XML-Based Format of PDT 2.0 Data

Petr Pajas

ÚFAL, MFF, Charles University, Prague

November 29, 2006

Introduction

In PDT 2.0 and some related annotation projects, we:

- use several (interlinked) annotation levels
- introduce various types of annotation: linear, tree-structured, ...
- apply one annotation schema to different languages
- sometimes use project-specific variants/flavors of existing annotations or annotation schemas
- use something called *"annotation dictionaries"*

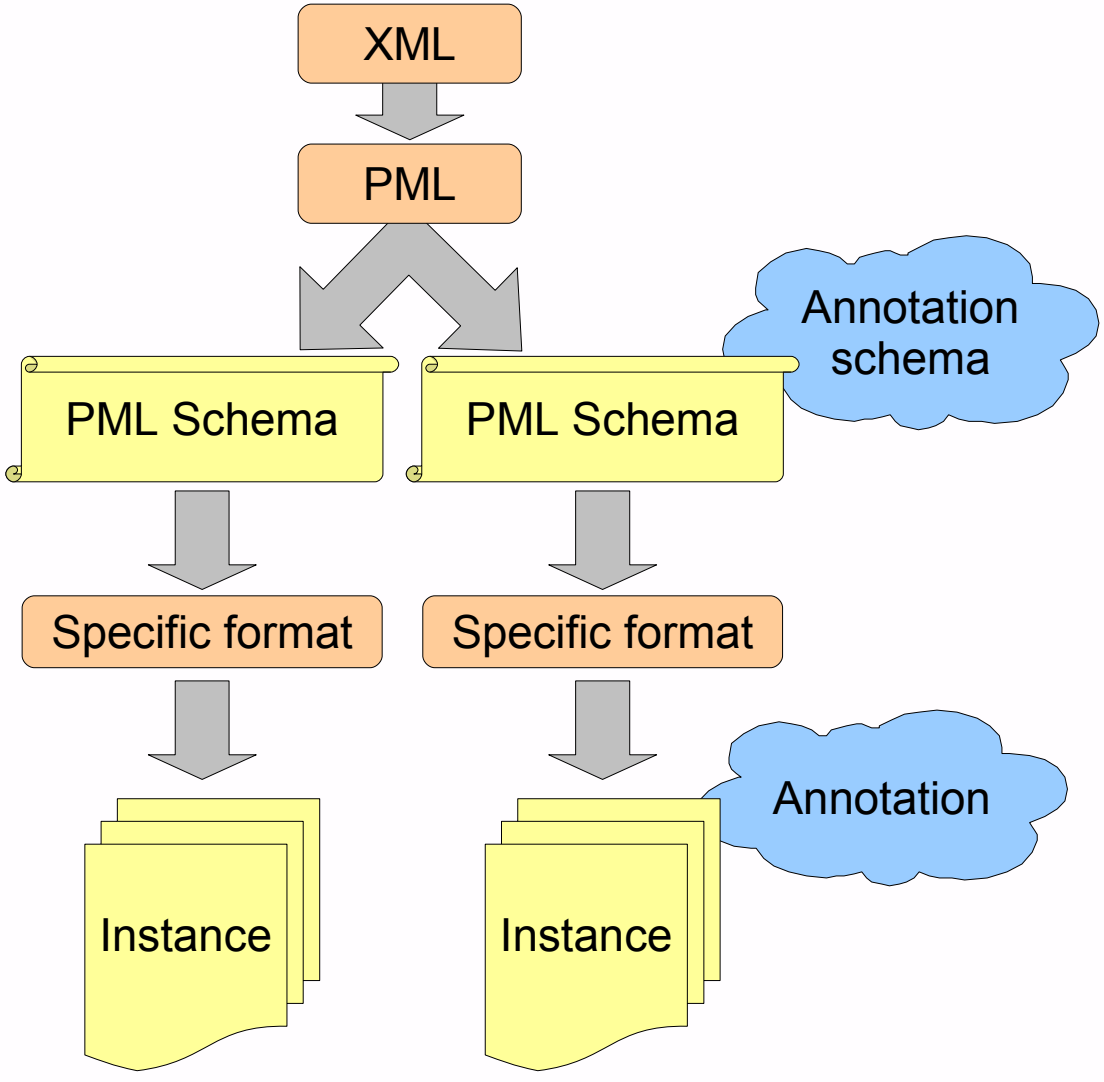
Our goal is to represent all data resulting from these projects coherently, in a uniform manner.

- We need a meta-format (from which we may derive specific formats for our annotation schemas and layers)
- XML is known to be an excellent meta-format, but still too generic to ensure uniformity of representation
- thus, we need another layer of abstraction, between the *"generic"* (XML) and the *"specific"* (data-specific format)

Requirements

- Uniformity of representation
- Stand-off annotation principles
- Unified cross-referencing and linking system
- Linearity and structure
- Structured attributes
- Handling ambiguity
- Human-readability
- Extensibility
- XML based
- Description language

PML – Prague Markup Language



PML – Prague Markup Language (2)

Every annotation layer has a PML-based format specified by something called *PML schema*.

