XLIFF 2.0:

Processing Great Expectations

Andrew Pimlott <andrew.pimlott@welocalize.com>
Welocalize

Goal

Add alt-trans support to OpenTM2.

Goal

Add alt-trans support to OpenTM2.

```
<trans-unit id='1' translate="yes">
  <source>Hello world</source>
  <alt-trans match-quality="65">
    <source>hello</source>
    <target>bonjour</target>
  </alt-trans>
</trans-unit>
```

Goal

Add alt-trans support to OpenTM2.

OR

"alt-trans: Can the Naive Implementor Do It?"

(A nod to Tim Bray and Michael Leventhal for
"XML: Can the Desperate Perl Hacker do it?")

Problem

```
<trans-unit id="1" translate="no">
  <source>abc</source>
</trans-unit>
```

Problem

```
<trans-unit id="1" translate="no">
  <source>abc</source>
</trans-unit>
```

- Do I still show them?
- Can the translator override?

Problem

```
<trans-unit id="1" translate="yes">
  <source>abc</source>
  <target>def</target> <!-- no state -->
</trans-unit>
```

Problem

```
<trans-unit id="1" translate="yes">
  <source>abc</source>
  <target>def</target> <!-- no state -->
</trans-unit>
```

- How should this segment be presented? Needs translation? Already translated?
- Does it matter that the target differs from the source?

Problem

```
<trans-unit id="1" translate="yes">
  <source>abc</source>
  <target state="translated">def</target>
</trans-unit>
```

Problem

```
<trans-unit id="1" translate="yes">
  <source>abc</source>
  <target state="translated">def</target>
</trans-unit>
```

- How should this segment be presented? Unmodifiable? Modifiable with a warning? Skipped by default?

Problem

```
<trans-unit id="1" translate="yes">
  <source>abc</source>
  <target state="translated">def</target>
</trans-unit>
```

- How should this segment be presented? Unmodifiable? Modifiable with a warning? Skipped by default?
- What about other states?

Problem

```
<alt-trans match-quality="65">
  <source>abc</source>
  <target>def</target>
</trans-unit>
```

Problem

```
<alt-trans match-quality="65">
  <source>abc</source>
  <target>def</target>
</trans-unit>
```

- What is the format of this attribute?
- What do I do with it?

Problem

How do I use an alt-trans?

Problem

How do I use an alt-trans?

- How do I set metadata when using an alt-trans proposal?
  - `state-qualifier`?

Problem

How do I use an alt-trans?

- How do I set metadata when using an alt-trans proposal?
  - `state-qualifier`?
- Do I have to adapt the inline markup? May I? How?

Problem

What does the output XLIFF look like?

- `<tool>`?
- `<phase-group>`?
- `tool-id`, `phase-name` attributes?
- What am I responsible for preserving?

Problem

By the way, how do I know I was supposed to translate this?

Goal

Add alt-trans support to OpenTM2.

OR

"alt-trans: Can the Naive Implementor Do It?"

ANSWER

Not really.

What's the real problem?

What's the real problem?

- I'm a doofus.

What's the real problem?

- I'm a doofus.
- XLIFF 1.2 is too complicated.

What's the real problem?

- I'm a doofus.
- XLIFF 1.2 is too complicated.
- XLIFF 1.2 lacks processing expectations.

And to me that [processing expectations] is where the biggest interoperability issue resides.

- Yves Savourel

My take of user feedback is that the standard does not ensure interoperability because the lack of well-defined processing expectations.

- David Filip

Two types of conformance are defined:

1. *conformance of XLIFF markup declarations*
2. *conformance of processing requirements for XLIFF markup*

- Christian Lieske

We need to start including in the specs what "XLIFF enabled tools" should do with each and every attribute or element we include in the specs. If there are no defined expectations for a given element, we should consider dropping it.

- Rodolfo M. Raya

What are processing expectations?

Docbook

- `<emphasis>` processing expectations:

  Formatted inline. Emphasized text is traditionally presented in italics or boldface. A Role attribute of bold or strong is often used to generate boldface, if italics is the default presentation.

