SERVICE-ORIENTED ARCHITECTURES: FROM DESKTOP TOOLS TO WEB SERVICES AND WEB APPLICATIONS

Verena HENRICH, Erhard HINRICHS, Marie HINRICHS, and Thomas ZASTROW

University of Tübingen, Department of Linguistics
E-mail: {firstname.lastname}@uni-tuebingen.de

Abstract. The migration of desktop tools to the World Wide Web in the form of web services and web applications has been a significant recent development in many areas of science including computational linguistics. As is often the case with new technologies there seems to be a certain amount of confusion among developers and users alike of these technologies about the meaning of terms such as web services and web applications. These two terms are often used interchangeably although they refer to different concepts and accompanying technical realizations and solutions. The purpose of this chapter is to bring clarity to the four related notions of desktop tools, web services, web applications, and service-oriented architectures using natural language processing tools as an illustrative example. A second and related goal is to explain the significance of the transition from traditional desktop tools to web-based applications for the emerging field of eScience. In practice, this transition amounts to a radical departure from a download-and-install-first paradigm to a seamless integration of multiple external tools into a web-based scientific workflow.

Key words: Service-Oriented Architecture, Web Service, Web Application.

1. INTRODUCTION: MOVING FROM THE LOCAL DESKTOP TO THE WEB

1.1. Background

Before the World Wide Web was established, computer users had command line tools and desktop applications that ran locally on their computers. It was difficult to exchange tools or data over a computer network, as it is now possible through the Internet. As soon as the World Wide Web became popular, web pages were accessible and also provided tools for download. First, these web pages were

static HTML pages, and in case of tools, these were mainly for downloading and local execution. But soon these static pages were replaced by dynamic web sites with dynamic server-side scripting (Common Gateway Interface, CGI). Later, Web 2.0 was born with all its online applications that can be accessed through the web – so called *web applications*. Here, Web 2.0 stands for a new trend in the online world – including new interactive web applications with user-generated content and collaborative techniques for updating web sites. Today, AJAX (Asynchronous JavaScript and XML), which supports an immediate dynamic interaction between a user and a web application (similar to the interaction between a user and a desktop application), is one of the most popular technologies for web applications (Shklar & Rosen, 2009).

1.2. Motivation for Service-Oriented Architectures

Local installations are often not as comfortable as applications that can be accessed through the web. Probably the most obvious reason is because the user has to manage their installation – they need to be downloaded, compiled, installed, updated, etc. Another important drawback of desktop applications is a problem of all digital data: after several years the techniques to read and write old media are no longer available. The resulting challenge is maintainability of applications with an increasing complexity. Distributed and heterogeneous systems need to interoperate in a flexible and scalable way. (Josuttis, 2008)

Service-oriented architectures provide one solution for all of these requirements. A *service-oriented architecture* (SOA) is a system architecture, which represents applications as reusable and publicly accessible services. One feature of a SOA is the possibility of a platform and language independent reuse of distributed services (Melzer, 2008).

Motivation for Service-Oriented Architectures

Service-oriented architectures offer:

- *Transfer of service functionalities from local computers to the web.*
- *Platform and language independent reuse of distributed services.*
- *Interoperability between different systems.*

1.3. The Goals and the Structure of this Chapter

The purpose of this chapter is to bring clarity to the four related notions of *desktop and command line tool*, *web service*, *web application*, and *service-oriented architecture* and gives common examples for them. As is often the case with new technologies, there seems to be a certain amount of confusion among developers

and users alike of these technologies about the meaning of terms. These terms are often used interchangeably although they refer to different concepts.

Section 2 introduces the basic terminology of the technologies covered in this chapter. The terms desktop tool, web service, web application, and service-oriented architecture are explained.

The realization of a service-oriented architecture with web services is the topic of section 3. First, the three roles of a SOA – service provider, service registry, and service consumer – are described together with the descriptions of services. Then, web services are explained in general, before the communication between web services, the orchestration of web services, and the security of web services are examined.