- PML schema
  - *A data format description language.*
  - *Formalizes the "annotation schema" for a particular layer*
  - *Defines the annotation structure*
  - *Assigns PML roles to certain pieces of annotation.*
- annotation structure

  data structure built from abstract data types, such as:
  atomic values, attribute-value structures, lists, alternatives, etc.
- PML role

  identifies a piece of annotation as a bearer of some additional higher-level property, such as being a node of a tree, being a unique identifier, etc.

XML documents conforming to a PML schema are called PML instances.

PML - Prague Markup Language (3)

Processing PML

PML instances can be processed using:

- arbitrary XML-oriented tools (based on DOM, XPath, etc.)
- format-specific tools with hard-wired knowledge about the XML vocabulary of a particular PML-based format
- intelligent generic tools aware of PML-schema:
  - data type declarations $\longrightarrow$ optimal in-memory representation (data binding)
  - role assignment $\longrightarrow$ adequate way of presenting the annotation to the user and providing some extra features (indexes, etc.)

Validation

- using conventional validators for XML such as `xmllint` or `jing`. (PML schema translates to a Relax NG – via XSLT)

Data types – overview

- **Atomic (simple)**
  - cdata
  - enumerated
  - constant
- **Complex**
  - structures
  - containers
  - lists
  - alternatives
  - sequences

Data types – atomic (1)

cdata type

- literal character-based content
- no explicitly marked internal structure
- value format can be restricted
- current formats include most W3C Schema simple types

PML schema declaration

```
<cdata format="token"/>
<cdata format="float"/>
<cdata format="positiveInteger"/>
<cdata format="long"/>
<cdata format="date"/>
<cdata format="time"/>
<cdata format="any"/>
...
```

instance example

```
<...>hallo234</...>
<...>12.7843E-2</...>
<...>17</...>
<...>-9223372054775808</...>
<...>1999-05-31</...>
<...>13:20:00.000</...>
<...>ar6!7rar¥ d@t&</...>
...
```

Data types – atomic (2)

enumerated type

Literal values from a fixed finite set.

PML schema declaration

```
<type name="boolean.type">
  <choice>
    <value>TRUE</value>
    <value>FALSE</value>
  </choice>
</type>
```

instance example

```
<...>TRUE</...>
```

constant

A fixed constant value.

PML schema declaration

```
<type name="root-node.type">
  <structure>
    <member name="node-type">
      <constant>root</constant>
    </member>
    ...
  </structure>
</type>
```

instance example

```
<...>
  <!-- no need to use explicitly -->
  ...
</...>
```

Data types – complex (1)

Attribute-value structure (AVS)

- consists of a fixed set of attribute-value pairs (called members)
- values can be atomic or complex
- value type of each member is declared in the PML schema
- members can be optional or required
- atomic-valued members can be rendered as attributes

PML schema declaration

```
<structure>
  <member name="id" as_attribute="1">
    <cdata format="ID" role="#ID"/>
  </member>
  <member name="form" required="1">
    <cdata format="token"/>
  </member>
  <member name="lemma">
    <cdata format="token"/>
  </member>
  <member name="tag"
       type="tagset.type"/>
</structure>
```

example instances

```
<... id="m-23">
  <form>walking</form>
  <lemma>walk-1</lemma>
  <tag>VBG</tag>
</...>

<...id="m-24">
  <form>away</form>
</...>
```

Data types – complex (2)

container

- attaches attributes to a single central value (content)
- attributes are name-value pairs with atomic values
- the set of attributes is declared in the schema
- content can be atomic or complex other than container or structure

PML schema declaration

```
<type name="word.type">
  <container>
    <attribute name="id"
               role="#ID">
      <cdata format="ID"/>
    </attribute>
    <cdata format="token"/>
  </container>
</type>
```

instance example

```
<... id="w-23">Walking</...>
```

Data types – complex (3)

list

- aggregate zero or more values of a certain type
- can be ordered or unordered (sets).
- reserved tag <LM> used for list values
- single <LM> can sometimes be ommitted (list folding)

PML schema declaration

```
<type name="sent.type">
  <list ordered="1" type="word.type">
</type>
```

example instance (several values)

```
<...>
  <LM id="w-34">Flies</LM>
  <LM id="w-35">like</LM>
  <LM id="w-36">an</LM>
  <LM id="w-37">arrow</LM>
  <LM id="w-38">.</LM>
</...>
```

example instance (single value)

```
<...>
  <LM id="w-34">Flies</LM>
</...>
```

can fold into:

```
<... id="w-34">Flies</...>
```

Data types – complex (3)

alternative

- aggregates values in parallel, i.e. as alternative annotations.
- reserved tag <AM> used for alternative values
- single <AM> can be ommitted (folding)

PML schema declaration

```
<type name="morph.type">
  <alt type="m.type">
</type>
```

instance example

```
<...>
  <AM id='m-34'>
    <form>flies</form>
    <lemma>fly-1</lemma>
    <tag>VBZ</tag>
  </AM>
  <AM id='m-34A'>
    <form>flies</form>
    <lemma>fly-2</lemma>
    <tag>NNS</tag>
  </AM>
</...>
```