XLIFF processing expectations

`match-quality` processing expectations:

XLIFF processing expectations

`match-quality` processing expectations:

- `match-quality` must be a whole, positive, decimal number between 0 and 100, inclusive.

XLIFF processing expectations

`match-quality` processing expectations:

- `match-quality` must be a whole, positive, decimal number between 0 and 100, inclusive.
- Higher `match-quality` indicates a better match.

XLIFF processing expectations

`match-quality` processing expectations:

- `match-quality` must be a whole, positive, decimal number between 0 and 100, inclusive.
- Higher `match-quality` indicates a better match.
- `match-quality` values are only comparable within a `<trans-unit>`, and only for `<alt-trans>` with the same `tool-id` attribute.

XLIFF processing expectations

`match-quality` processing expectations:

- `match-quality` must be a whole, positive, decimal number between 0 and 100, inclusive.
- Higher `match-quality` indicates a better match.
- `match-quality` values are only comparable within a `<trans-unit>`, and only for `<alt-trans>` with the same `tool-id` attribute.
- It is recommended that when the `tool-id` is not recognized, processors interpret a value of 74 and lower as "low fuzzy", 75 to 99 (inclusive) as "high fuzzy", and 100 as "exact".

XLIFF processing expectations

`<trans-unit translate="...">` processing expectations:

XLIFF processing expectations

`<trans-unit translate="...">` processing expectations:

- When `translate="no"`, processors should display the `<source>` and `<target>` as unmodifiable text.

XLIFF processing expectations

`<trans-unit translate="...">` processing expectations:

- When `translate="no"`, processors should display the `<source>` and `<target>` as unmodifiable text.
- When `translate="yes"`, processors should display the `<source>` as unmodifiable text, and the `<target>` as modifiable text.

What are processing expectations?

The hard part:

What are processing expectations?

The hard part:

- How do we transform an XLIFF document...

What are processing expectations?

The hard part:

- How do we transform an XLIFF document...
- based on the execution of a localization process...

What are processing expectations?

The hard part:

- How do we transform an XLIFF document...
- based on the execution of a localization process...
- so that other tools will accept the new document...

What are processing expectations?

The hard part:

- How do we transform an XLIFF document...
- based on the execution of a localization process...
- so that other tools will accept the new document...
- and understand what we've done.

What are processing expectations?

Docbook

- `<emphasis>` processing expectations:

  Formatted inline. Emphasized text is traditionally presented in italics or boldface. A Role attribute of bold or strong is often used to generate boldface, if italics is the default presentation.

What are processing expectations?

Docbook

- `<emphasis>` processing expectations:

  Formatted inline. Emphasized text is traditionally presented in italics or boldface. A Role attribute of bold or strong is often used to generate boldface, if italics is the default presentation.

- But interchange requires a 41-point "interchange questionnaire"!

What are processing expectations?

Processing expectations for XLIFF transforms:

What are processing expectations?

Processing expectations for XLIFF transforms:

- Conformance checking of (input, output) ...

What are processing expectations?

Processing expectations for XLIFF transforms:

- Conformance checking of (input, output) ...
- specified in pseudo-code ...

What are processing expectations?

Processing expectations for XLIFF transforms:

- Conformance checking of (input, output) ...
- specified in pseudo-code ...
- with respect to the execution of a localization process.

modcase processing expectations

The modcase transform operates on strings. It may only change the case of characters.

