A CMD Core Model for CLARIN Web Services

Menzo Windhouwer, Daan Broeder, Dieter van Uytvanck
Max Planck Institute for Psycholinguistics
Wundtlaan 1, 6525 XD Nijmegen, The Netherlands
Menzo.Windhouwer@mpi.nl, Daan.Broeder@mpi.nl, Dieter.vanUytvanck@mpi.nl

Abstract

In the CLARIN infrastructure various national projects have started initiatives to allow users of the infrastructure to create chains or workflows of web services. The Component Metadata (CMD) core model for web services described in this paper tries to align the metadata descriptions of these various initiatives. This should allow chaining/workflow engines to find matching and invoke services. The paper describes the landscape of web services architectures and the state of the national initiatives. Based on this a CMD core model for CLARIN is proposed, which, within some limits, can be adapted to the specific needs of an initiative by the standard facilities of CMD. The paper closes with the current state and usage of the model and a look into the future.

1. Introduction

In the grand CLARIN[1] (Váradi et al., 2008) vision "the user will have access to repositories of data with standardized descriptions, processing tools ready to operate on standardized data, and all of this will be available on the internet using a service oriented architecture" (CLARIN community, 2008). These processing tools can be dedicated desktop tools but also services hosted by various CLARIN (computing) centers and accessible over the web.

In the preparatory phase CLARIN national projects contributed existing or new initiatives in the domain of web services. In Spain UPF provides various families of services, e.g., statistical and CQP web services (see §4.1). A major result of the German D-SPIN project has been the first version of WebLicht, a chaining engine for linguistic web services (see §4.2). The Dutch and Flemish TTNWW project aims at supporting web service workflows for both textual and multimedia resources (see §4.3).

This means that there is not a single CLARIN web service chaining/workflow engine. However, in the CLARIN infrastructure, which aims at unification instead of fragmentation, it should at least be technically possible for all engines to find matching and invoke all known services within the infrastructure.

One pillar of CLARIN is that all metadata on resources, including web services, are to be specified using the Component MetaData Infrastructure (CMDI). This framework is very flexible and should allow mixing common and engine specific metadata for web services. This paper describes the design and use of an extensible CMD model for common web service metadata.

Sections 2 and 3 give an introduction of the major web service architectures and their impact on metadata descriptions, and the CMDI framework. The next section described how web services are described in various national CLARIN projects. On this basis the CMD core model for CLARIN Web Services and its possible usage will be fleshed out in section 5 and 6. The last section will deal with the current state and usage of the model.

[1] Acronyms can be looked up in §9

2. Web service architectures

In the history of the Internet several ways have been proposed to implement Service Oriented Architectures (SOAs) based on the basic protocol for the World Wide Web HTTP. According to (Richardson et al., 2007) three basic web service architectures can be identified. This classification is based on the differences in how the architectures handle two basic information items:

1. Method information: how does the client convey its intentions to the server, i.e., why should the server do *this* instead of doing *that*?
2. Scoping information: how does the client tell the server which part of the data set to operate on, i.e., why should the server operate on *this* data instead of *that* data?

In the CLARIN landscape all three architectures can be encountered.

2.1 RESTful resource-oriented architectures

A web service architecture is considered RESTful if the method information goes into the verb that determines the nature of the HTTP request, e.g., PUT, GET, POST or DELETE, and resource oriented if the scoping information goes into the URI. Resource orientation means also that this URI does not actually refer to a service but to a resource, where resolving the URI results in a representation of that resource. These architectures are directly build upon the technical foundations that made the World Wide Web successful (Fielding, 2000).

A well-known example of a RESTful resource-oriented architecture is Amazon's Simple Storage Service (Amazon Web Services LLC, 2006). Also services that are exposed by the Atom Publishing protocol (Gregorio et al., 2007) are examples.

2.2 RPC-style architectures

In RPC (Remote Procedure Call) architectures envelopes full of data are sent and received from the services. Both the method and scoping information are kept inside the envelope. The XML-RPC protocol (Winer, 2003) is a prime example of such architecture. It ignores most features of HTTP, i.e., only one URI (the service endpoint) is used and one HTTP method (POST). Contrary to RESTful architectures this disables a lot of the basic infrastructure, e.g., caching of GET requests, which made the World Wide Web scalable and successful.
The same can be said about most usages of SOAP (Simple Object Access Protocol) (W3C XML Protocol Working Group, 2007) on top of HTTP. In this case SOAP is the envelope format in which the method and scoping information is provided.

2.3 REST-RPC hybrid architectures

This group of service architectures have REST-like elements, e.g., they put the scoping information in the URI, but they do that as well for the method information, e.g., have a single endpoint with a query parameter that specifies the service to call. An example of a REST-RPC hybrid is the Flickr REST API (Flickr, 2012).

