A Simple API for XCONCUR

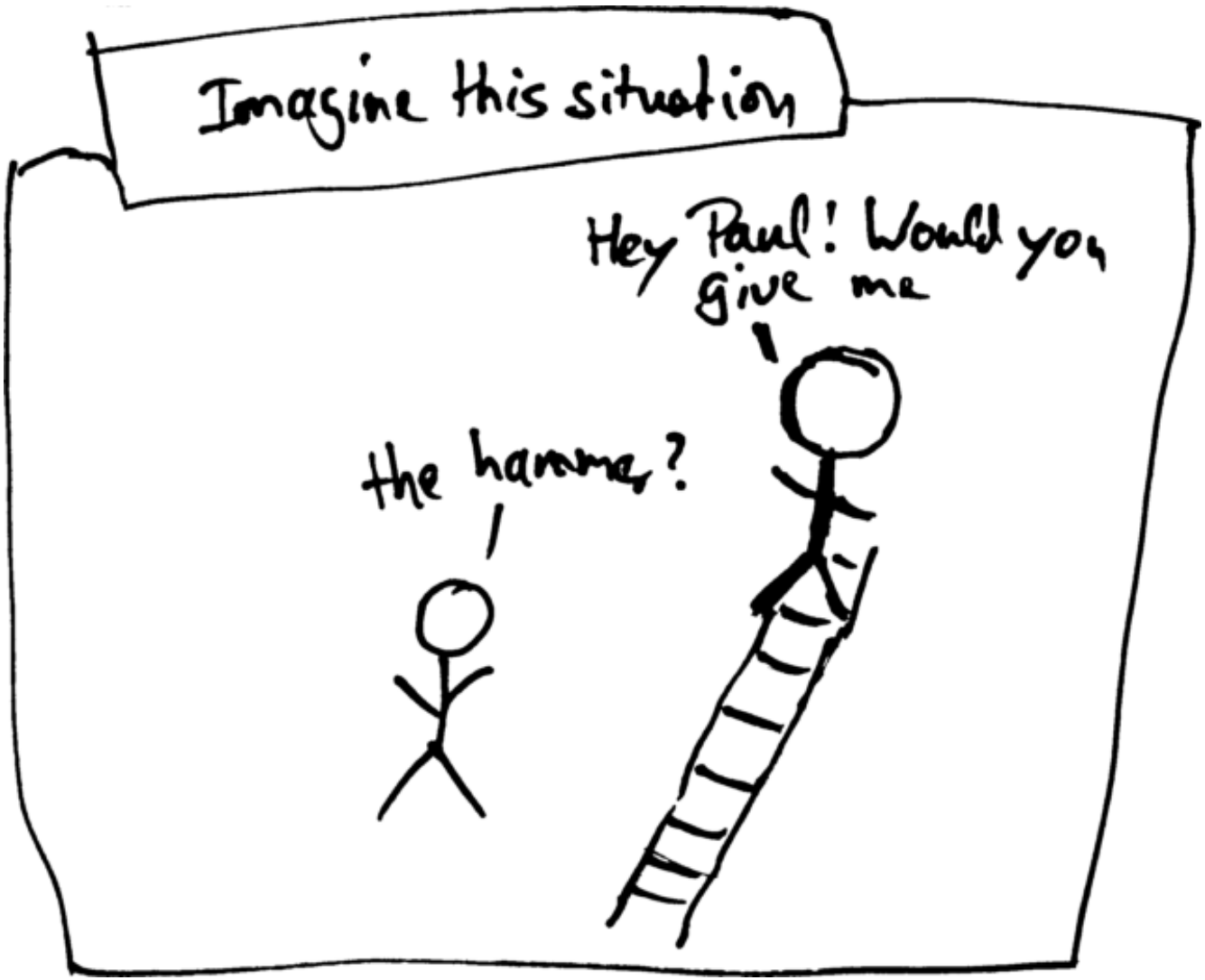
Processing concurrent markup using an event-centric API

Oliver Schonefeld

University of Tübingen

Balisage 2008, Montréal, Canada

August 13th 2008



With apologies to xkcd.com ... ;)

