A prototype infrastructure for D-Spin-Services based on a flexible multilayer architecture

Volker Boehlke[1],

[1] NLP Group, Department of Computer Science,
University of Leipzig,
Johanisgasse 26,
04103 Leipzig, Germany

Abstract: In this paper we will point out why building a prototypical infrastructure for D-Spin/Clarin will help us to identify and solve most of the key problems, like specifying data formats, when trying to combine many different resources and tools in one infrastructure. We will explain how input and output specifications enable us to validate and execute process chains in this environment. We will also describe how such an prototype can be implemented and how it will help as a baseline for further research.

Keywords: D-Spin, Clarin, data interchange format, resources, tools, services

1 Introduction

One of the main goals of D-Spin[1] and Clarin[2] is to build an infrastructure that allows non technicians to combine different resources and services to self defined workflows. They have to be able to solve their specific problems without spending much thought about technical infrastructures, standards, algorithms etc. A user may just "choose" a resource and define which processes should work on it, which other resources have to be used etc. In order to give a small example, suppose want to do POS-tagging on text from a specific resource. While doing this, we don't care about the fact that several different steps, like language recognition and tokenization, are necessary, or that the involved tools internally are using different data representations. We just want to pick a resource in a graphical user interface and define what to do with it.

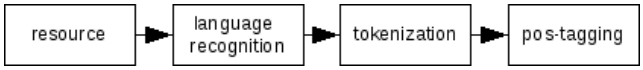

Figure 1: An example process chain.

1.1 Why we want build a prototype infrastructure

From a technical point of view, combining different tools and resources is a very difficult task. Not only because problems always begin when different tools from different sources have to work together, but also because it's hard to get a complete overview of use cases and problems that would allow to validate a possible solution. Even if such an overview is available, a validation will be very difficult to do on paper or by just thinking about it. We know that there are different technologies out there that might help us (ALPE[3], GATE[4], UIMA[5], ...). But maybe it's a good idea to draw and also implement some kind of simple baseline first. This baseline can be used for further tests in order to build up knowledge and to compare more complete but also complex solutions with it. This is why we want to implement some example process chains in D-Spin, using a rather simple infrastructure. It's only purpose is to not just describe, but to also establish some well known test cases for further experiments, that will allow us to validate other technologies against them and to gain experience on different levels. A very interesting question for example is, how users really want to use resources and tools provided in such an infrastructure. While we may be able to specify a few points by just thinking deeply about it, we also know from experience that some important things are always discovered later on, when users are really starting to use a certain product. If we do it right, we will be able to reuse many parts of the implementation of this small solution later on in our project, even if we're using a different, more complex technology in the future.

Another reason for building a prototype in D-Spin results from the fact that Clarin is planed on a thirteen years basis, while work on D-Spin right now is planned for three years. In fact, when D-Spin is over, Clarin will hopefully be reaching it's implementation phase. From a pragmatic point of view, D-Spin can be successful in implementing a basic technical infrastructure oriented on the ideas of Clarin. This infrastructure should be designed in a way that allows it to evolve in the future in order to meet all specifications made in Clarin. In short: While work and research in Clarin is designed and done in a top-down manner, D-Spin may try to follow a bottom-up approach. In fact this enables D-Spin to be successful in delivering a working product in the end, while producing interesting input for Clarin.

1.2 Analysis

It's pretty clear that combining different tools and resources is only possible, if all parts fit together in some way. The next logical question is: What functionality do we need to put those elements together automatically? We will perform a small and very simple analysis in order to find out what information we need and what to do with it. We don't want to put too much thought and especially complexity into this simple infrastructure, because we know this prototypical solution is only a first step and therefore should be kept rather simple. Any other, more complex solution has to

prove that every bit more complexity enables us to solve previously unsolvable problems without too much headache.

Data transfers from one element of a process chain to another have to be possible. In order to do this we need some kind of “common language” that every tool uses to express input and output data. We also need a strong, formal specification of input and output data, in order to make automatic(!) validation of process chains possible. Basically, automatic validation can be reduced to answering the following question for all parts of the chain: “Do these two components fit together?” or more technically: “Is all data this component needs AND understands available at this point of the chain?”. It clearly would be nice if this specification could be derived from sample input and output data of one chain element, because this is what most people naturally do on a piece of paper etc. when asked about this kind of thing.

Although we need a strong specification, we can't afford to be restricted on certain standards, for example, when specifying POS-tags. If people are forced to use only a small set of standards, we're bound to fail. But if it is attractive to build some kind of translation from one standard to another in order to be able to use tools that are only compatible to one of them, people will do it. The only thing that has to be taken care of is that the whole infrastructure and every tool in it automatically profits, once this is done. Shortly put: If we transfer data from one element to another, it has to be possible to encode information pieces in various standards. The specification has to be “strong/formal” on how these standards are specified/encapsulated and how data transfer is done, but not on which standards are to be used in which case!