An existing example of a service-oriented architecture is introduced in section 4. The Web-Based Linguistic Chaining Tool (WebLicht) is a SOA, combining several distributed services from computational linguistics. The functionality of WebLicht together with its implementation details are explained.

Finally, section 5 gives a conclusion.

2. TERMINOLOGY

2.1. Desktop and Command Line Tools

At the beginning of the computer age, communication between a user and the computer was based solely on the command line. Later came the first graphical desktop applications that are often still in use today. Both command line tools and desktop applications are executed locally on the computer of a user – in contrast to an application that is executed over a network and can be used in a web browser.

2.2. Web Services

A web service is any service which is available through a network and offers its functionality over this network. Uniform Resource Locators (URL) are (usually) used to address web services. Web services can communicate with each other through standardized interfaces. Internet-based protocols like HTTP are used to send standardized messages between web services. Web services are not bound to a single operating system or programming language. (Cerami, 2002)

There are many different descriptions of web services. In this chapter, the term *web service* is used with the following meaning:

Definition 1: Definition of a Web Service

> *A web service serves functionality through a standardized set of interfaces and can be accessed through a network. Web services support machine-to-machine communication. They can be invoked by a web application or other web services and are often part of a service-oriented architecture.*

Many definitions of web services, like the one from the World Wide Web Consortium[1] (W3C), make reference to the technologies SOAP, WSDL, and UDDI. These technologies are described in more detail in section 3 below. However, there are also other technologies for implementing web services, *e.g.* XML-RPC or REST. Web services are not necessarily bound to these technologies, but are defined more broadly. See section 3 for more details of the general idea of web services as well as possible underlying technologies.

A popular example of a web service is Google Translate[2] – a free online language translation service from Google. Google Translate instantly translates text and web pages into more than 50 different languages. This web service provides a simple application programming interface (API) that can be used by web applications to include the translation web service. Google itself provides a publicly available web application (at http://translate.google.com) that uses the web service. See section 3.6 for a demonstration of this web service.

2.3. Web Applications

An application that can be accessed through the Internet is called a *web application*. This includes applications that are executed in a client's (user's) web browser, for example HTML sites with JavaScript, Java Applets, Java Web Start, Flash, or Rich Internet Applications.

Definition 2: Definition of a Web Application

> *"What is a "web application"? It is a client-server application that uses a web browser as its client program. It delivers interactive services through web servers distributed over the Internet (or intranet). A web site simply delivers content from static files. A web application can present dynamically tailored content based on request parameters, tracked user behaviors, and security considerations."*
>
> (Shklar & Rosen, 2009)

[1] See http://www.w3.org/TR/ws-arch/
[2] See http://translate.google.com

A web application is not necessarily limited to a single web service but can also combine the functionality of several web services. Thus, a web application can be described as the combination of a browser-based user interface which accesses web services in the background[3]. This scenario is illustrated in Figure 1. Especially if more than one web service is used in a web application, these web services need to be coordinated and combined somehow, which is indicated by the orchestration box. This orchestration part may either represent a complex combination algorithm or simply the logic behind a web application.

As web browsers are available on virtually every computer today, web applications are widespread and very popular. This is because users do not need to download and install programs before execution. Due to their platform independence web applications can be easily used in heterogeneous system environments.

Some common web applications include webmail, online shopping sites, or e-learning platforms. Web applications are also becoming popular in the computational linguistics domain. *Das Wörterbuch der deutschen Sprache des 20. Jahrhunderts*[4] (DWDS) is an example of a web application from the field of computational linguistics. It represents a digital dictionary system based on large amounts of electronically available corpora. The web application can be executed in a web browser and allows searching and visualizing words. DWDS combines four lexical information types: dictionary articles, automatically generated information about synonyms, hypernyms, and hyponyms, keyword-in-context examples (co-occurrences), and statistical concurrency information of neighbored words (collocations).