Agenda

1 XCONCUR

2 Simple API for XCONCUR

3 Conclusion

XCONCUR

XCONCUR properties (1/2)

- multiple annotation layers can be encoded in one document
- all elements are assigned to an annotation layer by a prefixed annotation layer id
- an annotation schema for an annotation layer is declared ...
  - explicitly by using an annotation schema declaration
  - implicitly by the markup used on the annotation layer
- multi rooted trees: all annotation layers represented simultaneously as a set of trees spanning over the same character data
- “bring (back) SGML's CONCUR option to XML"

XCONCUR properties (2/2)

- concept of well-formedness
- two levels of validation:

  PER ANNOTATION LAYER
  : by means of an annotation schema language (such as, e.g. DTD, XML Schema or RELAX NG)

  ACROSS ANNOTATION LAYER
  : by means of a XCONCUR-CL constaint sets

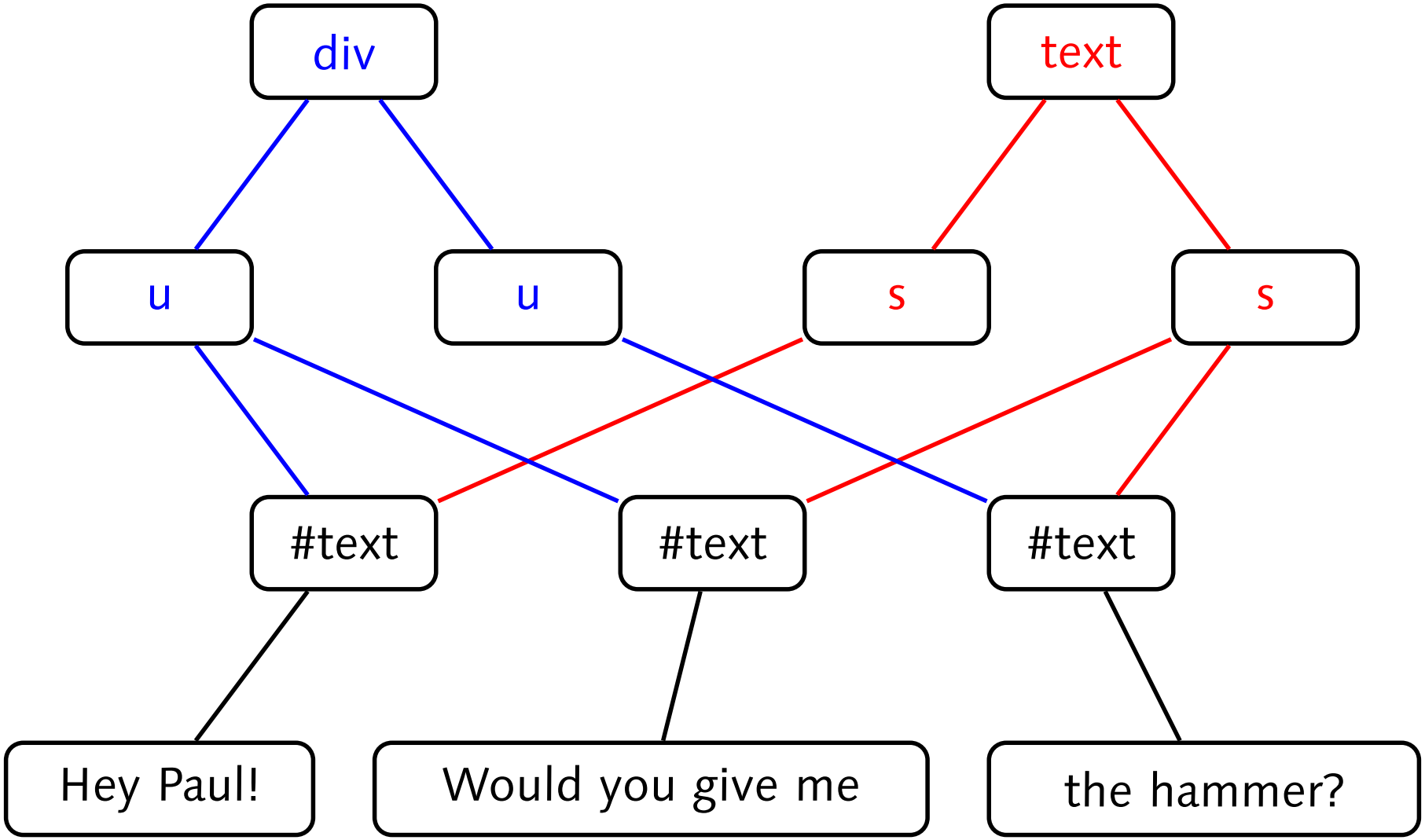
Example: the dialog annotated using XCONCUR (1/2)

```
<?xconcur version="1.1" encoding="utf-8"?>
<?xconcur-schema layer-id="l1" root="div"
   system="teispok2.dtd"?>
<?xconcur-schema layer-id="l2" root="text"
   system="teiana2.dtd"?>
<(l1)div type="dialog" org="uniform">
 <(l2)text>
   <(l1)u who="Peter">
     <(l2)s>Hey Paul!</(l2)s>
     <(l2)s>Would you give me
   </(l1)u> <(l1)u who="Paul">
      the hammer?</(l2)s>
   </(l1)u>
 </(l2)text>
</(l1)div>
```