2.4 Interface Description Language

An Interface Description Language (IDL) is commonly used by RPC architectures to specify the services which are available at an endpoint. In the case of SOAP the IDL is the Web Service Definition Language (WSDL) (Christensen et al., 2001). The WSDL provides information on the input and output of the services.
For RESTful resource-oriented architectures there has been an on-going debate if an IDL is needed. Patterns are proposed which enable the transition of one service, or resource representation, to another, e.g., Hypermedia as the Engine of Application State (HATEOAS) (Fielding, 2000; Fielding, 2008) where a client basically follows the links between resources just like a browser a does with the links embedded in a HTML page. However, in current practice this style of web services is too free form to automatically determine how to call a service. So relying only on a text document to define the API is naïve and does not scale. For example, parameters can be passed on in many ways, e.g., embedded in the URI path, as query parameters or as part of a multipart POST request. The Web Application Description Language (WADL) (Hadley, 2009) has been submitted to W3C as a possible IDL to describe RESTful web services. But WADL did not make it into a W3C recommendation yet and from time to time competing IDLs are proposed, e.g., ReLL (Alarcón et al., 2010) and the RDF-based RESTdesc (Verborgh, 2012). Also version 2 of WSDL allows describing this RESTful web services. IDLs suitable for RESTful web services can in general also be used for REST-RPC architectures.

3. The Component Metadata Infrastructure

This section introduces CMDI, the metadata infrastructure that is to be used for all metadata describing resources in the CLARIN domain, including web services. The role of and link between descriptions of a web service in an IDL and in CMDI will be described later on in this paper.
In the CLARIN infrastructure CMDI (Broeder et al., 2011) has been developed to be able to better tailor a metadata schema to the needs of a (type of) resource. Previous attempts resulted in either too few metadata elements, e.g., Dublin Core, or in too many, e.g., IMDI. Both cases can result in poor metadata quality as users misuse elements when there are too few or give up when there are too many.
CMDI is based on a registry of reusable components (CLARIN community, 2012). Users can combine suitable components into profiles. These profiles can be transformed into an XML Schema so actual instances of the profiles can be validated. When needed users can create new component and profiles, but they can also copy existing components and adapt them till they suit their specific needs. However, CLARIN will benefit if proliferation of components is kept to the minimum.
Components, elements and values in CMD can be linked to concepts or data categories defined in an external registry. In CLARIN the preferred registry is the ISOcat (Max Planck Institute for Psycholinguistics, 2012) Data Category Registry (DCR), which is an implementation of (ISO 12620, 2009) and as the ISO TC 37 DCR dedicated to the linguistic domain. These links allow establishing semantic interoperability between components, elements or values in different CMD profiles. And even allows for differences in the use of terminology, cases or orthography.

4. CLARIN web service chaining and workflow engines and registries

As stated before various national CLARIN projects have started initiatives in the area of Web Services. In this section these initiatives are sketched with a focus on their support for metadata description of the services.

4.1 Spain

In Spain IULA at UPF provides access to various families of web services (see §4.2.6 in (Funk et al., 2010) and (CLARIN-CAT and -ES community, 2012)):

- Format conversion services: provide different format conversion tools such as PDF, MS Word and HTML to plain text, character conversion tools, etc.;
- Statistical services: provide statistical information on an uploaded corpus, e.g., the "Herdan" index of lexical richness or all the n-grams with its number of occurrences;
- Annotation services: including morphosyntactic, syntactic and dependency annotators;
- Corpus management services: deploys a CWB as a web service and allows indexing and further exploitation of an annotated corpus.

Access to the services is provided via SOAP, so the technical, also known as the syntactic, description is given in WSDL. Additional metadata and semantics are provided in a separate semantic description, inspired by the SoapLab2 semantic annotations and the myGrid ontology (Villegas et al., 2010). A CMDI profile[2] has been created for these semantic descriptions. The following fragment[3] is taken from the XSLT processor service description:

```
<serviceDescription>
 <serviceName>xsltprocService</>
 …
 <locationURL>…/soaplab2-axis/</>
 <interfaceWSDL>…xsltproc?wsdl</>
 …
 <operations>
  <serviceOperation>
  <operationName>runAndWaitFor</>
  <portName>xsltproc</>
  …
  <operationInputs>
   <MyGridParameter>
    <parameterName>stylesheet</>
    …
    <isConfigurationParameter>false</>
    <semanticType>stylesheet</>
    …
    <XMLSchemaURI>…xsltproc?xsd=1</>
    …
    <formats>
     <formatIdentifier>text/xml</>
     <formatIdentifier>UTF-8</>
   </></>
   …
</></></>
```

Figure 1: Fragment of an UPF service description

This Spanish initiative is continued in the PANACEA project, a STREP project under EU-FP7 (Bel, 2010). The ELDA PANACEA web service registry (ELDA, 2012) provides the latest usage statistics.