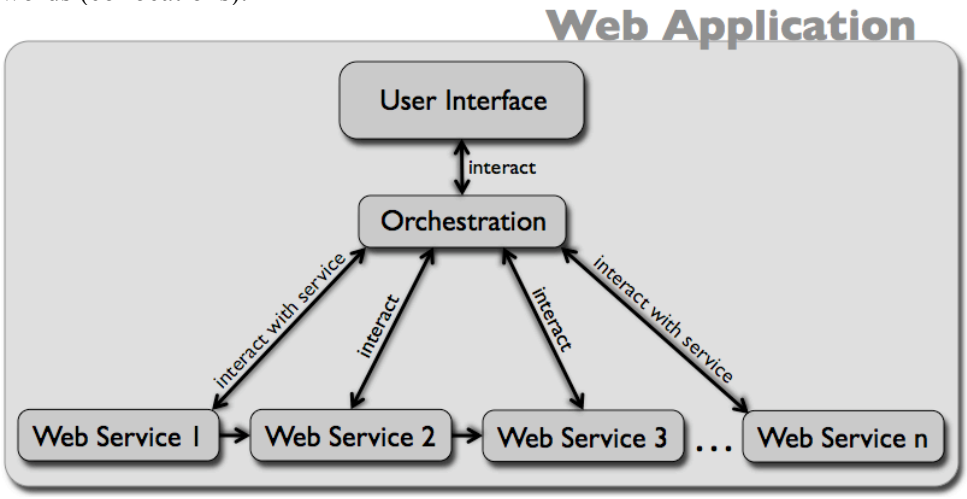

Figure 1. A web application may be represented by a user interface interacting with web services.

[3] Note that there are various possibilities of how a web application can be described. Some may differ fundamentally from the description in this chapter.

[4] See http://www.dwds.de

2.4. Service-Oriented Architectures (SOA)

A service-oriented architecture is a paradigm. It does not represent a concrete technique but an abstraction to a system architecture that can be instantiated in different ways. There is no widely accepted definition of a service-oriented architecture. This chapter follows the definition of Melzer *et al*. (2010), which covers the main features of service-oriented architectures:

Definition 3: Definition of a Service-Oriented Architecture

> *"A service-oriented architecture is a system architecture, which represents diverse, different, and possibly incompatible methods or applications as reusable and publicly accessible services, and thereby offers a platform and language independent usage and reuse."*
> (Melzer *et al*., 2010), translated from German[5]

The most important components are the services, which are all distributed and combined by the SOA into one system. The services of a SOA may be used directly by a user, by a web application, or even by each other. A web application that in turn interacts with the web services may act as an interface of a service-oriented architecture. This composition is illustrated in Figure 2.

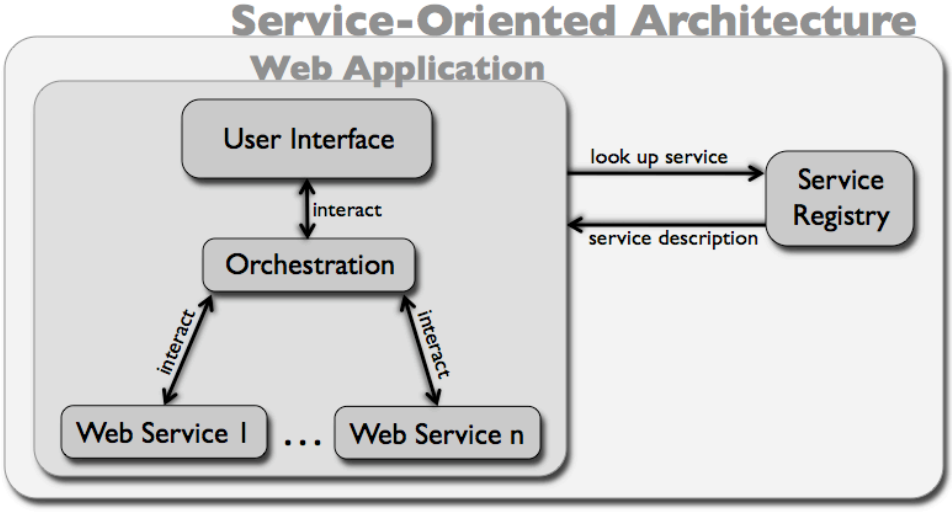

Figure 2. A service-oriented architecture including a web application and web services.