Example: the dialog annotated using XCONCUR (2/2)

```
1  <?xconcur version="1.1" encoding="utf-8"?>
2  <?xconcur-schema layer-id="l1" root="div"
3     system="teispok2.dtd"?>
4  <?xconcur-schema layer-id="l2" root="text"
5     system="teiana2.dtd"?>
6  <(l1)div type="dialog" org="uniform">
7    <(l2)text>
8      <(l1)u who="Peter">
9        <(l2)s>Hey Paul!</(l2)s>
10       <(l2)s>Would you give me
11     </(l1)u> <(l1)u who="Paul">
12       the hammer?</(l2)s>
13     </(l1)u>
14   </(l2)text>
15 </(l1)div>
```

Example: multi rooted trees view of the dialog



XCONCUR versus SGML CONCUR

Most important differences to SGML CONCUR . . .

- elements without an annotation layer id are not allowed
- elements with the same generic identifier must be annotated explicitly

Example

Accepted

```
<(l1)a><(l2)x>foo<(l1)br/><(l2)br/>bar</(l1)a></(l2)x>
```

Not Accepted

```
<(l1)a><(l2)x>foo<br/>bar</(l1)a></(l2)x>
```

XCONCUR versus XML Namespaces

XML Namespaces and annotation layer id are different concepts . . .

- XML Namespaces allow to use elements from different annotation schemas with conflicting names; elements must be nested properly
- annotation layer ids assign elements to an annotation layer
- XCONCUR allows to use (potentially multiple) namespaces on each annotation layer

Example

```
<(l1)foo:bar xmlns:foo="urn:x-foo:foo">...</(l1)foo:bar>
```

Simple API for XCONCUR

Simple API for XCONCUR

- API based on well-known XML SAX API
- defines events and a set classes and interfaces
- while parsing the structure of document is signaled by events
- event consumers must decide by themself what to do

Events defined by API (1/3)

Start Document
: beginning of a document

End Document
: end of a document

Start Layer*
: beginning of a document

End Layer*
: beginning of a document

* = events unique to XCONCUR API

Events defined by API (1/3)

Start Document

beginning of a document

End Document

end of a document

Start Layer*

beginning of a document

End Layer*

beginning of a document

* = events unique to XCONCUR API

Events defined by API (1/3)

START DOCUMENT
: beginning of a document

END DOCUMENT
: end of a document

START LAYER*
: beginning of a document

END LAYER*
: beginning of a document

* = events unique to XCONCUR API

Events defined by API (1/3)

START DOCUMENT
: beginning of a document

END DOCUMENT
: end of a document

START LAYER*
: beginning of a document

END LAYER*
: beginning of a document

* = events unique to XCONCUR API

Events defined by API (1/3)