Data types – complex (4)

sequence

- aggregates values of several different types (unlike lists which are type-homogeneous)
- consists of zero or more name-value pairs (called *elements*)
- element's name determines its value type
- elements may be arbitrarily repeated
- reg.exp.-like pattern may restrict element order

PML schema declaration

```
<type name="chapter.type">
  <sequence
        content_pattern="para*, sect+">
    <element name="para"
        type="paragraph.type"/>
    <element name="sect"
        type="section.type">
  </sequence>
</type>
```

instance example

```
<...>
  <para>
    In this chapter...
  </para>
  <sect>...</sect>
  <sect>...</sect>
  <sect>...</sect>
</...>
```

PML roles

- ID

  assigned to members or attributes uniquely identifying an AVS or a container within a PML instance
- KNIT

  assigned to links suitable for merging two annotation layers in such a way that the referred object is embedded into the referring object
- TREES

  marks lists or sequences of dependency or constituency trees
- NODE

  identifies nodes of dependency or constituency trees
- CHILDNODES

  identifies the member of a node (of a dependency or constituency tree) containing the list of its child nodes
- ORDER

  used to identify numerical values which determine a total ordering on a tree

Links

- Currently only ID-based links are supported
- Other types of links represented in PML on a per-application basis

ID-based links:

- Cross-references within a single PML instance
- Links to other instances
  Typically many links to only few target instances

  Therefore PML links have two parts:
  - the specification of the target instance – a label (ID) associated with the target instance in the referring instance header
  - the ID of the target object

Links - examples

Associating target URLs with IDs in the referring instance header

```
<references>
  <reffile id="a" href="doc73.a"/>
  <reffile id="v" href="http://mysite/vallex.xml"/>
</references>
```

Examples of ID-based links

```
<coref.rf>t-node-232</coref.rf>
```
— link to the same file

```
<val_frame.rf>v#f2234</val_frame.rf>
```
— link to http://mysite/vallex.xml

```
<lex.rf>a#doc73-w5</lex.rf>
```
— links to doc73.a

```
<aux.rf>
  <LM>a#doc73-w3</LM>
  <LM>a#doc73-w4</LM>
</aux.rf>
```

Links to non-PML data

Currently no guidelines.

Example of possible PML representation of pointers to an audio file:

```
<references>
  <reffile id="au1" href="spk1_129.ogg"/>
</references>
...
<w id="w-12941">
  <token>_SIL_</token>
  <audio>
    <time_start>600000</time_start>
    <time_end>4700000</time_end>
    <file.rf>au1</file.rf>
  </audio>
</w>
```

Extendibility

PML allows to upgrade or extend the data model in many ways while retaining the XML representation of the existing data.

- alternative folding
- list folding
- container transparency
- structure to sequence conversion
- structure and sequence extendibility

Extendibility – singleton alternative minimization

Data fields of any type may be upgraded to allow ambiguity:

PML schema (no ambiguity allowed)

```
<cdata format="any"/>
```

Data (value)

```
<afun>Adv</afun>
```

PML schema (allows ambiguity)

```
<alt>
  <cdata format="any"/>
</alt>
```

Data (unambiguous value – **same**)

```
<afun>Adv</afun>
```

Data (ambiguity)

```
<afun>
  <AM>Adv</AM>
  <AM>Obj</AM>
</afun>
```

Extendibility – singleton list minimization

By the „uniformity of representation“ principle, we get the same for lists.

Upgrading a value to a lists of values:

PML schema (one-to-one link)

```
<cdata format="PMLREF"/>
```

Data (link)

```
<m.rf>m-34</m.rf>
```

PML schema (one to many)

```
<list>
  <cdata format="PMLREF"/>
</list>
```

Data (folded singleton list)

```
<m.rf>m-34</m.rf>
```

Data (multiple values)

```
<m.rf>
  <LM>m-34</LM>
  <LM>m-34A</LM>
</m.rf>
```

Extendibility – container transparency

Add attribute annotation to flat data, lists, etc.

PML schema (flat value)

```
<cdata format="string"/>
```

Data (flat value)

```
<phrase>in the red</phrase>
```

PML schema (annotated value)

```
<container>
  <attribute name="category"
        type="phr-category.type"/>
  <attribute name="area"
        type="phr-area.type"/>
  <cdata format="string"/>
</container>
```

Data (container w/o attributes)

```
<phrase>in the red</phrase>
```

Data (adding attributes)

```
<phrase category="idiom" area="financial">in the red</phrase>
```

Modularization

Allows to easily derive one PML schema from another.

- revision numbering

  PML schemas can be assigned revision numbers of the form `X.Y.Z...`

- `<import>` instruction

  Copies type declarations from a different PML schema with revision restrictions.

- `<derive>` instruction

  Derive types from a previously declared or imported AVS structure, container, sequence, or enumerated type.

