JLIFF, Creating a JSON Serialization of OASIS XLIFF

Filip, Ritchie, & van Engelen
ADAPT Centre, Vistatec, Genivia

XML Prague 2019, 9th February

LIOM design

JLIFF Design

Thanks a million!

Q & A

@merzbauer
@philinthecloud
@vEngelenRobert

JLIFF, Creating a JSON Serialization of OASIS XLIFF

Filip, Ritchie, & van Engelen

ADAPT Centre, Vistatec, Genivia

XML Prague 2019, 9th February

XLIFF 2 namespaces

Namespaces that appear both in XLIFF 2.1 and XLIFF 2.0

urn:oasis:names:tc:xliff:document:2.0 <!-- Core -->
urn:oasis:names:tc:xliff:matches:2.0 <!-- Translation Candidates Module -->
urn:oasis:names:tc:xliff:glossary:2.0 <!-- Glossary Module -->
urn:oasis:names:tc:xliff:fs:2.0 <!-- Format Style Module -->
urn:oasis:names:tc:xliff:metadata:2.0 <!-- Metadata Module -->
urn:oasis:names:tc:xliff:resourcedata:2.0 <!-- Resource Data Module -->
urn:oasis:names:tc:xliff:sizerestriction:2.0 <!-- Size and Length Restriction Module -->
urn:oasis:names:tc:xliff:validation:2.0 <!-- Validation Module -->

Namespaces that appear only in XLIFF 2.1

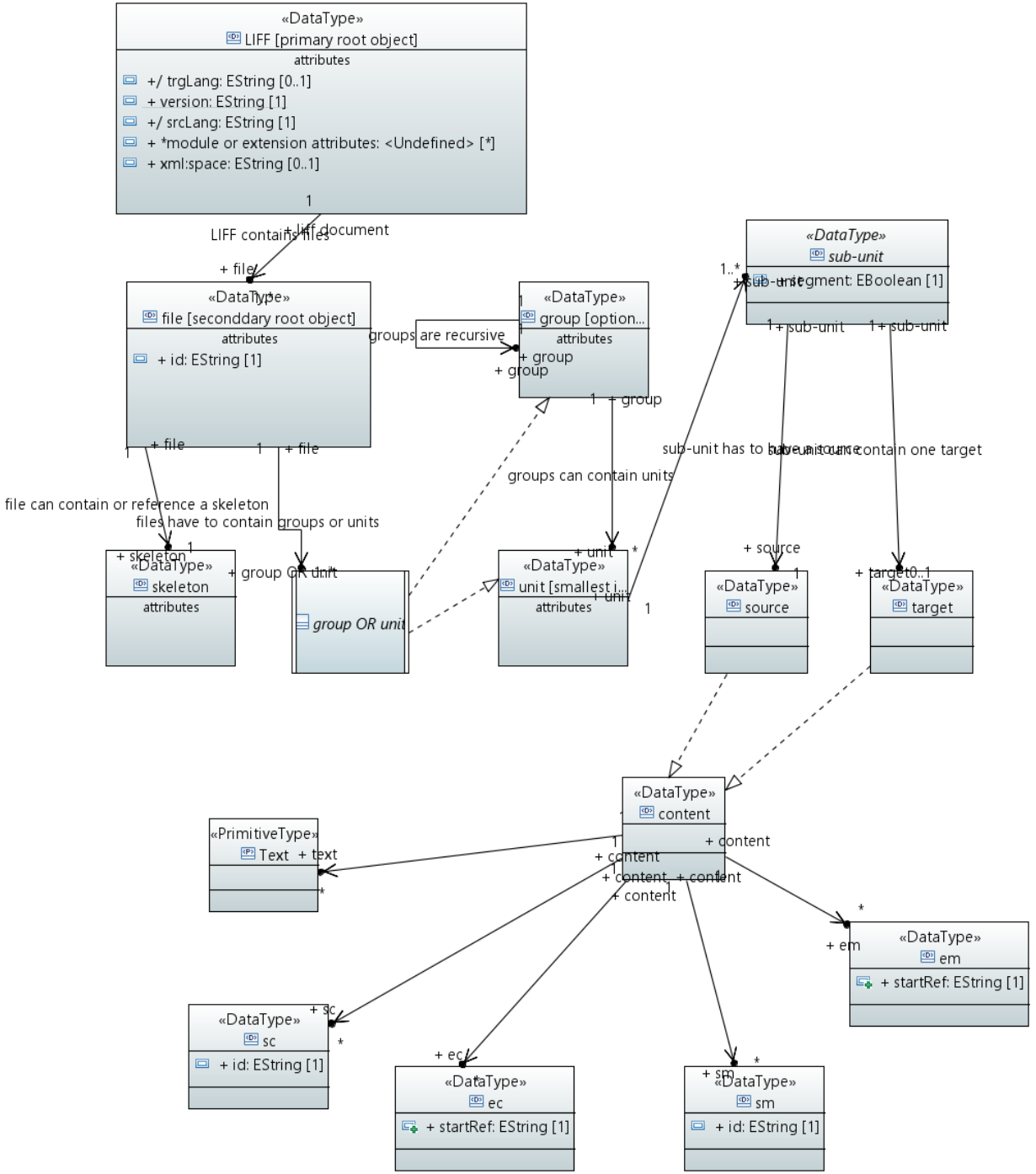
http://www.w3.org/2005/11/its <!-- ITS Module -->
urn:oasis:names:tc:xliff:itsm:2.1 <!-- ITS Module -->