4.2 Germany

The German D-SPIN project created the WebLicht chaining engine for web services (see §1 in (Ogrodniczuk et al., 2011)). Services in WebLicht are REST-based and in current practice a single TCF document is pushed through a pipeline of services, where each service adds a new layer to the TCF document. Around a hundred services, e.g., tokenizers and part-of-speech taggers, for various languages are accessible via WebLicht.

For the syntactic description of services there is no usage of an IDL as the invocation recipe for a service accessible by WebLicht is well known by the chaining engine, i.e., POST the TCF document. The metadata description of services focuses mainly on specifying the required input layers and produced output layers. This description supports profile matching to build a chain. The following fragment illustrates this:

```
<service>
 <name>TreeTagger 117 152</>
 <url>…/tree-tagger3.perl</>
 …
 <replacesinput>false</replacesinput>
 <input type="text/tcf+xml">
  <feature name="lang">
   <value name="de"/>
   <value name="it"/>
   <value name="en"/>
  </>
  <feature name="version">
   <value name="0.3"/>
  </>
  <feature name="layer.tokens"/>
 </>
 <output type="text/tcf+xml">
  <feature name="layer.postags"/>
  <feature name="layer.lemmas"/>
  <feature name="layer.postags.tagset">
   <value refValue="it" refFeature="lang"
     name="stein"/>
   <value refValue="en" refFeature="lang"
     name="penntb"/>
   <value refValue="de" refFeature="lang"
     name="stts"/>
</></></>
```

Figure 2: Fragment of a WebLicht service description

WebLicht (SfS Tübingen, 2012) can be used by the CLARIN community and development continues in the successor to D-SPIN the CLARIN-D project (CLARIN-D, 2012).

4.3 The Netherlands and Flanders

CLARIN-NL and CLARIN Flanders cooperate in the TTNWW project, which aims at providing access to national services as for example developed in the STEVIN project. Two modalities are being addressed: text and speech. In TTNWW no assumption is made with regard to the web service architecture, i.e., it should be possible to integrate services based on RESTful resource-oriented, RPC-style or REST-RPC hybrid architectures.

Metadata descriptions are based on the data model described in (Kemps-Snijders, 2010). The following example shows a fragment, including a reference to the WSDL via a CMD resource proxy.

```
<CMD>
 <Header>…</>
 <Resources>
  <ResourceProxyList>
   <ResourceProxy>
    <ResourceType>WSDL service</>
    <ResourceRef>…/LangId.asmx</>
   </ResourceProxy>
  </ResourceProxyList>
 </Resources>
 <Components>
  <Service>
```

[2] See http://catalog.clarin.eu/ds/ComponentRegistry?item=clarin.eu:cr1:p_1295178776924

[3] Due to limited space the XML has been trimmed by abbreviating all end tags to </> and to leave out some content (indicated by ellipses '…').

