Naming Convention for Web-Services in Platform for Automatic, Normalized Annotation and Cost-Effective Acquisition of Language Resources for Human Language Technologies (PANACEA) ILC-CNR

Del Gratta Riccardo

2011-06-08

Contents

1 The registry Structure **3**
1.1 Tags and ontology-like classification . . . 3

2 Naming the PANACEA Pipeline **4**
2.1 Naming convention for Corpus Acquisition Component . . . 4
2.1.1 Bilingual Parallel Corpus Acquisition Component . . . 4
2.1.2 Bilingual Comparable Corpus Acquisition Components . . . 6
2.2 Monolingual Corpus Acquisition Components . . . 7
2.3 Naming convention for Clean-Up and Normalization Components . . . 8
2.4 Naming convention for Text Processing Components . . . 9
2.4.1 Text Processing Component (TPC) caveat . . . 9
2.5 Naming convention for Format Converter . . . 10
2.5.1 FCs caveat . . . 10
2.5.2 Mono Converters . . . 10
2.5.3 Multi Converters . . . 11

3 Acronyms **12**

Bibliography **12**

List of Tables

2.1 Naming convention for parallel CACs 5
2.2 Naming convention rule for parallel CACs 5
2.3 Naming convention for comparable CACs 6
2.4 Naming convention rule for comparable CACs 6
2.5 Naming convention rule for CACs 6
2.6 Naming convention rule for (monolingual) CACs 7
2.7 Naming convention for CNCs 8
2.8 Naming convention rule for CNCs 8
2.9 Naming convention for TPCs 10
2.10 Naming convention rule for TPCs where tasks are grouped . . . 10
2.11 Naming convention for FCs 11
2.12 Naming convention rule for FCs where conversions are grouped . 11

Chapter 1

The registry Structure

The PANACEA registry will contain several web services which are pretty much grouped into "service" categories. These categories, in the registry language, are "tags" and can be added to web services when these are registered/updated. These tags can be modified even by people which use the services and not only by "registrators/creators" of services themselves.
In conclusion, one service can have more than one tag, and tags can be changed. This last aspect, certainly, will not help to implement a stable and persistent classification of proposed services.

1.1 Tags and ontology-like classification

Thanks to comments to our first draft proposal, we see that the PANACEA registry can use ontology tags. For example, if we want to use the tag "retrieving" from the *MyGrid* ontology, see *http://www.mygrid.org.uk/ontology*, we only have to write the following text as the tag:
http://www.mygrid.org.uk/ontology#retrieving
The use of ontologies is quite interesting for web service and workflow annotation in PANACEA, but it is NOT in the platform version 2.
Briefly, our bootstrap in using ontology is (will be) simply based on the following aspects:

- Customization of tags. In this case, we can create a sort of "ontology", so that each service can be enriched with these ontological properties;
- The use of tags as "ontological" properties. The ontological class being **Web Service**;
- Closed vocabularies to limit the possible values of tags.

The basic idea is to simplify the web service(s) naming convention by using this tag-system based classification. As an example, if a person is interested in searching services in the registry, maybe (s)he wants not to be bored with names such as *convert_FreeLing2TO_Up2MorphoSyn*... But if the "sort of ontology" is well done (and registering people is fine too), (s)he can browse the ontology, extract the services which expose the "format converter" tag and have a look to the tags to see the most suitable services.

Chapter 2

Naming the PANACEA Pipeline

According to deliverable $D4.1$, cfr [1], the PANACEA pipeline consists of the following components:

- Corpus Acquisition Component;
- Clean Up and Normalization Component;
- Text Processing Component;
- Possible WorkFlow.

2.1 Naming convention for Corpus Acquisition Component

The first classification is between *bilingual* and *monolingual* CACs. In the set of bilingual CACs, a further distinction is made for managing *parallel* and *comparable* corpora. In the set of monolingual CACs, the vocabulary of allowed languages is limited to $\{it, en, el, es, fr, de\}$.

2.1.1 Bilingual Parallel Corpus Acquisition Component

We propose two distinct naming conventions for such services:

Using *parallel* as a tag In this scenario the “ontology” is as follows:

```
Main Class: Web Service
    Type: CAC
        CAC Type: parallel CAC
    Name: CAC_<tool>_<lan1>_<lan2>_<domain>
```

Using *parallel* in the name In this case the *parallel* property is not inherited directly from the classification, even if the tag can be left:

```
Main Class: Web Service
    Type: CAC
        (CAC Type: parallel CAC)
    Name: CAC_parallelCAC_<tool>_<lan1>_<lan2>[_<domain>]
```

Here `<tool>` is the name of the tool used to create the service; $lan1$ and $lan2$ are languages which $\in \{it, en, el, es, fr, de\}$ and `<domain>` is the domain where CACs are applied, if available. We can summarize the naming convention as reported in table 2.1 below:

	Recommended	Optional
Type	Y	N
CAC Type	N	Y
Tool	Y	N
First Language (lan1)	Y	N
Second Language (lan2)	Y	N
domain	N	Y

Table 2.1: Naming convention for parallel CACs

Finally, the proposed naming convention, table 2.2:

`CAC[_parallelCAC]_<tool>_<lan1>_<lan2>[_domain]`

Table 2.2: Naming convention rule for parallel CACs

2.1.2 Bilingual Comparable Corpus Acquisition Components

We propose two distinct naming conventions for such services:

Using *comparable* as a tag In this scenario the "ontology" is as follows:

```
Main Class: Web Service
    Type: CAC
        CAC Type: comparable CAC
    Name: CAC_<tool>_<lan1>_<lan2>[_<domain>]
```

Using *comparable* in the name In this case the *comparable* property is not inherited directly from the classification, even if the tag can be left:

```
Main Class: Web Service
    Type: CAC
        (CAC Type: comparable CAC)
    Name: CAC_comparableCAC_<tool>_<lan1>_<lan2>[_<domain>]
```

Here `<tool>` is the name of the tool used to create the service; $lan1$ and $lan2$ are languages which $\in \{it, en, el, es, fr\}$ and `<domain>` is the domain where CACs are applied, if available. We can summarize the naming convention as reported in table 2.3 below:

	Recommended	Optional
Type	Y	N
CAC Type	N	Y
Tool	Y	N
First Language (lan1)	Y	N
Second Language (lan2)	Y	N
domain	N	Y

Table 2.3: Naming convention for comparable CACs

Finally, the proposed naming convention, table 2.4:

`CAC[_comparableCAC]_<tool>_<lan1>_<lan2>[_domain]`

Table 2.4: Naming convention rule for comparable CACs

Tables 2.2 and 2.4 can be summarized in the next one (table 2.5):

`CAC[_<type_of_cac>+CAC]_<tool>_<lan1>_<lan2>[_domain]`

Table 2.5: Naming convention rule for CACs

2.2 Monolingual Corpus Acquisition Components

The naming convention is simply:

`CAC_<tool>_<lan2>[_domain]`

Table 2.6: Naming convention rule for (monolingual) CACs

2.3 Naming convention for Clean-Up and Normalization Components

Clean-Up and Normalization Component (CNC) is a tool used to remove irrelevant parts from downloaded web pages. According to [1], CNCs can be classified following the tasks they address. These tasks are limited to the following set of value:
tasks = { *text normalization*, *language identification*, *web-page cleaning*, *duplicate detection* }
We propose two distinct naming conventions for such services:

Using the *task* as a tag In this scenario the "ontology" is as follows:

```
Main Class: Web Service
    Type: CNC
        CNC Task: task
    Name: CNC_<tool>[[_<lan1>]_[<lan2>....]][_<domain>]
```

Using the *task* in the name In this case the *task* property is not inherited directly from the classification, even if the tag can be left:

```
Main Class: Web Service
    Type: CNC
        (CNC Task: task)
    Name: CNC_<tool>_<task>[[_<lan1>]_[<lan2>....]][_<domain>]
```

Here `<tool>` is the name of the tool used to create the service; $lan1$ and $lan2\ldots$ are optional languages which $\in \{it, en, el, es, fr, de\}$ and `<domain>` is the domain where CNCs are applied, if available. We can summarize the naming convention as reported in table 2.7 below:

	Recommended	Optional
Type	Y	N
CNC Task	N	Y
Tool	Y	N
First Language (lan1)	N	Y
Second Language (lan2)	N	Y
other language(s) (lan$x$)	N	Y
domain	N	Y

Table 2.7: Naming convention for CNCs

Table 2.7 can be re-written as follows, (table 2.8):

`CNC[<tool>_<task>[[_<lan1>][_<lan2>..]][_domain]`

Table 2.8: Naming convention rule for CNCs

2.4 Naming convention for Text Processing Components

Text Processing Components follow Corpus Acquisition Components and Clean-Up and Normalization Components and deal with the processing of the automatically acquired and normalized corpora. These tools are adapted Natural Language Processing (NLP) tools.
Here the *language* parameter is essential, as essential is the set of NLP task they provide.
The naming convention for these services is challenging, since inserting all the tasks they provide in the name of the service could result annoying for the users. On the counterpart, inserting no task in the name could make complex for the users the retrieval of the services they are looking for.
In cases such this one, the ontological structure is even more needed, since with tags we can avoid inserting tasks into names.
We propose two distinct naming conventions for such services:

Using the *task(s)* as a tag In this scenario the "ontology" is as follows:

```
Main Class: Web Service
    Type: TPC
        TPC Task: task1
        TPC Task: task2
        TPC Task: task3
        ...............
    Name: TPC_<tool>[[_<lan1>]_[<lan2>....]][_<domain>]
```