Namespaces that appear only in XLIFF 2.0

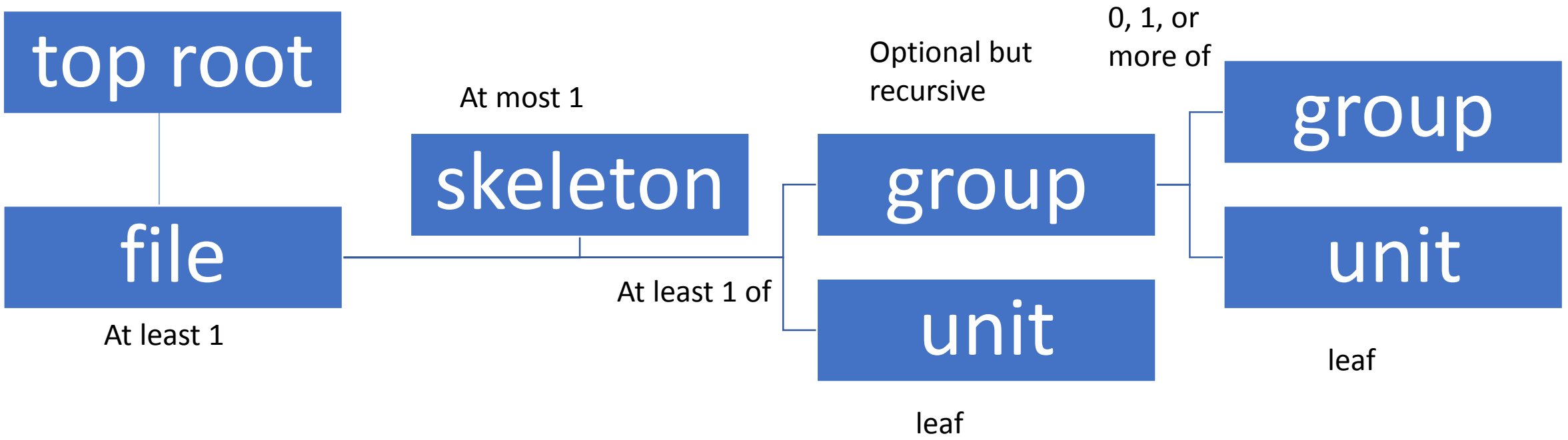
urn:oasis:names:tc:xliff:changetracking:2.0 <!-- Change Tracking Module -->