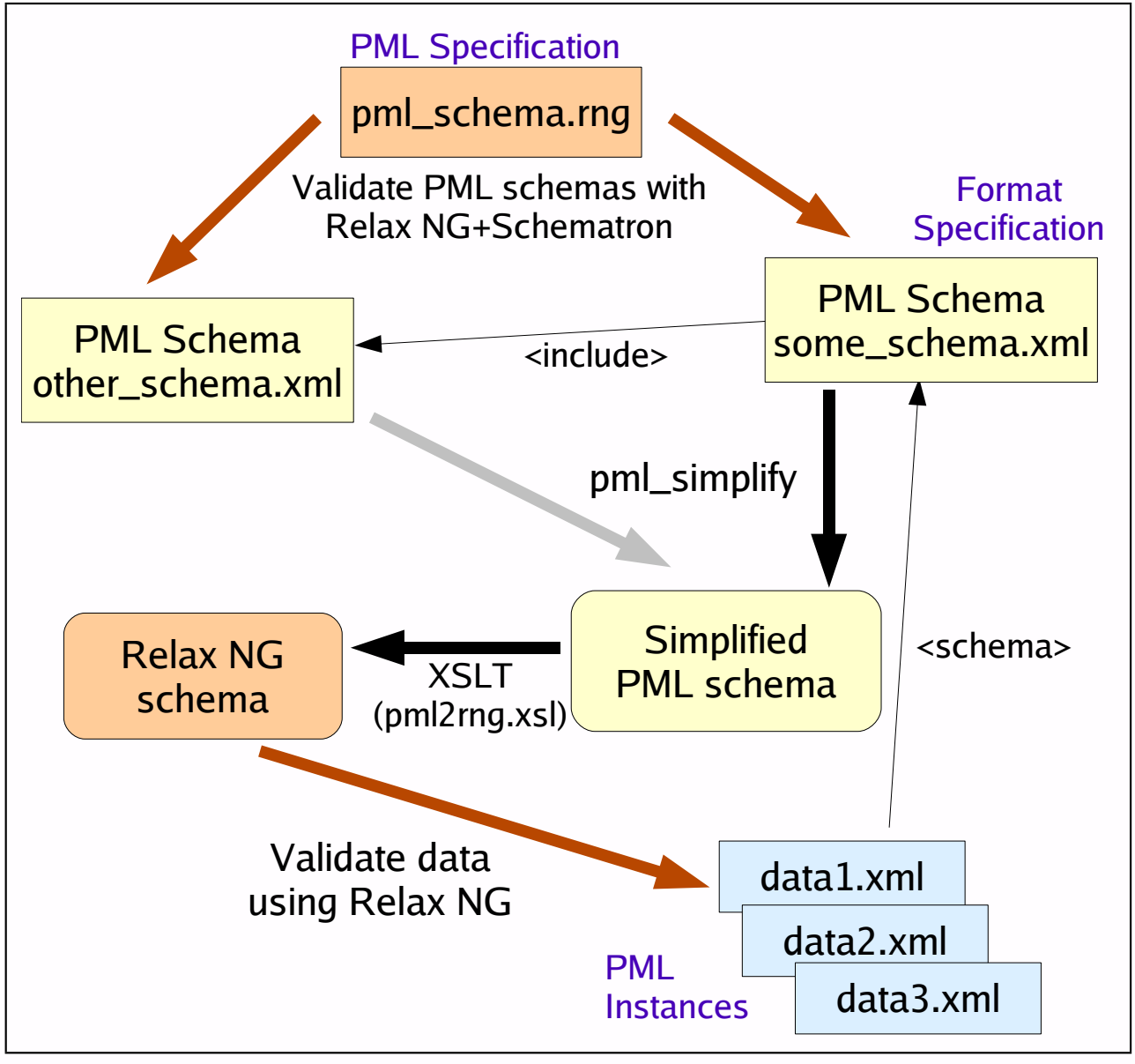
- simplified PML schemas

  `pml_simplify` - *a PML schema preprocessor,*
  resolves all `<import>` *and* `<derive>` *instructions*

  Useful e.g. before XSLT 1.0 transformations.

Validation

- Conformance of PML schemas to the spec can be verified via Relax NG and Schematron.
- Simplified PML schemas can be XSLT-transformed into Relax NG for data validation purposes.
- Many validators for Relax NG exist.
- The script `validate_pml` automates all the necessary steps.



Why Relax NG + Schematron

...and not W3C XML schemas or DTD?

- First, DTD? Really? You must be kidding, right?
- W3C schemas support attributes as poorly as DTDs.
- Relax NG has way more flexible structural support than W3C schemas.
- Extendibility of W3C schemas is questionable.
- W3C schemas are very hard to implement, while Relax NG are basically automata.
- DocBook, OpenOffice, SVG, or XHTML use Relax NG.
- Relax NG can use the best of W3C schemas, i.e. the simple types.
- ISO Schematron handles best non-structural constraints.
- In general, for W3C XML Schemas vs. Relax NG read