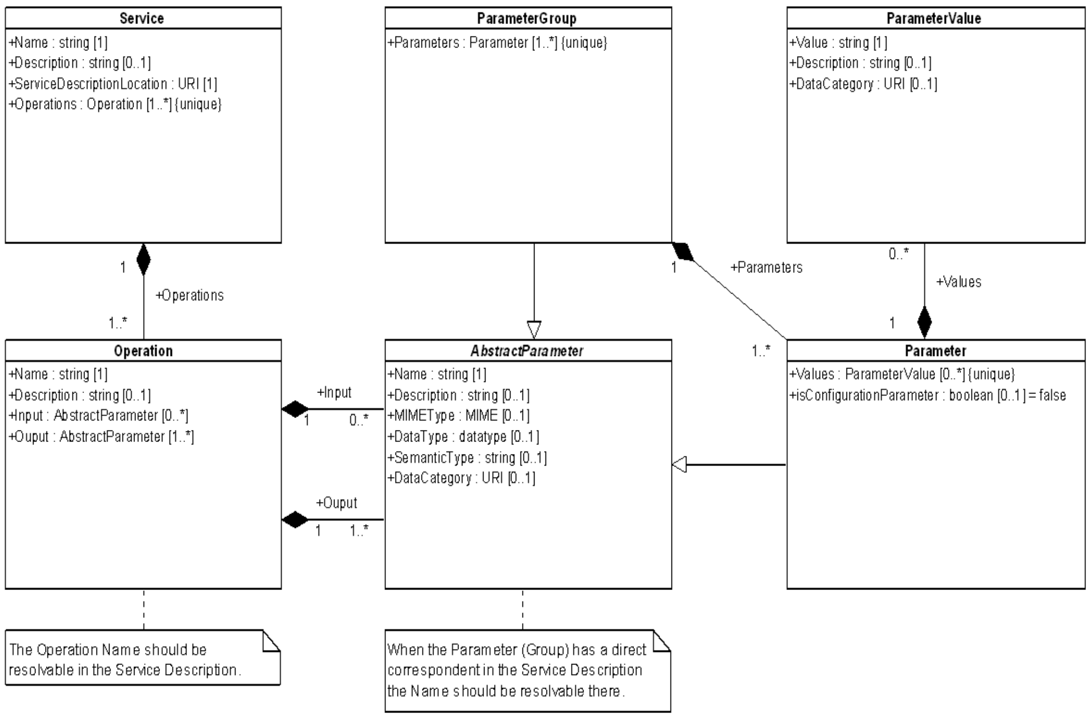
Service
+Name : string [1]
+Description : string [0..1]
+ServiceDescriptionLocation : URI [1]
+Operations : Operation [1..*] {unique}
ParameterGroup
+Parameters : Parameter [1..*] {unique}
ParameterValue
+Value : string [1]
+Description : string [0..1]
+DataCategory : URI [0..1]
1
+Operations
1..*
1
+Parameters
0..*
+Values
1
1..*
Operation
+Name : string [1]
+Description : string [0..1]
+Input : AbstractParameter [0..*]
+Ouput : AbstractParameter [1..*]
AbstractParameter
+Name : string [1]
+Description : string [0..1]
+MIMEType : MIME [0..1]
+DataType : datatype [0..1]
+SemanticType : string [0..1]
+DataCategory : URI [0..1]
Parameter
+Values : ParameterValue [0..*] {unique}
+isConfigurationParameter : boolean [0..1] = false
+Input
1
0..*
+Ouput
1
1..*
The Operation Name should be resolvable in the Service Description.
When the Parameter (Group) has a direct correspondent in the Service Description the Name should be resolvable there.

Figure 3: UML model for CLARIN web services

```
<Type>SOAP</>
…
<Name>LangIdWebService</>
<URL>hdl:service</>
<Operation>
 <Name>IdentifyLanguage</>
 <Action>…/IdentifyLanguage</>
 <Input>
  <Parameter>
   <Name>IdentifyLanguage.text</>
   …
   <TechnicalMetadata>
    <MimeType>text/plain</>
    <CharacterEncoding>UTF-8</>
   </TechnicalMetadata>
  </>
  <Parameter>
   <Name>….modern_languages</>
   <DataCategory>…</>
  </Parameter>
  <Parameter>
   <Name>….rare_languages</>
   <DataCategory>…</>
  </Parameter>
 </>
 <Output>
  <Parameter>
   <Name>….Language</>
   <TechnicalMetadata
     parameterRef="IdentifyLanguage.text">
    <MimeType>text/plain</>
    <CharacterEncoding>UTF-8</>
    <PLORK>WAF</PLORK>
    <ContentEncoding>
        <URL>hdl:testSchema</URL>
        <ResourceFormat>PlainTextResource</>
     </></></>
     <Parameter>
      <Name>….Confidence</>
      <DataCategory>…</>
</></></></></></></>
```

Figure 4: Fragment of a TTNWW service description

The TTNWW project is on-going and has, at time of writing, not been publically released.

5. A CMD core model for web services

As shown in the previous section the national CLARIN projects support diverse web service architectures including various mechanisms for describing web services on the semantic and syntactic levels. The CMD core model described in this section is an attempt to distil a common core out of these existing descriptions.

5.1 An initial UML model

Discussion on the core model were based on an UML model and after several iterations resulted in the class diagram shown in Figure 3.
In a hierarchical perspective on the diagram, which matches the CMD approach, the *Service* class is taken as the root. A major design decision is that each Service should refer to a service description (see the *Service-DescriptionLocation* attribute), e.g., a reference to a WSDL or WADL instance. Here the core model follows the Spanish approach. The CMD description mainly focuses on semantics and there is an additional syntactic description that provides more technical details. These technical details are needed as the CMD description might be powerful enough to do profile matching, i.e., determine if the output of one service can be used as

input to another service, but it does not provide enough information to really invoke these services. This is the penalty for the freedom that REST-style web services allow developers. Take for example an WebLicht service: the WebLicht chaining engine knows its own recipe, i.e., it should POST the TCF document to the URI of a service, but another chaining or workflow engine would not know that. In the syntactical service description for REST-style web services this recipe is made explicit, so any engine can know how to invoke a service.