Start Document
: beginning of a document

End Document
: end of a document

Start Layer*
: beginning of a document

End Layer*
: beginning of a document

* = events unique to XCONCUR API

Events defined by API (2/3)

START PRIMARY DATA*

begining of primary data

END PRIMARY DATA*

end of primary data

START ELEMENT

beginning of an element

END ELEMENT

end of an element

* = events unique to XCONCUR API

Events defined by API (2/3)

START PRIMARY DATA*

begining of primary data

END PRIMARY DATA*

end of primary data

START ELEMENT

beginning of an element

END ELEMENT

end of an element

* = events unique to XCONCUR API

Events defined by API (2/3)

START PRIMARY DATA*
: begining of primary data

END PRIMARY DATA*
: end of primary data

START ELEMENT
: beginning of an element

END ELEMENT
: end of an element

* = events unique to XCONCUR API

Events defined by API (2/3)

START PRIMARY DATA*
: begining of primary data

END PRIMARY DATA*
: end of primary data

START ELEMENT
: beginning of an element

END ELEMENT
: end of an element

* = events unique to XCONCUR API

Events defined by API (2/3)

START PRIMARY DATA*
: begining of primary data

END PRIMARY DATA*
: end of primary data

START ELEMENT
: beginning of an element

END ELEMENT
: end of an element

* = events unique to XCONCUR API

Events defined by API (3/3)

Characaters
: character data

Ignoreable Whitespace
: whitespace data before and after primary data

Start Prefix Mapping
: beginning of scope for prefixed namespace mapping

End Prefix Mapping
: end of scope for prefixed namespace mapping

* = events unique to XCONCUR API

Events defined by API (3/3)

Characaters

character data

Ignoreable Whitespace

whitespace data before and after primary data

Start Prefix Mapping

beginning of scope for prefixed namespace mapping

End Prefix Mapping

end of scope for prefixed namespace mapping

* = events unique to XCONCUR API

Events defined by API (3/3)

Characaters
: character data

Ignoreable Whitespace
: whitespace data before and after primary data

Start Prefix Mapping
: beginning of scope for prefixed namespace mapping

End Prefix Mapping
: end of scope for prefixed namespace mapping

* = events unique to XCONCUR API

Events defined by API (3/3)

Characaters
: character data

Ignoreable Whitespace
: whitespace data before and after primary data

Start Prefix Mapping
: beginning of scope for prefixed namespace mapping

End Prefix Mapping
: end of scope for prefixed namespace mapping

* = events unique to XCONCUR API

Events defined by API (3/3)

Characaters
: character data

Ignoreable Whitespace
: whitespace data before and after primary data

Start Prefix Mapping
: beginning of scope for prefixed namespace mapping

End Prefix Mapping
: end of scope for prefixed namespace mapping

* = events unique to XCONCUR API

Document as events (1/7)

```
<?xconcur version="1.1" encoding="utf-8"?>
<(l1)div type="dialog" org="uniform">
  <(l2)text>
    <(l1)u who="Peter">
      <(l2)s>Hey Paul!</(l2)s>
      <(l2)s>Would you give me
    </(l1)u> <(l1)u who="Paul">
       the hammer?</(l2)s>
    </(l1)u>
  </(l2)text>
</(l1)div>
```

Document as events (2/7)

```
<?xconcur version="1.1" encoding="utf-8"?>
<(l1)div type="dialog" org="uniform">
  <(l2)text>
    <(l1)u who="Peter">
      <(l2)s>Hey Paul!</(l2)s>
      <(l2)s>Would you give me
    </(l1)u> <(l1)u who="Paul">
       the hammer?</(l2)s>
    </(l1)u>
  </(l2)text>
</(l1)div>
```

START DOCUMENT

Document as events (3/7)

```
<?xconcur version="1.1" encoding="utf-8"?>
<(l1)div type="dialog" org="uniform">
  <(l2)text>
    <(l1)u who="Peter">
      <(l2)s>Hey Paul!</(l2)s>
      <(l2)s>Would you give me
    </(l1)u> <(l1)u who="Paul">
       the hammer?</(l2)s>
    </(l1)u>
  </(l2)text>
</(l1)div>
```