LIOM design

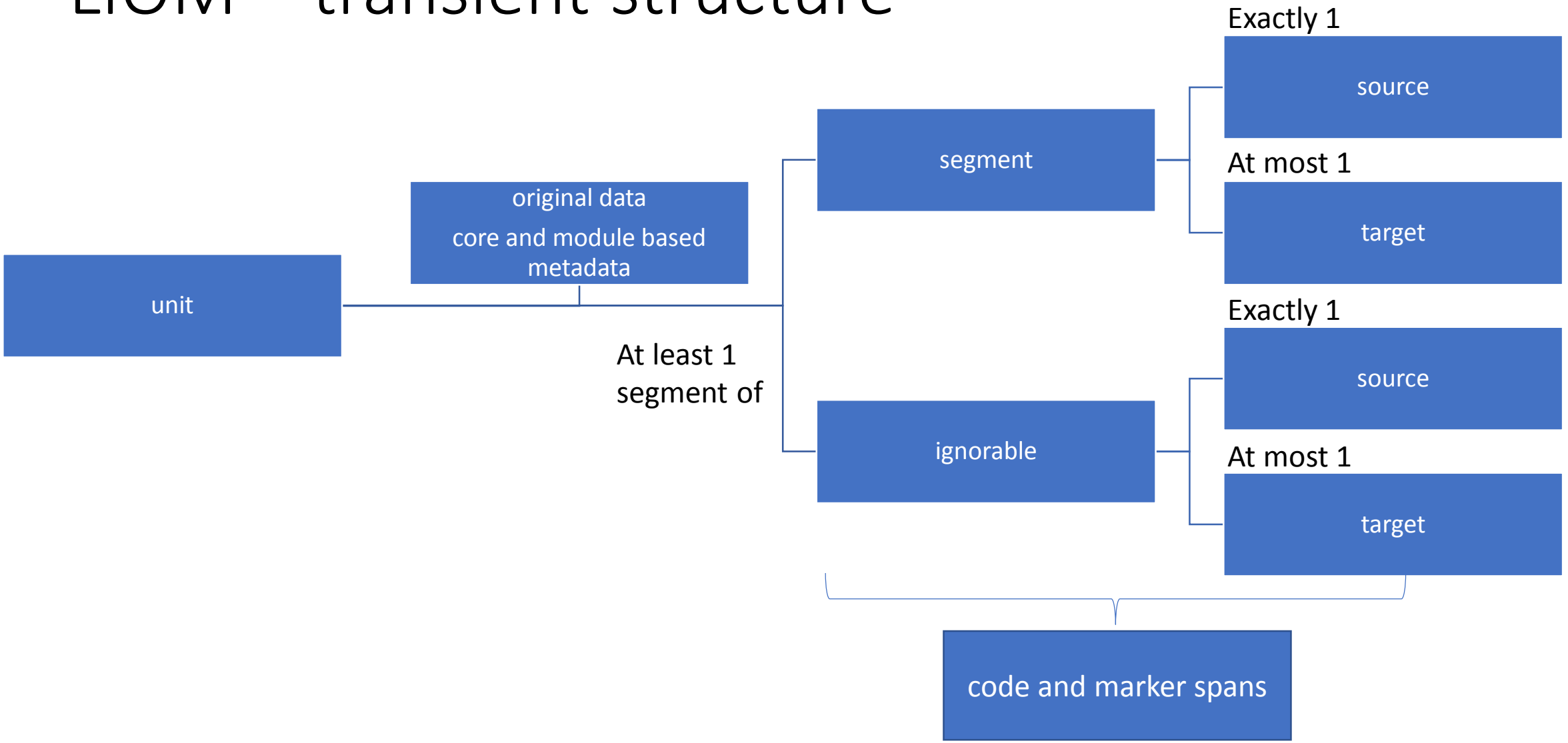
LIOM


«DataType»
LIFF [primary root object]
attributes
+/ trgLang: EString [0..1]
+ version: EString [1]
+/ srcLang: EString [1]
+ *module or extension attributes: <Undefined> [*]
+ xml:space: EString [0..1]
1
LIFF contains files
+ liff document
+ file
«DataType»
file [seconddary root object]
attributes
+ id: EString [1]
groups are recursive
«DataType»
group [option...
attributes
+ group
+ group
«DataType»
sub-unit
+ segment: EBoolean [1]
1..*
+ sub-unit
1 + sub-unit
1 + sub-unit
sub-unit has to have a source
sub-unit can contain one target
groups can contain units
file can contain or reference a skeleton
files have to contain groups or units
+ skeleton
«DataType»
skeleton
attributes
+ group OR unit
group OR unit
«DataType»
unit [smallest i...
attributes
+ unit
+ unit
«DataType»
source
+ source
«DataType»
target
+ target 0..1
«DataType»
content
+ content
+ content
+ content
+ content
«PrimitiveType»
Text
+ text
«DataType»
em
+ startRef: EString [1]
+ em
«DataType»
sc
+ id: EString [1]
+ sc
«DataType»
ec
+ startRef: EString [1]
+ ec
«DataType»
sm
+ id: EString [1]
+ sm