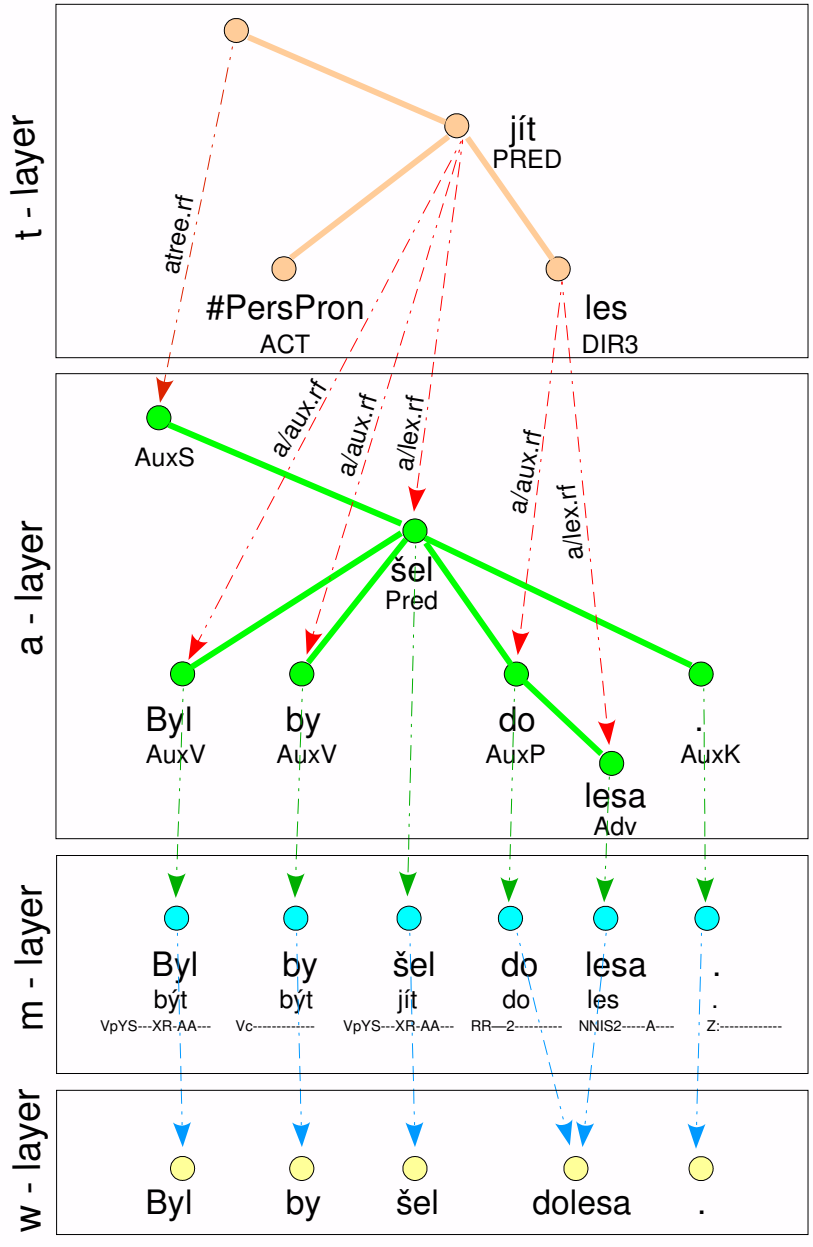
  `http://www.imc.org/ietf-xml-use/mail-archive/msg00217.html`

Layering

- One PML schema per annotation layer.
- The annotation layers are interconnected by PML links.
- If the annotation structure allows it, the `KNIT` role can be used to allow for merging annotation layers.

For example:

- PDT 2.0 uses the following PML schemas: `wdata_schema.xml`, `mdata_schema.xml`, `tdata_schema.xml`, and `adata_schema.xml`.
- via `KNIT`, w-layer can be merged into m-layer, m-layer can be merged into a-layer.



Intermediate layers

- Different machine processing strategies may have different views on what compounds a single layer.

  A typical example is tokenization and sentence-break segmentation (usually done before tagging, but for some languages, such as Arabic, it may be reasonable to do all at the same time).

- Intermediate layers can also result from incomplete manual annotation as fragments of the final annotation layers.

  PML schemas for annotation formats can be derived by PML modularization support.

PML representation of PDT 2.0 dependency layers

a-layer instance consists of:

- meta data (annotation info)
- a list of trees (each by a technical root node structure)

Two types of nodes (both structures)

- technical root (a-root.type)
- analytical node (a-node.type)

Each node carries:

- member `ord` providing tree ordering (role `#ORDER`)
- a list of child nodes (member `children` with role `#CHILDNODES`)
- pointers to m-layer (with role `#KNIT` on a-node.type)
- analytical function `afun` (const. `AuxS` on root, enumerated elsewhere)
- coordination/apposition and parenthesis membership flags

t-layer layer follows the same pattern for representing ordered dependency trees, only t-node structures carry much richer annotation and some extra relational stuff, like co-reference links and quotation sets.