Using the *task* in the name In this case the *task* property is not inherited directly from the classification, even if the tag(s) can be left:

```
Main Class: Web Service
    Type: TPC
        (TPC Task: task1
         TPC Task: task2
         TPC Task: task3)
    Name: TPC_<tool>_<list_of_task>[[_<lan1>]_[<lan2>....]][_<domain>]
```

Here `<tool>` is the name of the tool used to create the service; $lan1$ and $lan2\ldots$ are optional languages which $\in \{it, en, el, es, fr, de\}$ and `<domain>` is the domain where TPCs are applied, if available. We can summarize the naming convention as reported in table 2.9 below:

2.4.1 TPC caveat

It is clear that this strategy in naming services is not winning. We need something shorter. One idea could be define a list of tasks and group them into groups (*tpc_group*).
TPCs $\in$ *tpc_group** iif $\forall$ task they provide, this task $\in$ *tpc_group**. Table 2.9 can be rewritten as table 2.10

	Recommended	Optional
Type	Y	N
Tool	Y	N
TPC Task1	N	Y
TPC Task2	N	Y
...	N	Y
TPC Task$n$	N	Y
First Language (lan1)	N	Y
Second Language (lan2)	N	Y
other language(s) (lan$x$)	N	Y
domain	N	Y

Table 2.9: Naming convention for TPCs

`TPC_<tool>[_<tpc_group>][[_<lan1>][_<lan2>..]][_domain]`

Table 2.10: Naming convention rule for TPCs where tasks are grouped

2.5 Naming convention for Format Converter

Format Converters are a special type of Text Processing Components where the performed task is the conversion to and from many formats.
The first classification is between multi and mono converters. Services which accept one single format in input and provide one single format as output are mono converters, while services which accept different formats in input and provide different outputs, including one to many and many to one conversions, are multi converters.

2.5.1 FCs caveat

It is important to note that these converters only change the structure of the format, leaving untouched the annotation levels; so that we do not have to add properties (tag(s)) for describing annotation levels for both input and output but only the ones for input are needed.

2.5.2 Mono Converters

We propose two distinct naming conventions for such services:

Using both *input* and *output* formats as tags In this scenario the "ontology" is as follows:

```
Main Class: Web Service
    Type: FC
        FC input format: format_in
        FC output format: format_out
    Name: FC_<tool>[_<domain>]
```

Using both *input* and *output* formats in the name In this case the *input/output* properties are not inherited directly from the classification, even if the tag(s) can be left:

```
Main Class: Web Service
    Type: monoFC
        (FC input format: format_in
        FC output format: format_out)
    Name: FC_<format_in>_2_<format_out>[_<domain>]
```

Here `<tool>` is the name of the tool used to create the service; $format\_in$ and $format\_out$ are formats managed and `<domain>` is the domain where FCs are applied, if available. We can summarize the naming convention as reported in table 2.11 below:

	Recommended	Optional
Type	Y	N
Tool	Y	N
FC $format\_in$	N	Y
FC $format\_out$	N	Y
FC Task (conversion)	N	Y
domain	N	Y

Table 2.11: Naming convention for FCs

2.5.3 Multi Converters

As exposed in section 2.4.1, managing multi formats is similar to manage multi tasks when the naming convention is addressed. Even here, the strategy of inserting all possible combination in naming services is not winning. We need something shorter. One idea could be define a list of formats and group them into groups $in\_formats\_group$ and $out\_formats\_group$. and $formats\_group$ which is the group which maps input format(s) over output(s).
FCs $\in format\_group^*$ iif $\forall$ conversion they provide, this conversion $\in formats\_group^*$.
Table 2.11 can be rewritten as table 2.12

`[multi]FC_<tool>[_<formats_group>][_domain]`

Table 2.12: Naming convention rule for FCs where conversions are grouped

Chapter 3

Acronyms

CAC Corpus Acquisition Component

CNC Clean-Up and Normalization Component 8

FC Format Converter

NLP Natural Language Processing 9

PANACEA Platform for Automatic, Normalized Annotation and Cost-Effective Acquisition of Language Resources for Human Language Technologies 7th Framework Program; Information and Communication Technologies Grant agreement for: Small or medium scale focused research project(STREP). Grant agreement: 248064 1

TPC Text Processing Component 1

Bibliography

[1] Prokopis Prokopidis, Vassilis Papavassiliou, Pavel Cecina, Laura Rimmel, Thierry Poibeau, Roberto Bartolini, Tommaso Caselli, Vera Aleksic, Gregor Thurmair Marc Poch, Núria Bel, and Olivier Hamon. Technologies and tools for corpus creation normalization and annotation (deliverable d4.1). Technical report, PANACEA, 2010.