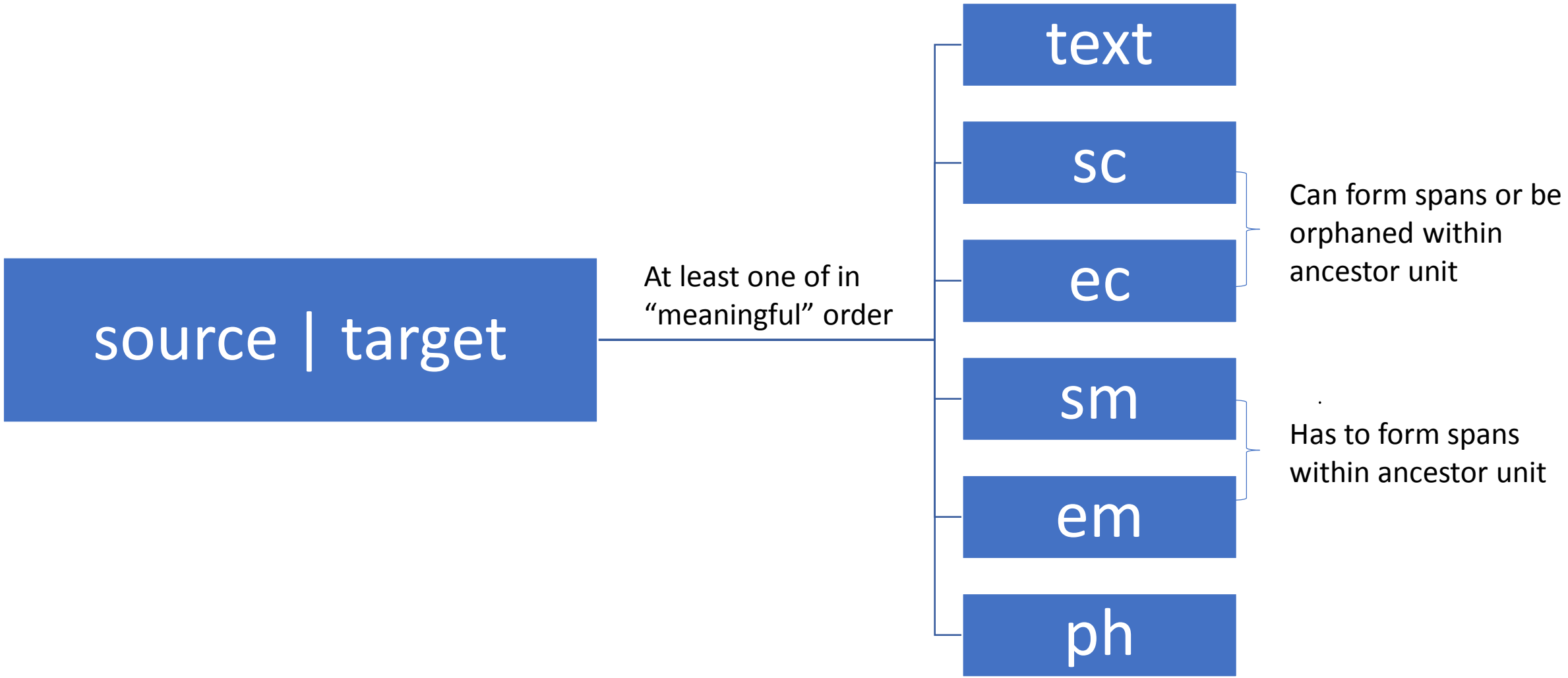
LIOM – static structure



LIOM – transient structure



LIOM – inline content model



XLIFF Top Level Element

```
<xliff xmlns="urn:oasis:names:tc:xliff:document:2.0"
xmlns:uext1="http://example.com/userextension/1.0"
xmlns:uext2="http://example.com/userextension/2.0"
version="2.1" srcLang="en" trgLang="fr">
 <file … >
    <group … > /arbitrary group depth including 0/
       <unit … > [ … /truncated payload structure / … ]
       </unit>
    </group>
 <file>
</xliff>
```

JLIFF Anonymous Top Level Object

```
{
 "jliff": "2.1",
 "@context": {
 "uext1": "http://example.com/userextension/1.0",
 "uext2": "http://example.com/userextension/2.0"
 },
 "srcLang": "en",
 "trgLang": "fr",
 "files": [ … /truncated payload structure / … ]
}
```

"files" "subfiles" "sugbroups" "subunits"

JLIFF Design

General design principles

Avoid XMLism, SGMLism

- such as xml:space, escaping XML illegal characters (cp)
- dropped well formed spanning inline markup

Different approach to structure

- elements & attributes vs objects & properties
- everything needs to be in key value pairs, no mixing character data with markup
- objects are not ordered, so need to use arrays to represent sequences
- JSON is always typed -> modularization harder
- fewer types in JSON -> custom types via patterns

General design principles

Avoid XMLism, SGMLism

- such as xml:space, escaping XML illegal characters (cp)
- dropped well formed spanning inline markup

Different approach to structure

- elements & attributes vs objects & properties
- everything needs to be in key value pairs, **no mixing character data with markup**
- objects are not ordered, so need to use arrays to represent sequences
- JSON is always typed -> modularization harder
- fewer types in JSON -> custom types via patterns