- `"Abc"` => `"Abc"` conformant
- `"Abc"` => `"aBC"` conformant
- `"Abc"` => `"cbA"` not conformant

mod_case processing expectations

```
modcase_ok(string old, string new) {
  assert(tolower(old) == tolower(new));
}
```

```
<xs:element name="xliff">
  <xs:complexType>
    <xs:sequence>
      <xs:element minOccurs="1" maxOccurs="unbounded"
                  ref="xlf:file"/>
    </xs:sequence>
    <xs:attribute name="version" fixed="2.0"
                  use="required"/>
  </xs:complexType>
</xs:element>
```

```
translate_xliff_ok(xliff old, xliff new) {
  assert(same_attributes(old, new));
  assert(same_children(old, new));

  iterator old_children = child_iterator(old);
  iterator new_children = child_iterator(new);
  while (! empty(old_children)) {
    assert(translate_file_ok(next(old_children),
                             next(new_children)));
  }
}
```

```
<xs:element name="file">
  <xs:complexType>
    <xs:sequence>
      <xs:element minOccurs="0" maxOccurs="1"
                  ref="xlf:skeleton"/>
      <xs:element minOccurs="1" maxOccurs="unbounded"
                  ref="xlf:unit"/>
    </xs:sequence>
    <xs:attribute name="srclang" use="optional"/>
    <xs:attribute name="tgtlang" use="optional"/>
    <xs:attribute name="original" use="optional"/>
    <xs:attribute name="skeleton" use="optional"/>
  </xs:complexType>
</xs:element>
```

```
translate_file_ok(file old, file new) {
  assert(same_attributes(old, new));
  assert(same_children(old, new));

  if (has_child(old, "skeleton")) {
    assert(same_xml(get_child(old, "skeleton"),
                    get_child(new, "skeleton")));
  }

  iterator old_units = child_iterator(old, "unit");
  iterator new_units = child_iterator(new, "unit");
  while (! empty(old_units)) {
    assert(translate_unit_ok(next(old_units),
                             next(new_units)));
  }
}
```

```
<xs:element name="segment">
  <xs:complexType>
    <xs:sequence>
      <xs:element minOccurs="1" maxOccurs="1"
                  ref="xlf:source"/>
      <xs:element minOccurs="0" maxOccurs="1"
                  ref="xlf:target"/>
      <xs:element minOccurs="0" maxOccurs="1"
                  ref="xlf:notes"/>
      <xs:element minOccurs="0" maxOccurs="1"
                  ref="xlf:matches"/>
    </xs:sequence>
    <xs:attribute name="id" use="optional"/>
    <xs:attribute name="translate" type="xlf:yesNo"
                  default="yes"/>
    <xs:attribute name="approved" type="xlf:yesNo"
                  default="no"/>
  </xs:complexType>
</xs:element>
```

```
translate_segment_ok(segment old, segment new) {
  assert(same_attributes(old, new));
  assert(same_xml(get_child(old, "source"),
                  get_child(new, "source")));

  if (has_child(old, "matches")) {
    assert(has_child(new, "matches"));
    assert(same_xml(get_child(old, "matches"),
                    get_child(new, "matches")));
  }

  if (has_child(old, "notes")) {
    assert(has_child(new, "notes"));
    translate_notes_ok(get_child(old, "notes"),
                       get_child(new, "notes"));
  }
  ...
}
```

```
translate_segment_ok(segment old, segment new) {
  ...
  if (get_attribute(old, "translate") == "no"
      // || translator didn't translate this segment) {
    assert(has_child(old, "target") ==
           has_child(new, "target"));
    if (has_child(old, "target")) {
      assert(same_xml(get_child(old, "target"),
                      get_child(new, "target")));
    }
  } else {  // translate="yes"
    translate_target_ok(old, new);
  }
}
```

```
<xs:element name="target">
  <xs:complexType mixed="true">
    <xs:sequence>
      <xs:element minOccurs="0" maxOccurs="unbounded"
                  ref="xlf:sc"/>
      <xs:element minOccurs="0" maxOccurs="unbounded"
                  ref="xlf:ec"/>
      <xs:element minOccurs="0" maxOccurs="unbounded"
                  ref="xlf:ph"/>
      <xs:element minOccurs="0" maxOccurs="unbounded"
                  ref="xlf:pc"/>
      <xs:element minOccurs="0" maxOccurs="unbounded"
                  ref="xlf:cp"/>
    </xs:sequence>
    <xs:attribute ref="xml:lang" use="optional"/>
    <xs:attribute ref="xml:space" use="optional"/>
  </xs:complexType>
</xs:element>
```

```
translate_target_ok(segment old, segment new) {
  // ???
  // Do we require there to be a target?
  // What inline elements are allowed?
}
```

XLIFF processing expectations

Wait a minute:

XLIFF processing expectations

Wait a minute:

- How did we know this was a translation process?