The core model actually does not state which IDL should be used. For the time being WSDL (2) and WADL seem to be the most appropriate candidates able to support all the web service architectures described in Section 2.

The *Service* class does not contain any attribute to specify the URI of the service (endpoint) as this is considered technical information, which is provided in the syntactical service description.

The syntactical service description is able to describe a collection of services. In an RPC architecture these are the operations provided by a single endpoint, and also a one WADL document can describe a collection of REST-style web services. A *Service* instance can thus refer to one or more operations.

Each operation is an instance of the *Operation* class which contains the in- and output specifications. As it should be clear how to invoke this operation the name of the operation in the semantic description should be the same as the one used for it in the syntactical description.

Input and output are sets of parameters. As illustrated in the case of the TCF document used by WebLicht, profile matching might actually need to look into the contents of the resource send around in the chain or workflow, i.e., it should be possible to state that a lemmatizer needs an input TCF document containing a token layer. Notice that the syntactical description does not need to specify about layers in the file, it only needs to specify how to ship the TCF document to the service. The UML model deals with this by allowing an in- or output parameter to be either a *ParameterGroup* or a *Parameter*, which are both subclasses of the abstract *AbstractParameter* class. In WebLicht the in- or output TCF document would correspond to a *ParameterGroup* and a layer to a *Parameter* in this group. Both *Parameter* and *ParamaterGroup* share a number of optional attributes that allow providing various levels of profile matching from technical to service specific semantics:

1. *MIMEType*: the technical MIME type of a resource will also reveal its media type, e.g., text/plain;
2. *DataType*: a value domain, in general taken from the well-known XML Schema data types (Biron et al., 2004), e.g., ID;
3. *DataCategory*: a reference to a data category, in general taken from ISOcat, e.g., http://www.isocat.org/datcat/DC-2535 (*/project id/*);
4. *SemanticType*: free form string to indicate service specific types, e.g., ‘clam.project.adelheid’.

A profile matching algorithm can use these various levels to prune away semantic mismatches from a list of syntactic matches, e.g., matching an Adelheid (Halteren, 2009) project id with a service that accepts arbitrary plain text would be useless.

The names of parameters or parameter groups, depending on which corresponds to an actual technical parameter, should correspond to names used for the same parameter in the syntactical description.

The lowest level of the hierarchy contains the *ParameterValue* class which is used to capture descriptive information of value enumerations for parameters.

This UML model covers major parts of the various semantic descriptions mentioned in Section 4. The CMD infrastructure will provide the means to add any repository specific information to this common part.

5.2 CMD components for the core model

To be useful in the CLARIN context the UML model has to be instantiated as a set of CMD components. However, CMD does not support any inheritance, i.e., one cannot create an *AbstractParameter* component and describe how *Parameter* and *ParameterGroup* components are related to it, so specific mapping rules between the two models, aimed at maintaining as much of the semantics as possible, have to be followed:

1. Each non-abstract class becomes a component, e.g., *Service* and *Operation* but not *AbstractParameter*;
2. Each attribute, both inherited and local, becomes an element, e.g., *Name* or *Description*, but
3. attributes, both inherited and local, referring to non-abstract classes become components with a child component representing the referred non-abstract class, e.g., *Operations*;
4. Attributes, both inherited and local, referring to abstract classes should become components with optional child components representing all the non-abstract classes lower in the inheritance hierarchy, e.g., *Input* and *Output*;
5. Cardinality constraints are copied where possible, e.g., in the case of the attributes referring to abstract classes these will be lost, e.g., CMD cannot express that an *Output* instance should refer to at least one *Parameter* or *ParameterGroup* instance.

Reusability considerations determine which components related to classes exist on their own in the registry, while others only exist within another component. *ParameterValue*, for example, is only used inside Parameter and is considered unlikely to be reused somewhere else.

The CMD components resulting from this mapping UML model have been created in the Component Registry and combined into a profile[4]. Only in one case the rules described in this section were not followed: the *ServiceDescriptionLocation* attribute was not mapped to a CMD element but to a CMD component. The idea behind this has been to enable the use of a CMD resource proxy for the reference to the syntactic description. This promotes the approach taken in TTNWW as shown in Figure 4.