[5] The original definition is in German: *"Unter einer SOA versteht man eine Systemarchitektur, die vielfältige, verschiedene und eventuell inkompatible Methoden oder Applikationen als wieder verwendbare und offen zugreifbare Dienste repräsentiert und dadurch eine plattform- und sprachenunabhängige Nutzung und Wiederverwendung ermöglicht."*

Services in a SOA expose stable interfaces and enable loose coupling. Messaging is the communication paradigm used. The service interfaces are described in terms of the messages a service consumes and produces. A service interface description may be implemented and provided for use on multiple locations by multiple parties.

An example of a service-oriented architecture from the field of computational linguistics is the *Web-Based Linguistic Chaining Tool* (WebLicht), which is explained in detail in section 4.

3. SERVICE-ORIENTED ACHITECTURES USING WEB SERVICES

This section starts with an overview of a service-oriented architecture (subsection 3.1). The three roles of a SOA are outlined in subsections 3.2, 3.3, and 3.4. Subsection 3.5 covers the description of services in a SOA. Both SOAP-based and RESTful web services are introduced in subsection 3.6, followed by descriptions of the orchestration of web services (subsection 3.7) and the security of web services (subsection 3.8).

3.1. Overview of a Service-Oriented Architecture

A service-oriented architecture is an abstract paradigm, which describes an architectural style but not a concrete technique. The most important components of a SOA are the distributed services, which are all combined into one system. The services of a SOA may be used directly by a user, by a web application, or even by each other. They might represent wrappers around command line tools to make them available through the web or they might provide functionality that was formerly served by local desktop applications.

The fundamental features of a service-oriented architecture are (Melzer *et al*., 2010):

- **Loose coupling** and **dynamic binding** of the services: Applications or other web services search, find, and integrate the services dynamically during run-time, which means that – at the time the program is built – it is not known which services will be called later.
- **Service registry**: maintains a list of all registered and available services with the possibility to search for a service.
- **Use of standards**: To allow the use of a service, all interfaces need to be specified in a machine-readable open standard.
- **Reuse** and **simplicity**: The separation of a service interface and its implementation allows the simple reuse of services in different environments.
- **Distribution**: The idea behind a service-oriented architecture is that the services may be distributed at several different locations in the web.

To support these fundamental features, a service-oriented architecture defines three roles: *Service providers*, *service consumers*, and *service registries* (see the following subsections 0, 0, and 0 for descriptions). The workflow – illustrated in Figure 3 – of how these roles interact with each other is as follows (the numbering below corresponds to the numbers in Figure 3):

1. **Register service**: Service providers register their services in a service registry (which may also be provided as a service). This makes it possible to discover the services for a service consumer.
2. **Look up service**: The service consumer searches for a service in the list of services located at a service registry.
3. **Send service location**: The answer of the service registry to the service consumer contains references to the description of the service in question and to the service itself.
4. **Request service**: The service consumer decides which service to use with the help of its description. After selecting a service to be used, the service consumer uses the address of the web service to request the responsible service provider for the service.
5. **Interact with service**: The service consumer uses a service by invoking it. Now, both service consumer and provider interact with each other. Note that the services are not downloaded.