PML schema for a-layer (1)

```
<pml_schema
    xmlns="http://ufal.mff.cuni.cz/pdt/pml/schema/" version="1.1">
  <revision>1.0.3</revision>
  <description>PDT 2.0 analytical trees</description>
  <reference name="mdata" readas="dom"/>
  <reference name="wdata" readas="dom"/>

  <import schema="mdata_schema.xml" type="m-node.type"
    minimal_revision="1.0.3"/>
  <import schema="mdata_schema.xml" type="bool.type"/>

  <derive type="m-node.type">
    <structure name="m-node">
      <member name="id" as_attribute="1" role="#ID" required="1">
        <cdata format="PMLREF"/>
      </member>
    </structure>
  </derive>
...
```

PML schema for a-layer (2)

```
...
<root name="adata" type="a-adata.type"/>

<type name="a-adata.type">
  <structure>
    <member name="meta" required="0" type="a-meta.type"/>
    <member name="trees" role="#TREES" required="1">
      <list type="a-root.type" ordered="1"/>
    </member>
  </structure>
</type>

<type name="a-meta.type">
  <structure>
    <member name="annotation_info">
      <structure name="a-annotation-info">
        <member name="version_info"><cdata format="any"/></member>
        <member name="desc"><cdata format="any"/></member>
      </structure>
    </member>
  </structure>
</type>
...
```

PML schema for a-layer (3)

```
...
<type name="a-root.type">
  <structure role="#NODE" name="a-root">
    <member name="id" role="#ID" as_attribute="1" required="1">
      <cdata format="ID"/>
    </member>
    <member name="s.rf"><cdata format="PMLREF"/></member>
    <member name="afun"><constant>AuxS</constant></member>
    <member name="ord" role="#ORDER" required="1">
      <cdata format="nonNegativeInteger"/>
    </member>
    <member name="children" role="#CHILDNODES">
      <list type="a-node.type" ordered="1"/>
    </member>
  </structure>
</type>
...
```

PML schema for a-layer (4)

```
...
<type name="a-node.type">
  <structure role="#NODE" name="a-node">
    <member name="id" role="#ID" as_attribute="1" required="1">
      <cdata format="ID"/>
    </member>
    <member name="m.rf" role="#KNIT" type="m-node.type">
      <cdata format="PMLREF"/>
    </member>
    <member name="afun" type="a-afun.type" required="1"/>
    <member name="is_member" type="bool.type"/>
    <member type="bool.type" name="is_parenthesis_root"/>
    <member name="ord" role="#ORDER" required="1">
      <cdata format="nonNegativeInteger"/>
    </member>
    <member name="children" role="#CHILDNODES">
      <list type="a-node.type" ordered="1"/>
    </member>
  </structure>
</type>
...
```

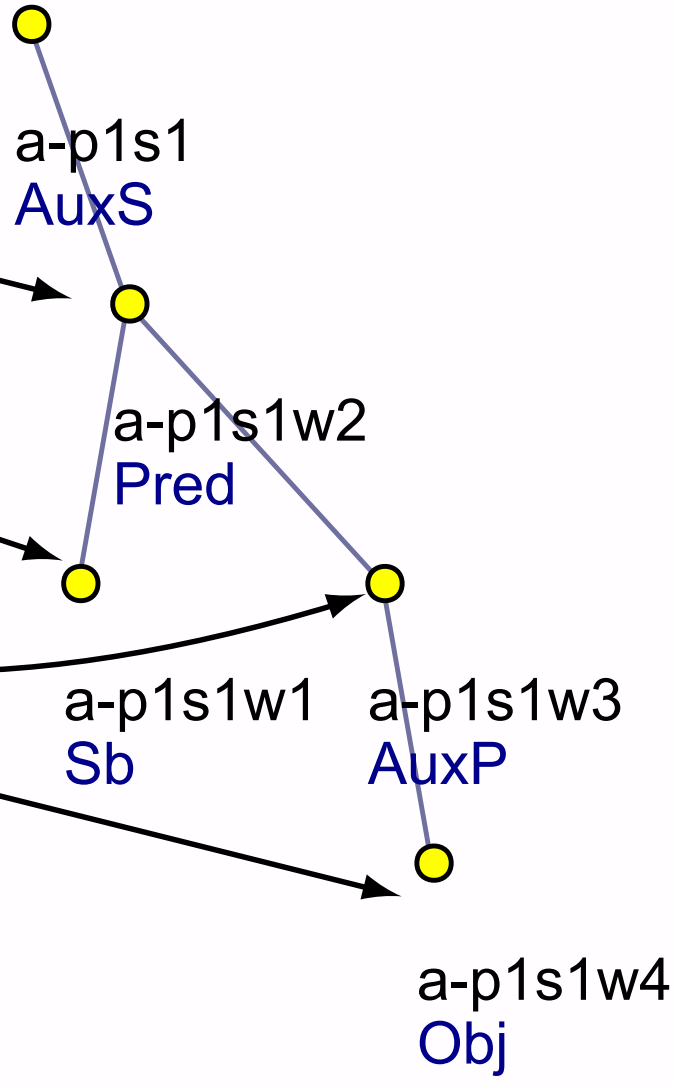
Sample a-layer instance

```
<adata xmlns="http://ufal.mff.cuni.cz/pdt/pml/">
 <head>
  <schema href="adata_schema.xml" />
  <references>
   <reffile id="m" name="mdata" href="sample4.m.gz" />
   <reffile id="w" name="wdata" href="sample4.w.gz" />
  </references>
 </head>
 <meta>
  <annotation_info>
   <desc>Manual annotation</desc>
  </annotation_info>
 </meta>
 <trees>
  <LM id="a-p1s1">...</LM>
  <LM id="a-p1s2">...</LM>
  ...
 </trees>
```