[4] See http://catalog.clarin.eu/ds/ComponentRegistry?item=clarin.eu:cr1:p_1311927752335&space=public

```
<Resources>
 <ResourceProxyList>
  <ResourceProxy id="h1">
   <ResourceType
     mimetype="application/vnd.sun.wadl+xml">
    Resource
   </>
   <ResourceRef> …/tds-services.wadl</>
 </></>
 …
</Resources>
<Components>
 <ToolService>
  …
  <Service CoreVersion="1.0">
   <Name> Typological Database System</>
   <ServiceDescriptionLocation ref="h1" />
    <Operations>
     …
</></></></></></>
```

Figure 5 *ServiceDescriptionLocation* uses a resource proxy

6. Usage of the core model

Now that the CMD core model for CLARIN Web Services is available the question arises: how can a web service repository adapt and use it? The core model profile should not be instantiated directly as it functions as a template for profiles specific to the various national initiatives. For CLARIN-NL an extension has been created where a *TechnicalMetaData* component (see also the *TechnicalMetaData* fragments in Figure 4) has been added to the *ParameterGroup* and *Parameter* components. This component contains elements to specify, for example, the character encoding, a reference to an XML schema or the location of an output parameter in a resource. In the following fragment the bold parts of the instance correspond to the core model.

```
<Operation>
 <Name>query</>
 <Description>Query the data section of an IDDF
document.</>
 <Input>
  <Parameter>
   <Name>file</>
   <DataType>string</>
   <SemanticType>iddf.file</>
   <TechnicalMetadata>
    <CharacterEncoding>UTF-8</>
  </></>
  <Parameter>
   <Name>query</>
   <MIMEType>text/xml</>
   <TechnicalMetadata>
    <CharacterEncoding>UTF-8</>
    <ContentEncoding>
     <URL>…/query.rng</>
     <ResourceFormat>IDDF Query XML</>
  </></></>
  …
 </>
 <Output>
  <ParameterGroup>
   <Name>query-result</>
   <MIMEType>text/xml</>
   <Parameters>
    <Parameter>
     <Name>notion</>
     <DataType>ID</>
     <SemanticType>iddf.notion</>
     <TechnicalMetadata>
      <CharacterEncoding>UTF-8</>
      <ContentEncoding>
       <RelativeLocation>//@iddf:notion</>
    </></></>
    …
</></></></>
```

Figure 6 Fragment of a CLARIN-NL service description

The CLARIN-NL tool and services description profile was created by copying the components from the core model and adding the additional components and elements. This need to copy and edit existing components opens up the possibility to also delete components/elements which were mandatory in the core model. Additional components or elements can be freely added but changes to existing components or elements need to follow some rules, so instances are valid both in the core model and the extension:

1. Cardinalities in the extension should be within the boundaries set by the core model, e.g., mandatory elements cannot become optional but optional elements like *Description* can become mandatory;
2. Closed value domains cannot be extended, but open value domains like for *SemanticType* can be turned into closed ones;
3. Data category references in the core model should not be touched as this could imply different semantics.

By following these rules it should be possible to strip of all additional components and elements from an instance and still be left with a valid instance of the core model. Taking the example fragment in Figure 6 only the bold styled elements would be left. This validation process has been implemented and is available to developers at http://www.isocat.org/clarin/ws/cmd-core/. The target audience of the core model consists of developers of web service registries. Web service developers, which want to make their services available to one of the CLARIN chaining/workflow engines, should just use the core model compliant CMD profile of a CLARIN registry.
The fragment in Figure 6 showed part of the semantic description of the TDS IDDF query web service (Dimitriadis, 2009). This fragment has its counterpart in the syntactic, or technical, WSDL description.

```
<method name="POST" id="query">
 <request>
  <representation
    mediaType="multipart/form-data">
   <param name="service" type="xs:string" …
     fixed="query" style="query"
     required="true"/>
   <param name="file" type="xs:string"
     style="query" required="true"/>
   <param name="query" style="query"
     required="true"/>
  …
 </></>
 <response>
  <representation mediaType="text/xml">
   <param name="notion" path="//@iddf:notion"
     repeating="true" style="plain"/>
   …
</></></>
```

Figure 7 Fragment of a CLARIN-NL WSDL

The TDS IDDF web services use a RPC-REST hybrid architecture, where the method information is passed on in the URI as a query parameter. In Figure 7 this is the first input parameter named *service* with the fixed value 'query', which is the name of the service to be executed by the RPC endpoint. This parameter does not appear in Figure 6, which shows that this low-level implementation detail is hidden from the semantic description of the web service. Also notice that the names for the operation, i.e., 'query', and the in- and output parameters, e.g., 'file' and 'notion', are the same in the two descriptions, so one can connect the semantic and syntactic information levels.

7. Future work and conclusions

This paper described the development of a CMD core model for CLARIN web service descriptions. At the time of writing only the CLARIN-NL tool and service description profile is compliant with the core model and publically available in the public workspace of the Component Registry. A few Dutch web services have been described, but this profile is not yet in use by the TTNWW project. The German WebLicht project is adopting CMDI and the core model in version 2.0.