No mixing character data with markup

```
<source>Eat <ph id="1" equiv="[number]"/> eggs for <mrk
id="2">breakfast</mrk>. </source>
```

```
{
"source" : [
{"text" : "Eat "} ,
{"kind" : "ph" , "id" : "1", "equiv" : "[number]"} ,
{"text" : " eggs for "} , {"kind" : "sm" , "id" : "2"} ,
{"text" : "breakfast"} ,
{"kind" : "em" , "id" : "3" , "startRef" : "2"} ,
{"text" : ". "}
]
}
```

xs data types via patterns

NCName

```
"NCName": {
    "description": "XSD NCName type for xml:id interoperability",
    "type": "string",
    "pattern": "^[_A-Za-z][-._A-Za-z0-9]*$"
 }
```

XLIFF mapping to JSON

1. JSON object property names are used to represent XLIFF elements and attributes, with the exception of element sequences that must be represented by JSON arrays;
2. JSON arrays represent element sequences, for example a sequence of <file> elements becomes an array identified by the JSON object property "files": […] where each array item is an anonymous file object that contains an array of "subfiles": […]. Plural forms refer to arrays in JLIFF as a reminder of the structural differences between XML and JSON;
3. To store units and groups that exist within files, JSON object property "subfiles": […] is an array of unit and group objects representing XLIFF <unit> and <group> elements, where an anonymous unit object is identified by a "kind": "unit" and an anonymous group object is identified by "kind": "group";
4. Likewise, "subunits": […] is an array of subunits of a unit object, where a segment subunit is identified as an object with "kind": "segment" and a ignorable object is identified as "kind": "ignorable";
5. A subset of XSD data types that are used in XLIFF are also adopted in the JLIFF schema by defining corresponding JSON schema string types with restricted value spaces defined by regex patterns for NCName, NMTOKEN, NMTOKENS, etc.