Another main goal should be that it has to be “simple” (for programmers) to translate from their own service-interface (that can and should be service specific) to the interchange format by just adding a small(!) “abstraction layer”, that translates to and from the internal data-format. This translation has to be done one way or another. Either we specify how our data looks like internally and translation is done automatically, or we translate into one or one of several common interchange formats, but we won't get around it. As a matter of fact people won't be able, or simply might not want, to implement their already existing tools once more in a more compatible, standardized way. Adding a small layer seems to be the easier approach and is the common solution whenever this kind of problem occurs. If we used some kind of data standard in the used/produced data before, we are still able to use it on this level too. To put it more simple: If the basic implementation was done well (a simple and good interface, usage of well known and accepted standards), integration into the infrastructure will be easy.

1.3 Building process chains

If we want to build process chains, all parts that are plugged together have to be validated using their input/output specifications. On some point, human interaction

(decision which data to use) may be needed. In these cases we at least have to be able to automatically offer all possible choices. What we want to achieve is a unified information transfer from one service (-implementation) to another, in order for different tools to work together in one small (test-) infrastructure. In a second step, we will be able to build a tool that automatically puts different (simple) services together, automatically validates a chain, identifies missing links/chain elements and allows us to gather experience on how people are using or want to use such a tool and how we have to built it. This experience on how the end user wants to work, is a very important aspect we should never forget about and that might even, or in my eyes will, have influence on the technical implementation/underlying technologies!

Lets flesh out this idea of a data interchange format and all the interactions needed in order to build a process chain a bit more: What information do we need? Of courses we have to be able to describe a specific piece of data. Basically we need to formally express the following dialog: “What are you?”; answer: “I'm a sentence encoded in utf8.” or “I'm a POS-tag in the STTS[7]-standard.” For example the last component of our example process chain has to answer the following questions:

> “What data do you need?”; possible answer: “I need sentences encoded in utf8 and annotated in ISO639-3 (SIL[8] three letter language code).”
>
> “Which kind of data do you produce?”; possible answer: “I produce tokens annotated using the STTS standard.”

It's pretty clear that there has to be some kind of specified vocabulary for values like “sentence”, “STTS” or “utf8”, but we are free to use ANY standardization for that and we're also able to add new values or standardizations at ANY point in the future. There just needs to be some kind of central registry (or one standardization; but this would be difficult) for this kind of values, because that's the vocabulary a formal validation is done on. Information on who created a specific piece of data will be helpful too, if human interaction is needed (for example, if there is more than one option and the user has to choose which data to use in a specific step).

The next question is: “How does an input/output description for a chain element look like?”. It has to use the specified vocabulary to express which format is used and which data is needed in order to perform a certain task. The description of the output should be described exactly the same way. If there is more than one possibility, for example, if a function of our chain element is able to process data using different standards, there might be more than one input specification. The output specification clearly has to be restricted to one possibility (keep in mind that a chain element might be only one of several functions of a service) or otherwise validation is much more difficult, if not impossible for all following chain elements. When validating a given process chain, the output of every chain element is added to / combined with the last

known input. By doing this we're able to decide after every step, wether all needed information will be available for the next node in the chain.

2 Building a prototype infrastructure

> *It is not the strongest of the species that survives, nor the most intelligent that survives. It is the one that is the most adaptable to change. Charles Darwin*

In order to implement an application that enables a user to easily configure process chains according to his needs, there has to be an infrastructure that consists of several key elements. At first we clearly need some kind of basic service registry/central registry, in order to discover all services currently available on the infrastructure. There are plans in Clarin to use technologies based on PID's on this level. But so far these tools aren't ready to use and there isn't very much experience or a distinct best practice documentation on it. Because of that, we will start with a very simple and pragmatic service registry, based on a webservice and a database in the back-end, that will allow users or automatic tools to register and describe their own services or to search for other available services. In a second step we will make further adjustments to this service registry in order to meet Clarin requirements, especially on metadata level.