It will only be a first step if the various registries use a CMD profile that is compliant with the core model. The next, and most important step to measure uptake, is when the various chaining/workflow engines are able to process both the semantic and syntactic web service descriptions and thus are able to invoke generic services not specifically tailored to their system.

As the construction of the CLARIN infrastructure proceeds more complex use cases are being addressed, also in the area of web services. One of the trends is to incorporate asynchronous web services. In general these are not single services that do an (advanced) operation and return their result 'immediately', but instead various services need to be called in a specific sequence. A common pattern is to call a service to start the operation, then use another service to poll at regular intervals if the operation has finished, and if so to fetch the result by yet another service. In the Netherlands CLAM (Gompel, 2011) is a popular REST-based framework that is based on this pattern. This is in fact a mini workflow and projects like TTNWW are implementing them as such and compose larger workflows out of multiple mini workflows. Users of the infrastructure then call these pre-composed workflows instead of single web services. It remains to be seen if this kind of workflows can be handled in the same way as web services and thus can use the core model, or if another model or adaptions to this model are needed.

Alignment with or reuse of the core model by other (metadata) infrastructure initiatives could enable wider integration. The META-SHARE meta model is also based on components and ISOcat and contains a section on Tools and Services (see §8 in (Desipri et al., 2012)) and would thus be a prime candidate.

8. Acknowledgements

The CMD core model for CLARIN Web Services has profited from feedback from the members of the ISOcat CLARIN Web Services group, especially Marc Kemps-Snijders, Marta Villegas and Thomas Zastrow.

9. Acronyms

API	Application Programming Interface
CLARIN	Common Language Resources and Technology Infrastructure
CLAM	Computational Linguistics Application Mediator
CMDI	Component Metadata Infrastructure
CWB	Corpus Workbench
DCR	Data Category Registry
D-SPIN	Deutsche Sprachressourcen-Infrastruktur
HATEOAS	Hypertext as the Engine of Application State
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
ID	Identifier
IDDF	Integrated Data and Documentation Format
IDL	Interface Description Language
IMDI	ISLE MetaData Initiative
ISO	International Organization for Standardization
IULA	Institut Universitari de Lingüística Aplicada
META	Multilingual Europe Technology Alliance
MIME	Multipurpose Internet Mail Extensions
MS	Microsoft
PANACEA	Platform for Automatic, Normalized Annotation and Cost-Effective Acquisition
PDF	Portable Document Format
RDF	Resource Description Format
REST	Representational State Transfer
RPC	Remote Procedure Call
SOA	Service Oriented Architecture
SOAP	Simple Object Access Protocol
STEVIN	Spraak- en Taaltechnologische Essentiële Voorzieningen In het Nederlands
TC	Technical Committee
TCF	Text Corpus Format
TDS	Typological Database System
TTNWW	TST Tools voor het Nederlands als Webservices in een Workflow
UML	Unified Modeling Language
UPF	Universitat Pompeu Fabra
URI	Uniform Resource Identifier
W3C	World Wide Web Consortium
WADL	Web Application Description Language
WSDL	Web Service Description Language
XML	Extensible Markup Language
XSLT	Extensible Stylesheet Language Transformations

10. References

Alarcón, R. and E. Wilde (2010). RESTler: Crawling RESTful Services. WWW 2010. Raleigh, North Carolina, USA, ACM.

Amazon Web Services LLC. (2006). Amazon Simple Storage Service API Reference. Retrieved 16 February 2012, from http://docs.amazonwebservices.com/AmazonS3/latest/API/APIRest.html.

Bel, N. (2010). Platform for Automatic, Normalized Annotation and Cost-Effective Acquisition of Language Resources for Human Language Technologies: PANACEA. XXVI Congreso de la Sociedad Española para el Procesamiento del Lenguaje Natural (SEPLN-2010). Valencia, Spain.

Biron, P. V., K. Permanente and A. Malhotra. (2004). XML Schema Part 2: Datatypes Second Edition. W3C recommendation Retrieved 16 February 2012, from http://www.w3.org/TR/xmlschema-2/.

Broeder, D., O. Schonefeld, T. Trippel, D. v. Uytvanck and A. Witt (2011). A pragmatic approach to XML interoperability – the Component Metadata Infrastructure (CMDI). Balisage: The Markup Conference 2011. Montréal, Canada.

Christensen, E., F. Curbera, G. Meredith and S. Weerawarana. (2001). Web Services Description Language. Retrieved 16 February 2012, from http://www.w3.org/TR/wsdl.

CLARIN-CAT and -ES community. (2012). Clarin-Cat-Lab and Clarin-Es-Lab Retrieved 27 Feburary 2012, from http://clarin-cat-lab.org/ and http://clarin-es-lab.org/.