Start Document
Start Layer
Start Element

Document as events (4/7)

```
<?xconcur version="1.1" encoding="utf-8"?>
<(l1)div type="dialog" org="uniform">
  <(l2)text>
    <(l1)u who="Peter">
      <(l2)s>Hey Paul!</(l2)s>
      <(l2)s>Would you give me
    </(l1)u> <(l1)u who="Paul">
       the hammer?</(l2)s>
    </(l1)u>
  </(l2)text>
</(l1)div>
```

Start Document
Start Layer
Start Element
Start Layer
Start Element

Document as events (5/7)

```
<?xconcur version="1.1" encoding="utf-8"?>
<(l1)div type="dialog" org="uniform">
  <(l2)text>_
___<(l1)u who="Peter">
     <(l2)s>Hey Paul!</(l2)s>
     <(l2)s>Would you give me
    </(l1)u> <(l1)u who="Paul">
      the hammer?</(l2)s>
    </(l1)u>
  </(l2)text>
</(l1)div>
```

Start Document
Start Layer
Start Element
Start Layer
Start Element
Start Primary Data
Characters

Document as events (6/7)

```
<?xconcur version="1.1" encoding="utf-8"?>
<(l1)div type="dialog" org="uniform">
  <(l2)text>
    <(l1)u who="Peter">
      <(l2)s>Hey Paul!</(l2)s>
      <(l2)s>Would you give me
    </(l1)u> <(l1)u who="Paul">
       the hammer?</(l2)s>
    </(l1)u>
  </(l2)text>
</(l1)div>
```

- Start Document
- Start Layer
- Start Element
- Start Layer
- Start Element
- Start Primary Data
- Characters
- Start Element

Document as events (7/7)

```
<?xconcur version="1.1" encoding="utf-8"?>
<(l1)div type="dialog" org="uniform">
  <(l2)text>
    <(l1)u who="Peter">
      <(l2)s>Hey Paul!</(l2)s>
      <(l2)s>Would you give me
    </(l1)u> <(l1)u who="Paul">
       the hammer?</(l2)s>
    </(l1)u>
  </(l2)text>
</(l1)div>
```

Start Document
Start Layer
Start Element
Start Layer
Start Element
Start Primary Data
Characters
Start Element
⋮
End Element
End Layer
End Document

Most important API classes

- Class XConcurReader
  - abstract interface to under-laying parser
  - querying and setting options
  - hooking up the ContentHandler and an error handler
  - parse a document
- Interface ContentHandler
  - message sink for events
  - needs to be implemented by user application
- various auxiliary classes and interfaces

Example: using the API (1/2)

```
1  class FilterContentHandler : public ContentHandler {
2     virtual void StartElement(const char* const layer,
3                               const char* const uri,
4                               const char* const localname,
5                               const char* const qname,
6                               const Attributes &attrs) {
7        if (strcmp(layer, "l1") == 0) {
8           // output a XML tag
9        }
10    }
11
12    // other handlers omitted
13
14 }; // class FilterContentHandler
```

Example: using the API (2/2)

```
1  try {
2     // obtain reader instance
3     XConcurReader *reader =
4        XConcurReaderFactory::CreateReader();
5     // create new ContentHandler
6     FilterContentHandler handler;
7     // register content handler with reader
8     reader->SetContentHandler(handler);
9     // create input source, "input" is an InputStream
10    InputSource source(input);
11    // parse document
12    reader->parse(&source);
13 } catch (XConcurException &e) {
14    // handle exception
15 }
```

Software

- LibXConcur
  - prototype implementation in C++
  - provides XConcurReader, ContentHandler and utility classes
  - parser currently lacks validation
  - support for a subset of DTDs is currently worked on
- package org.xconcur
  - (slightly outdated) Java bindings
  - API in 100% pure Java
  - parser uses libxconcur though Java Native Interface

Software for evaluation purposes available upon request.

Conclusion

- XCONCUR documents are processed using events
- based on well-known XML SAX API
- simple but sufficently powerful
- easy to use

Thank you for your attention!

Questions?

Oliver Schonefeld, University of Tübingen
oliver.schonefeld@uni-tuebingen.de