The basic functionality we want to achieve is:

- enable the developers of tools or resource-access-APIs to adapt their tools to the D-Spin prototype infrastructure by adding only a small abstraction layer (toolwrapper)
- describe and instantiate process chains
- validate process chains; or more formally: validate if the information needed on a certain “node” of the process chain (input) may be/is available judging from the output specifications of previous services
- in order to do that, we have to be able to extract interface specifications (input/output) of a service from example input/output xml-files

2.1 A toolwrapper layer

The next logical step is to describe how data is interchanged between different resources and tools. As described above, the goal is to specify an architecture, that enables us to describe, validate and instantiate process chains that consists of many different services. These services are using a huge variety of different data formats internally. As it was already pointed out, there has to be some kind of agreement on how certain types of data, like sentences, words, pos-tags etc, are represented in a data structure or otherwise it won't be possible for different tools to communicate with each other. On the other hand it won't be possible to force all involved parties to use

the same data format. At first I started to work on the idea of a container format that allowed to describe the data it contained, using only some very basic knowledge of the task. However the problem of this approach is, that it would be very time consuming to adjust existing tools or resource-access-APIs to compute input and output formated this way. Helmut Schmid, researcher at the IMS, University of Stuttgart, came up with the idea of a "toolwrapper" layer (see Figure 2 in 2.3). While there is still a specification on how data structures are to be designed, this approach is much more flexible and allows to choose a document layout that is much closer to the task at hand. The basic idea is that there might be several XML-formats designed to represent data of a certain type, like text corpora or lexical data. XML was the logical choice to represent this kind of data, because XML technologies like XPath will enable us to easily do complex operations, for example asking if certain elements (subtrees of the XML-element tree) are present, on our data-structure. Every toolwrapper-document consists of a metadata part, a header that describes it's type and content (attributes of `<TextCorpus>`) and a data part that holds the actual data (the content inside of `<TextCorpus>`):

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<CLARIN>
 <MetaData>
  <source>IMS, Uni Stuttgart</source>
 </MetaData>
 <TextCorpus lang="de" encoding="latin1" text="yes" tokens="yes"
       token_boundaries="yes" sentences="yes"
       source="ftp://www.ims.uni-stuttgart.de/pub/D-Spin/example.xml">
  <text>Peter aß eine Pizza. Sie schmeckte ihm.</text>
  <tokens>
   <token ID="t1" start="1" end="5">Peter</token>
   ...
 </TextCorpus>
</CLARIN>
```

While the concrete attributes of `<TextCorpus>`, the tag name itself and the format of the data stored within are part of the agreement of the "TextCorpus" toolwrapper-format, the fact that there is a `<MetaData>`-tag and a part of the document that describes the content is something we expect to find in every other toolwrapper-format. Every node of a process chain defines which kind of toolwrapper-format is accepted and defines which data is needed in the input using the vocabulary defined in it. The input document may contain more elements than specified. These parts of the document will stay untouched. The output will most likely contribute new content to the document and therefore new attributes may be added to or changed in the content specification. But it's also possible to switch from one toolwrapper-format in the input to another one in the output. This might be done by a converter service or a service that accepts data from "one world", like the "TextCorpus"-format in the example above, and produces output using the vocabulary and structure of another one.

Based on this specification and knowledge, the process chain builder and validator will be able to extract input and output specifications from example documents. This is possible by extracting the “type”-tag and it's attributes from example documents given by the tool developer:

```
<service>
 <name>MySentenceSegmentationService</name>
 <input>
  <TextCorpus lang="de" encoding="utf8" />
 </input>
 <output>
  <TextCorpus lang="de" encoding="utf8" sentences="yes"/>
 </output>
</service>
```

Defining and validating process chains is done based on this kind of information. We even should be able to automatically identify and add the different nodes needed to be present in a process chain in order for tool B to work on resource A. Therefore it will be possible to just specify the starting and endpoint of a chain like this: “I want to use this corpus and I want it tagged using this tagger!”. An automatic process chain builder and validator will find all the nodes needed in between those two services, if these are available. We may also think about ranking different service in a way, that allows to learn from the feedback other users gave on a certain combination of process chain elements in the past, in order to choose the right service out of a variety, if there is more than one possibility. While doing all this, we don't need to put too many restrictions on the concrete toolwrapper-formats, allowing them to represent their data in a “close to the task” way. Of course there will be different “bubbles” of compatible services, because there are different toolwrapper-formats and there might even be more than one “TextCorpus” format. In my opinion, it will be one of the tasks of a Clarin-Center[8], or a number of them, to establish a specification process that minimizes the amount of different toolwrapper-formats and also to provide bridges/”translator services” between popular ones, if necessary.

Toolwrappers right now are planned as very simple REST-webservices. They will provide only one function that takes an input document and produces an output document according to the rules described above.

2.2 A D-Spin-webservice

On the next layer we need a basic, abstract D-Spin webservice that will be used to actually build and instantiate the process chains and will also provide functionality to ensure quality of service and implements an authentification system. This service will be the same for every tool or resource. It just has to be configured for every new service that is plugged into the infrastructure. Following the idea of Clarin centers, this will be done by the technical staff of such a center. Tool developers will assist in this task, but they will only provide additional information that is needed in order configurate this common webservice.

Put shortly, this webservice, while configured to work with a certain toolwrapper, offers the same basic functionality every time it is used. Possible functions are:

- String authenticate(...): authenticates a user
- Ticket getTicket(authId): enter a queue, ... (ensure availability of the service)
- String invoke(Ticket ticket, String inputDoc): invoke one interaction with the underlying toolwrapper and the "wrapped" tool/resource
- String getInputHeader(), String getOutputHeader(): input/output specifications for the next layer (may be this will be moved into the service registry)

A sketch of an example configuration file could look like this:

```
...
inputHeaderSpec=exampleInput.xml
outputHeaderSpec=exampleOutput.xml
maxSimultanousTickets=10
maxSimultanousTickesPerUser=4
ticketLeaseTime=30000
memoryPerTicket=1M
authentificationServiceUrl=http://localhost/MyAuthService
toolwrapperURL=http://localhost/MyToolWrapper
...
```

This service will evolve over time in order to fit to the Clarin specifications. It addresses the quality of service issue by using a simple ticket system. Each time a service is called, the caller has to ask for a ticket that allows him to do this call (or a certain amount of calls). While some services might be simple and will allow a huge frequency of calls, others may consume a vast amount of memory or cpu time and therefore should be limited. It will be a difficult task to implement all of the functionality of this service and even configuration of it won't be very easy in some cases. But no tool or resource-access-API developer needs to handle this task and the implementation has to be done only once. If there are new features or updates to the infrastructures architecture or functionality, it is the task of a Clarin center to update all deployed instances.

2.3 Outlook

After these first essential parts of the infrastructure described above are completed, we will implement a simple graphical tool that allows to combine known services by non programmers as it was already described in the introduction. In the end, the prototype architecture will consist of four layers:

1. A graphical tool based on a service builder and validator.
2. A common D-Spin webservice, that handles all the functionality needed to build and instantiate process chains and to ensure quality of service and authentification etc..

3. A variety of toolwrapper formats and toolwrappers for concrete tools/resources.
4. The basic tools and resources that are part of the D-Spin/Clarin infrastructure.

A complete overview of the D-Spin prototype infrastructure may look like this:

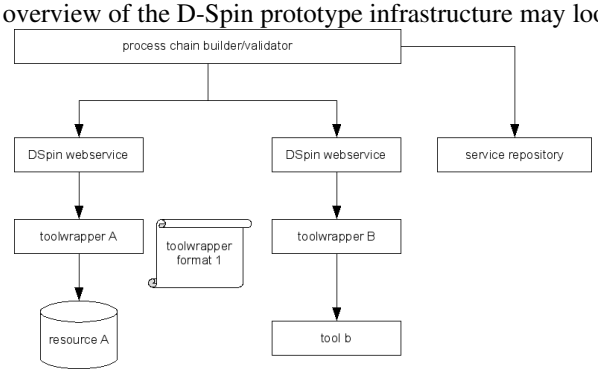

Figure 2: prototype infrastructure overview

3 Conclusion

Starting from the goal described in the introduction, we have pointed out why and how we want to build a prototypical infrastructure. We have sketched which parts of the infrastructure need to be implemented in order to get the core functionality to work. The problem of putting different services together was identified as one of the most challenging features, and we concluded that we need to learn more about it in order to gather real life experience. We don't feel like we have found the solution for all of the problems that might arise, but we want to draw a first simple baseline every other more complex technology can be measured upon. We have addressed the problem of integrating different tools and resources that are based on different standards into one infrastructure by abstracting from it in different data interchange formats. By using the information about the used standards and the contained data we are able to automatically determine if several different parts of a service chain are able to work together or if additional information or data transformation is needed. After we implemented this prototype infrastructure, we will be able to make statements based on real life experience on what problems weren't solvable using such an basic approach, how much time and effort was needed and we will be able to estimate where the boundaries for any solution are. This will allow a much more distinct way of evaluating other technologies and infrastructure layouts in a next step of evolution.

References

1. http://www.sfs.uni-tuebingen.de/dspin/ (2008)
2. http://www.clarin.eu/ (2008)
3. Cristea, D., Pistol, I.C.: Managing Language Resources and Tools using a Hierarchy of Annotation Schemas, LREC (2008)
4. http://gate.ac.uk/ (2008)
5. http://www.research.ibm.com/UIMA/ (2008)
6. http://www.sfs.uni-tuebingen.de/Elwis/stts/stts.html (2008)
7. http://www.sil.org/iso639-3/default.asp (2008)
8. Dirk Roorda, Dieter van Uytvanck, Peter Wittenburg, Martin Wynne: Clarin centres, http://www.clarin.eu/filemanager/active?fid=232 (2009)