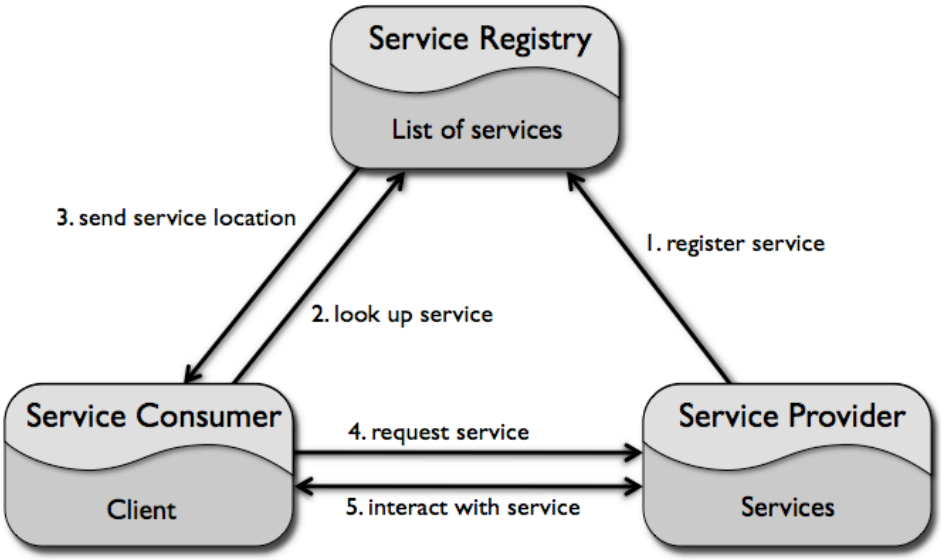

Figure 3. The three roles of a SOA with their interactions.

3.2. Service Provider

The services themselves are offered by service providers and are made available through a common online platform. Before the services can be used, the service provider needs to register them in a service registry to make them available for public searches. Besides offering the services, the service provider needs to maintain the services and keep them up to date. Further tasks concern a quality of service assurance and the security of the services, which includes authentication and authorization. (Melzer *et al*., 2010)

In summary, service providers act as hosting servers (offer services), as managers (register services), and as administrators (update services and provide quality of service and security of the services) all at once.

3.3. Service Registry

The primary task of a service registry is to offer the service consumer the possibility to find services. Therefore, the service providers need to register their services in such a registry. Information about the services – especially references to the service interface descriptions and to the proper services – are stored in a service registry. The services themselves are not stored in a service registry – the registry only stores information which service provider maintains a service and where it can be found. A service registry is sometimes also called broker, discovery component, or repository.

The service registry offers loose coupling, but, in practice, if service consumer and service provider know each other, the service registry is often left out. This saves resources and reduces the execution time, but it also reduces flexibility, because changes at the services need to be distributed not only through the service registry. This disadvantage is expensive especially on a long-term basis. (Melzer *et al*., 2010)

The technical specification of a discovery service for web services based on SOAP can be described by UDDI (Universal Description, Discovery and Integration). UDDI enables service providers to publish web services and service consumers to find web services. Therefore, both references to the descriptions of web service interfaces and references to the actual implementation of a web service are stored. This information comes from the WSDL files (see subsection 3.5). (Cerami, 2002)

3.4. Service Consumer

The service consumer looks up a web service in a service registry and selects a service with the help of the service description. The provider of a service does not need to be known to the service consumer beforehand – the important part is that

there are standards available that describe the interface of a service. Then the service consumer requests and binds a web service from the responsible service provider, before the service can be used. The communication between service consumer and service provider takes place through a protocol that is known to both. It is – especially for the service consumer – very important to have open standards for the description of a service: to find a service, to know how to invoke the service, and to know how to communicate with the service. (Melzer *et al*., 2010)

3.5. Service Description

A service description allows a service consumer to find, invoke, and use a service. The description should be in an open standard, independent from the implementation, the programming language, and the underlying architecture. (Melzer *et al*., 2010)

The Web Services Description Language[6] (WSDL) is an example of such a standard to describe the interface of a web service (Melzer *et al*., 2010). WSDL files are encoded in XML and describe the functionalities of SOAP-based web services (Richardson & Ruby, 2007). (See subsection 3.6 for more information on SOAP.) A WSDL file can be requested by a web service and it can be used by a software developer to implement a web application. The WSDL specification consists of six major elements (Cerami, 2002):

- *Definitions*: The root element of all WSDL documents is the definitions element, which defines the name of the web service and declares namespaces.
- *Types*: All data types are described by the types element.
- *Message*: All messages that will be transmitted, are described by the message element.
- *PortType*: The portType element defines all supported functions and operations.
- *Binding*: This element specifies how the web service needs to be implemented and how messages will be transmitted. Therefore, it may include SOAP-specific information.
- *Service*: The address – usually a URL – where a web service is located and where it can be invoked is encoded in the service element.