XLIFF processing expectations

Wait a minute:

- How did we know this was a translation process?
- It was implicit.

XLIFF processing expectations

Wait a minute:

- How did we know this was a translation process?
- It was implicit.
- Make it explicit!

XLIFF processing expectations

Straw man proposal:

```
<xliff version="2.0" process="translate">
  ...
</xliff>

transform_ok_xliff(xliff old, xliff new) {
  if (get_attribute(old, "process") == "translate") {
    return translate_xliff_ok(old, new);
  } elseif (get_attribute(old, "process") == "review") {
    return review_ok_xliff(old, new);
  }
  elseif ...
}
```

XLIFF processing expectations

Limitations and challenges:

XLIFF processing expectations

Limitations and challenges:

- Only handles out-and-back workflow.

XLIFF processing expectations

Limitations and challenges:

- Only handles out-and-back workflow.
  - Challenge: processing expectations for a multi-step workflow.

XLIFF processing expectations

Limitations and challenges:

- Only handles out-and-back workflow.
  - Challenge: processing expectations for a multi-step workflow.
- Only handles a fixed set of process types.

XLIFF processing expectations

Limitations and challenges:

- Only handles out-and-back workflow.
  - Challenge: processing expectations for a multi-step workflow.
- Only handles a fixed set of process types.
  - Challenge: extensible process types.

Conclusions

Processing expectations:

Conclusions

Processing expectations:

- Tell the implementer exactly what to do.

Conclusions

Processing expectations:

- Tell the implementer exactly what to do.
- Can show that an implementation is not conformant.

Conclusions

Processing expectations:

- Tell the implementer exactly what to do.
- Can show that an implementation is not conformant.
- Promote interoperability.

Conclusions

Processing expectations:

- Tell the implementer exactly what to do.
- Can show that an implementation is not conformant.
- Promote interoperability.
- Are laborious to formulate.

Conclusions

Processing expectations:

- Tell the implementer exactly what to do.
- Can show that an implementation is not conformant.
- Promote interoperability.
- Are laborious to formulate.
- Require brutal simplification.

Conclusions

Processing expectations:

- Tell the implementer exactly what to do.
- Can show that an implementation is not conformant.
- Promote interoperability.
- Are laborious to formulate.
- Require brutal simplification.
- Provide a solid foundation for new features.

Conclusions

Now, I return to this young [standard].
And the communication I have got to
make is, that [it] has great expectations.

- Charles Dickens

Conclusions

Now, I return to this young [standard].
And the communication I have got to
make is, that [it] has great expectations.

- Charles Dickens

- ... with processing expectations.

Conclusions

Now, I return to this young [standard].
And the communication I have got to
make is, that [it] has great expectations.

- Charles Dickens

- ... with processing expectations.
- It's not going to be easy.

Conclusions

Now, I return to this young [standard].
And the communication I have got to
make is, that [it] has great expectations.

- Charles Dickens

- ... with processing expectations.
- It's not going to be easy.
- It will require leaving out features.

Conclusions

Now, I return to this young [standard].
And the communication I have got to
make is, that [it] has great expectations.

- Charles Dickens

- ... with processing expectations.
- It's not going to be easy.
- It will require leaving out features.
- But it's necessary for effective interoperability.

Conclusions

Now, I return to this young [standard].
And the communication I have got to
make is, that [it] has great expectations.

- Charles Dickens

- ... with processing expectations.
- It's not going to be easy.
- It will require leaving out features.
- But it's necessary for effective interoperability.
- And it's best for the future of XLIFF.