Sample PDT 2.0 instance (a-layer)

a-layer

```
<LM id="a-p1s1">
<s.rf>m#m-p1s1</s.rf>
<ord>0</ord>
<children>
  <LM id="a-p1s1w2">
    <m.rf>m#m-p1s1w2</m.rf>
    <afun>Pred</afun>
    <ord>2</ord>
    <children>
      <LM id="a-p1s1w1">
        <m.rf>m#m-p1s1w1</m.rf>
        <afun>Sb</afun>
        <ord>1</ord>
      </LM>
      <LM id="a-p1s1w3">
        <m.rf>m#m-p1s1w3</m.rf>
        <afun>AuxP</afun>
        <ord>3</ord>
        <children id="a-p1s1w4">
          <m.rf>m#m-p1s1w4</m.rf>
          <afun>Obj</afun>
          <ord>4</ord>
        </children>
      </LM>
    </children>
  </LM>
</children>
</LM>
```



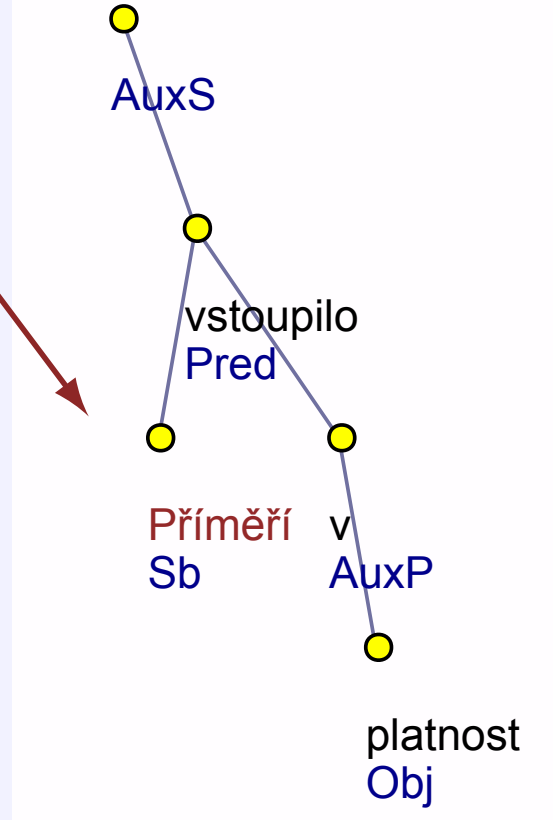
Sample PDT 2.0 instance (a+m-layer)

a-layer

```
<LM id="a-p1s1">
<s.rf>m#m-p1s1</s.rf>
<ord>0</ord>
<children>
  <LM id="a-p1s1w2">
    <m.rf>m#m-p1s1w2</m.rf>
    <afun>Pred</afun>
    <ord>2</ord>
    <children>
      <LM id="a-p1s1w1">
        <m.rf>m#m-p1s1w1</m.rf>
        <afun>Sb</afun>
        <ord>1</ord>
      </LM>
      <LM id="a-p1s1w3">
        <m.rf>m#m-p1s1w3</m.rf>
        <afun>AuxP</afun>
        <ord>3</ord>
        <children id="a-p1s1w4">
          <m.rf>m#m-p1s1w4</m.rf>
          <afun>Obj</afun>
          <ord>4</ord>
        </children>
      </LM>
    </children>
  </LM>
</children>
</LM>
```

m-layer

```
<s id="m-p1s1">
<m id="m-p1s1w1">
  <src.rf>manual</src.rf>
  <w.rf>w#w-p1s1w1</w.rf>
  <form>Příměří</form>
  <lemma>příměří</lemma>
  <tag>NNNS1-----A----</tag>
</m>
<m id="m-p1s1w2">
  <src.rf>manual</src.rf>
  <w.rf>w#w-p1s1w2</w.rf>
  <form>vstoupilo</form>
  <lemma>vstoupit_:W</lemma>
  <tag>VpNS---XR-AA---</tag>
</m>
<m id="m-p1s1w3">
  <src.rf>manual</src.rf>
  <w.rf>w#w-p1s1w3</w.rf>
  <form>v</form>
  <lemma>v-1</lemma>
  <tag>RR--4----------</tag>
</m>
<m id="m-p1s1w4">
  <src.rf>manual</src.rf>
  <w.rf>w#w-p1s1w4</w.rf>
  <form>platnost</form>
  <lemma>platnost_^(*3ý)</lemma>
  <tag>NNFS4-----A----</tag>
</m>
</s>
```