An example of a WSDL file is given in the Appendix at the end of the paper. The equivalent of WSDL for RESTful web services is the Web Application Description Language[7] (WADL). (See subsection 3.6 for more information on REST.)

[6] See http://www.w3.org/TR/wsdl20/
[7] See http://www.w3.org/Submission/wadl/

3.6. Web Services

There is a wide range of alternative technologies that can be used to realize distributed services. Especially the choice of the standardized messaging system differs. Earlier implementations of distributed services were based on CORBA or COM, but these technologies were not bound to the web. Web-based distributed services may be realized by the following:

- **XML-RPC**: *XML Remote Procedure Calls* are a data structure format to represent function calls with their return values. Some web services still use XML-RPC on top of HTTP, but it is mostly outdated. (Richardson & Ruby, 2007)
- **SOAP**: At the beginning, SOAP was an acronym for *Simple Object Access Protocol*, but now SOAP is simply a proper name and not an acronym anymore (Melzer *et al.*, 2010). SOAP is based on RPCs and it describes an XML-based envelope format for exchanging messages between computers (Richardson & Ruby, 2007).
- **REST**: *Representational State Transfer*, uses four standard functions of the HTTP protocol: GET, PUT, DELETE, and POST (Burke, 2007).

The two commonly used messaging formats for web services are SOAP and REST. Both are described in more detail in the following paragraphs.

SOAP

SOAP is a platform-independent XML-based envelope format for exchanging messages between computers. It is possible to use SOAP in various messaging systems and with various transport protocols, but mainly SOAP is based on Remote Procedure Calls and is transported via HTTP. SOAP consists of three major parts: (Cerami, 2002)

- SOAP envelope specification,
- Data encoding rules,
- RPC conventions.

To exemplify SOAP, XMethods.net provides a simple weather service example, which gives the current temperature by zip code (see Figure 4). The example is taken from (Cerami, 2002) and Ethan Cerami's web site[8]. There is a method `getWeather` that requires a zip code as a string and returns the temperature as a float value. The corresponding web service description in WSDL can be found in the Appendix at the end of the paper.

[8] See http://www.webservicex.net/WS/faq.aspx

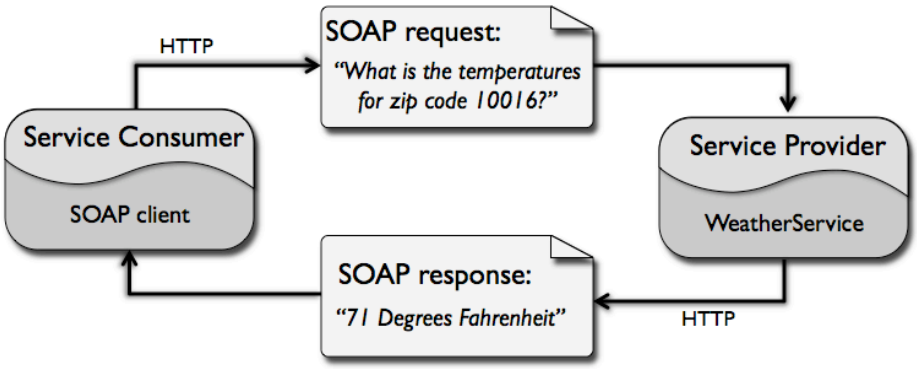

Figure 4. SOAP example: XMethods weather service.