XLIFF mapping to JSON

6. Because JSON intrinsically lacks namespace support, qualified JSON object property names represent XLIFF modules, which is purely syntactic to enhance JLIFF document readability and processing. For example, ITS module properties are identified by prefix its_, such as "its_locQualityIssues". Generally undersore "_" is used as the namespace prefix separator for modules (unlike custom namespace based extensions);
7. JLIFF extensions are defined by the optional JSON-LD context "@context": {…} as a property of the anonymous JLIFF root object. [JSON-LD] offers a suitable replacement of XML namespaces required for extension identification and processing. A JSON-LD context is a mapping of prefixes to IRIs. A JSON-LD processor resolves the prefix in an object property name and thus creates a fully qualified name containing the corresponding IRI qualifier;
8. To identify JLIFF documents, the anonymous JLIFF root object has a required property "jliff": "2.0" or "jliff": "2.1";
9. One of the decisions taken in relation to element mappings was not to explicitly support well-formed <pc/> and <mrk/> elements, therefore <mrk/> is mapped to <sm/> and pairs, and <pc/> is mapped to <sc/> and <ec/>pairs. See also LIOM Core.

Other design considerations (1)

While JSON supports the Boolean type values true and false, string based enumerations of yes and no are used in JLIFF to represent XLIFF attributes of the yesNo type.

Reason 1

Omission of a Boolean value is usually associated with the value false by processors and applications.

The XLIFF default of the yesNo attributes is yes.

Absence of an attribute typically indicates permission.

Hence we defined the yes defaults in the JLIFF JSON schema, which would have conflicted with the defaulting behavior of the JSON Boolean type.

Reason 2

The object property canReorder of the ec, ph, and sc objects is a three-valued enumeration with the yes, no, and firstNo values, necessitating the use of a JSON string type with enumeration rather than a JSON Boolean type in JLIFF.

Other design considerations (2)

Almost all JSON schema types defined for JLIFF correspond one-to-one with JSON object property names in JLIFF documents. This design choice reduces efforts to comprehend the JLIFF JSON schema structure for implementers versed in XLIFF. For example, the files property mentioned earlier has a corresponding files type in the JLIFF JSON schema, which is an array that references the file schema type. However, this schema design deviates in one important aspect that is intended to avoid unnecessary duplication of the schema types for the properties subfiles and subgroups that share the same data model. It was decided to introduce the schema type subitems to represent the value space of both subfiles and subgroups. We also added named types to the schema that have no corresponding property name, to break out the JSON structure more clearly. For example, elements is an array of mixed types, which is one of (element-text, element-ph, element-sc, element-ec, element-sm, and element-em in the schema. Note that element-text is a string while the other types are objects.

Other design considerations (3)

JLIFF Modules are an integral part of the JLIFF specification, meaning that all Modules are part of the single JSON schema specification of JLIFF [JLIFFSchema].
This simplifies the processing of Modules by processors, as Modules are frequently used (albeit different subsets based on individual needs and specializations) by implementers.
Extensions are registered externally and included in JLIFF documents as userdata objects.
A userdata object contains one or more extensions as key-value pairs: each extension is identified by a qualified property with an extension-specific JSON value. The prefix of the qualified property of an extension is bound to the IRI of the extension using the JSON-LD @context to identify the unique extension namespace IRI.
Processors that are required to handle extensions should resolve the prefix to the extension's objects fully qualified names as per JSON-LD processing requirements.
Otherwise extensions can be ignored without raising validation failures. This approach offers an extensible and flexible mechanism for JLIFF extensions.
JSON-LD workaround for namespaces great for extensions,
but heavy weight and too complex for modules that are used regularly.
The "context" of modules is considered a shared JLIFF agent knowledge documented in the prose specification rather then being resolved each time when module data need processed, hammering OASIS servers...