Sample PDT 2.0 instance (a+m+w-layer)

a-layer

```
<LM id="a-p1s1">
<s.rf>m#m-p1s1</s.rf>
<ord>0</ord>
<children>
  <LM id="a-p1s1w2">
    <m.rf>m#m-p1s1w2</m.rf>
    <afun>Pred</afun>
    <ord>2</ord>
    <children>
      <LM id="a-p1s1w1">
        <m.rf>m#m-p1s1w1</m.rf>
        <afun>Sb</afun>
        <ord>1</ord>
      </LM>
      <LM id="a-p1s1w3">
        <m.rf>m#m-p1s1w3</m.rf>
        <afun>AuxP</afun>
        <ord>3</ord>
        <children id="a-p1s1w4">
          <m.rf>m#m-p1s1w4</m.rf>
          <afun>Obj</afun>
          <ord>4</ord>
        </children>
      </LM>
    </children>
  </LM>
</children>
</LM>
```

m-layer

```
<s id="m-p1s1">
<m id="m-p1s1w1">
  <src.rf>manual</src.rf>
  <w.rf>w#w-p1s1w1</w.rf>
  <form>Příměří</form>
  <lemma>příměří</lemma>
  <tag>NNNS1-----A----</tag>
</m>
<m id="m-p1s1w2">
  <src.rf>manual</src.rf>
  <w.rf>w#w-p1s1w2</w.rf>
  <form>vstoupilo</form>
  <lemma>vstoupit_:W</lemma>
  <tag>VpNS---XR-AA---</tag>
</m>
<m id="m-p1s1w3">
  <src.rf>manual</src.rf>
  <w.rf>w#w-p1s1w3</w.rf>
  <form>v</form>
  <lemma>v-1</lemma>
  <tag>RR--4----------</tag>
</m>
<m id="m-p1s1w4">
  <src.rf>manual</src.rf>
  <w.rf>w#w-p1s1w4</w.rf>
  <form>platnost</form>
  <lemma>platnost_^(*3ý)</lemma>
  <tag>NNFS4-----A----</tag>
</m>
</s>
```

w-layer

```
<w id="w-p1s1w1">
  <token>Příměří</token>
</w>
<w id="w-p1s1w2">
  <token>vstoupilo</token>
</w>
<w id="w-p1s1w3">
  <token>v</token>
</w>
<w id="w-p1s1w4">
  <token>platnost</token>
</w>
```

Sample PDT 2.0 instance (a+m+w-layers knitted)

a-layer

```
<LM id="a-p1s1">
<s.rf>m#m-p1s1</s.rf>
<ord>0</ord>
<children>
  <LM id="a-p1s1w2">
    <m.rf>m#m-p1s1w2</m.rf>
    <afun>Pred</afun>
    <ord>2</ord>
    <children>
      <LM id="a-p1s1w1">
        <m.rf>m#m-p1s1w1</m.rf>
        <afun>Sb</afun>
        <ord>1</ord>
      </LM>
      <LM id="a-p1s1w3">
        <m.rf>m#m-p1s1w3</m.rf>
        <afun>AuxP</afun>
        <ord>3</ord>
        <children id="a-p1s1w4">
          <m.rf>m#m-p1s1w4</m.rf>
          <afun>Obj</afun>
          <ord>4</ord>
        </children>
      </LM>
    </children>
  </LM>
</children>
</LM>
```

m-layer

```
<s id="m-p1s1">
<m id="m-p1s1w1">
</m>
</s>
```

w-layer

After knitting

```
<LM id="a-p1s1w1">
  <m id="m#m-p1s1w1">
    <src.rf>manual</src.rf>
    <w id="w#w-p1s1w1">
      <token>Příměří</token>
    </w>
    <form>Příměří</form>
    <lemma>příměří</lemma>
    <tag>NNNS1-----A----</tag>
  </m>
  <afun>Sb</afun>
  <ord>1</ord>
</LM>
```

Technical issues with multi-layered annotations in PML

In PDT 2.0, the full annotation of each document comprises of four PML instances (one per layer), which contain references to one another.

This raises some technical obstacles to the users and tool implementators:

- PML instances are not copy-safe (one cannot simply rename e.g. the m-layer instance without fixing the reference to it in the a-layer instance)
- PDT 2.0 data are distributed in gzip-compressed forms. Same problem: decompressing removes the `.gz` suffix, which renames the instance. Again, references have to be fixed.

Future development

- foreign namespaces

  allow XML data from non-PML namespaces within PML instance (MathML, XLink, RDF,...)

- meta-data

  uniform representation of meta-data (via RDF)

- schema annotation

  similar to what W3C schema has

- new roles

  Current set of roles doesn't cover all situations, e.g. lexicons

- guidelines for more types of links

  foreign XML, other media (text, audio, video, graphics, . . .)

- automatic data-binding and API generation

  translate PML schemas into a ready-to-use library with optimal in-memory representation, validation, parser, serializer, indexing, etc.

PML homepage

`http://ufal.mff.cuni.cz/jazz/PML`

- PML spec (with many examples)
- Relax NG schema for PML schema
- tools (simplification, validation, ...)
- updated PDT 2.0 schemas
- links