas Witt, Daniela Goecke, Felix Sasaki, and Harald Lüngen (2005). *Unification of XML Documents with Concurrent Markup* Literary and Linguistic Computing, 20:1 103-116. http://www.w3.org/TR/2004/REC-DOM-Level-3-Core-20040407/core.html

---

Making CONCUR work

*Mirco Hilbert [Justus-Liebig-University Gießen]*
*Oliver Schonefeld [Bielefeld University]*
*Andreas Witt [Bielefeld University]*

---