CLARIN-D. (2012). CLARIN-D: a web and centres-based research infrastructure for the social sciences and humanities. Retrieved 16 February 2012, from http://www.clarin-d.de/.

CLARIN community. (2008). About CLARIN » Mission. Retrieved 16 February 2012, from http://www.clarin.eu/external/index.php?page=about-clarin&sub=0.

CLARIN community. (2012). Clarin Component Browser. Retrieved 16 February 2012, from http://catalog.clarin.eu/ds/ComponentRegistry/.

Desipri, E., M. Gavrilidou, P. Labropoulou, S. Piperidis, F. Frontini, M. Monachini, V. Arranz, V. Mapelli, G. Francopoulo and T. Declerck (2012). Documentation and User Manual of the META-SHARE Metadata Model P. Labropoulou and E. Desipri.

Dimitriadis, A. (2009). TDS Curator - A web-services architecture to curate the Typological Database System. CLARIN Call I prohect Retrieved 17 February 2012, from http://www.clarin.nl/node/70#TDS_Curator.

ELDA. (2012). The PANACEA registry. Retrieved 28 March 2012, from http://registry.elda.org/.

Fielding, R. (2000). Architectural Styles and the Design of Network-based Software Architectures. Irvine, University of California.

Fielding, R. T. (2008). REST APIs must be hypertext-driven. Retrieved 16 February 2012, from http://roy.gbiv.com/untangled/2008/rest-apis-must-be-hypertext-driven.

Flickr. (2012). Flickr Services. Retrieved 16 February 2012, from http://www.flickr.com/services/api/request.rest.html.

Funk, A., N. Bel, S. Bel, M. Büchler, D. Cristea, F. Fritzinger, E. Hinrichs, Marie Hinrichs, R. Ion, M. Kemps-Snijders, Y. Panchenko, H. Schmid, P. Wittenburg, U. Quasthoff and T. Zastrow (2010). Requirements Specification Web Services and Workflow Systems. CLARIN deliverable. D2-R6b.

Gompel, M. v. (2011). CLAM: Computational Linguistics Application Mediator. Retrieved 17 February 2012, from http://ilk.uvt.nl/clam/.

Gregorio, J. and B. d. hOra (2007). The Atom Publishing Protocol IETF - Network Working Group. RFC 5023.

Hadley, M. (2009). Web Application Description Language. W3C submission Retrieved 16 Feburary 2012, from http://www.w3.org/Submission/wadl/.

Halteren, H. v. (2009). Adelheid - A Distributed Lemmatizer for Historical Dutch. CLARIN Call 1 project Retrieved 17 February 2012, from http://www.clarin.nl/node/70#Adelheid.

ISO 12620 (2009). Terminology and other language and content resources - Specification of data categories and management of a Data Category Registry for language resources, International Organization for Standardization.

Kemps-Snijders, M. (2010). Web services and workflow creation. CLARIN deliverable. D2R-7b.

Max Planck Institute for Psycholinguistics. (2012). ISOcat - Data Category Registry. Retrieved 16 February 2012, from http://www.isocat.org/.

Ogrodniczuk, M. and A. Przepiórkowski (2011). Integration of Language Resources into Web service infrastructure. CLARIN deliverable. D5R-3b.

Richardson, L. and S. Ruby (2007). RESTful Web Services, O'Reilly.

SfS Tübingen. (2012). WebLicht Web-based Linguistic Chaining Tool. Retrieved 28 March 2012, from https://weblicht.sfs.uni-tuebingen.de/.

Váradi, T., S. Krauwer, P. Wittenburg, M. Wynne and K. Koskenniemi (2008). CLARIN: Common Language Resources and Technology Infrastructure. Sixth International Conference on Language Resources and Evaluation (LREC'08). N. Calzolari, K. Choukri, B. Maegaardet al. Marrakech, Morocco, European Language Resources Association (ELRA).

Verborgh, R. (2012). RESTdesc – Semantic descriptions for RESTful Web APIs. Retrieved 16 February 2012, from http://restdesc.org/.

Villegas, M., N. Bel, S. Bel and V. Rodríguez (2010). A Case Study on Interoperability for Language Resources and Applications. The Seventh International Conference on Language Resources and Evaluation (LREC'10). N. Calzolari, K. Choukri, B. Maegaardet al. Valletta, Malta, European Language Resources Association (ELRA): 3512-3519.

W3C XML Protocol Working Group. (2007). SOAP Specifications. W3C recommendation Retrieved 16 February 2012, from http://www.w3.org/TR/soap/.

Winer, D. (2003). XML-RPC Specification. Retrieved 16 February 2012, from http://xmlrpc.scripting.com/spec.