XLIFF, LIOM, and JLIFF resources (1)

[ITS20] D. Filip, S. McCance, D. Lewis, C. Lieske, A. Lommel, J. Kosek, F. Sasaki, Y. Savourel, Eds.: Internationalization Tag Set (ITS) Version 2.0. W3C Recommendation, 29 October 2013. W3C. http://www.w3.org/TR/its20/

[JGT] P. Ritchie, JLIFF Graph Tools. Vistatec, 2019. https://github.com/vistatec/JliffGraphTools/commit/74ffde990d8dd6d6d5d3f80d78e76ea8b0dc8736

[JLIFF] D. Filip and R. van Engelen, JLIFF Version 1.0 [wd01]. OASIS, 2018. https://github.com/oasis-tcs/xliff-omos-jliff/commit/7e63e0d766bb7394f9dccaa93d7fa54bf1a394d3

[JLIFFSchema] R. van Engelen, JLIFF Version 1.0, JSON Schema [wd01]. OASIS, 2018. https://github.com/oasis-tcs/xliff-omos-jliff/commit/2ed3b57f38548600f1261995c466499ad0ade224/>

[JSON-LD] M. Sporny, G. Kellogg, M. Lanthaler, Eds. JSON-LD 1.0, A JSON-based Serialization for Linked Data W3C Recommendation 16 January 2014. https://www.w3.org/TR/2014/REC-json-ld-20140116//>

[L10nStandards] D. Filip: Localization Standards Reader 4.0 [v4.0.1], Multilingual, vol. 30, no. 1, pp. 59–73, Jan/Feb-2019. https://magazine.multilingual.com/issue/jan-feb-2019dm/localization-standards-reader-4-0/

[LIOM] D. Filip, XLIFF 2 Object Model Version 1.0 [wd01]. OASIS, 2018. https://github.com/oasis-tcs/xliff-omos-om/commit/030828c327998e7c305d9be48d7dbe49c8ddf202/>

XLIFF, LIOM, and JLIFF resources (2)

[XLIFF20] T. Comerford, D. Filip, R. M. Raya, and Y. Savourel, Eds.: XLIFF Version 2.0. OASIS Standard, 05 August 2014. OASIS. http://docs.oasis-open.org/xliff/xliff-core/v2.0/os/xliff-core-v2.0-os.html

[XLIFF21] D. Filip, T. Comerford, S. Saadatfar, F. Sasaki, and Y. Savourel, Eds.: XLIFF Version 2.1. OASIS Standard, 13 February 2018. OASIS http://docs.oasis-open.org/xliff/xliff-core/v2.1/os/xliff-core-v2.1-os.html

[XLIFFEMBP] D. Filip and J. Husarčík, Eds., XLIFF 2 Extraction and Merging Best Practice, Version 1.0. Globalization and Localization Association (GALA) TAPICC, 2018. https://galaglobal.github.io/TAPICC/T1/WG3/rs01/XLIFF-EM-BP-V1.0-rs01.xhtml/>

[XLIFFglsTBXBasic] J. Hayes, S. E. Wright, D. Filip, A. Melby, and D. Reineke, Interoperability of XLIFF 2.0 Glossary Module and TBX-Basic, Localisation Focus, vol. 14, no. 1, pp. 43–50, Apr. 2015.https://www.localisation.ie/resources/publications/2015/260

[XLIFFRender] D. Filip and J. Husarčík, Modification and Rendering in Context of a Comprehensive Standards Based L10n Architecture, Proceedings ASLING Translating and the Computer, vol. 40, pp. 95–112, Nov. 2018. https://www.asling.org/tc40/wp-content/uploads/TC40-Proceedings.pdf

Thanks a million!

Q & A

@merzbauer

@philinthecloud

@vEngelenRobert