The SOAP request needs to include the name of the method (here `getWeather`) to invoke and all required parameters (here `zipcode`). The following listing shows an example request for the XMethods weather service:

```
<?xml version='1.0' encoding='UTF-8'?>
<SOAP-ENV:Envelope
   xmlns:SOAP-ENV="http://www.w3.org/2001/09/soap-envelope"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xmlns:xsd="http://www.w3.org/2001/XMLSchema">
   <SOAP-ENV:Body>
      <ns1:getWeather xmlns:ns1="urn:examples:weatherservice"
       SOAP-ENV:encodingStyle="http://www.w3.org/2001/09/soap-encoding/">
       <zipcode xsi:type="xsd:string">10016</zipcode>
      </ns1:getWeather>
   </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

This request uses XML namespaces and XML Schemas. The corresponding SOAP response might look like this:

```
<?xml version='1.0' encoding='UTF-8'?>
<SOAP-ENV:Envelope
   xmlns:SOAP-ENV="http://www.w3.org/2001/09/soap-envelope"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xmlns:xsd="http://www.w3.org/2001/XMLSchema">
   <SOAP-ENV:Body>
      <ns1:getWeatherResponse xmlns:ns1="urn:examples:weatherservice"
         SOAP-ENV:encodingStyle="http://www.w3.org/2001/09/soap-encoding">
          <return xsi:type="xsd:int">65</return>
      </ns1:getWeatherResponse>
   </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

REST

REST, which stands for *Representational State Transfer*, is an architectural style for distributed hypermedia systems, defined by Roy Fielding in his dissertation (Fielding, 2000). The implementation of RESTful web services – especially in comparison to the implementation of SOAP-based web services – is simpler. The architectural principles of REST are the following (Burke, 2007):

- *Addressable resources*: Each resource needs to be addressable through a URL.
- *A uniform, constrained interface*: A small set of well-defined methods are used to access resources. This means that only the HTTP methods are used, mainly GET, PUT, DELETE, and POST.
- *Representation-oriented*: A service may have different implementations of the same representation.

An example from the computational linguistics of a RESTful web service is Google Translate. The API specification of this service is available online[9]. The URL that addresses the language translation service (`http://ajax.googleapis.com/ajax/services/language/translate`) can be extended with parameters to translate a string. For example, the string *Hello World* can be translated from English to German by the following extended URL:

```
http://ajax.googleapis.com/ajax/services/language/translate?v=1.0&q=
Hello%20world&langpair=en%7Cde
```

The string to be translated – *Hello world* – is encoded by the extension `q=Hello%20world` where `%20` represents a space sign. Similar is the specification of the language pair, which is *from English to German* in this example, encoded by `langpair=en%7Cde` where `%7C` represents a vertical bar. The corresponding translation from *Hello world* into German is *Hallo Welt*, which can be seen in the result:

```
{"responseData": {"translatedText":"Hallo Welt"}, "responseDetails":
null, "responseStatus": 200}
```

The user interface of Google Translate (see http://translate.google.de) provides interaction to the translation web service for humans. An easy way for entering the input parameter (text to translate and language pair) is offered. The URL to call the service is produced from the user-specified input and the service is called. The response string of the service is then parsed and presented to the user.

[9] See http://code.google.com/intl/de-DE/apis/ajaxlanguage/documentation/

3.7. Orchestration of Services

A SOA consists of several services, which need to be composed and executed in a workflow. A *workflow* describes the combined execution of several services in succession. The term *orchestration* describes the conditions (the execution order and input parameters) under which the individual web services are executed, *i.e.* the business logic, and the communication between the web services. (Melzer *et al.*, 2010)

The Business Process Execution Language (BPEL) is an XML-based OASIS standard[10] that allows the specification of web service interactions. With the help of BPEL, workflows consisting of web services can be described.

3.8. Security of Web Services

Service providers need to be concerned about the security of their web services and servers. Otherwise, everyone who knows the URL of a web service might use it without permissions. There are a few established techniques for building a security architecture for web services, *e.g.*:

- *SAML*: Security Assertion Markup Language, is an XML-based OASIS standard[11] for exchanging authentication and authorization data between a service consumer and a service provider.
- *Certificates*: Web services can be made secure with digital certificates that verify the identity of a service consumer through a third-party such as Verisign[12]. In case the service consumer requests a web service, the web service checks the certificate and reacts accordingly.
- *WS-Security*: Web Services Security, is an extension to SOAP that allows an easy and flexible use of existing security techniques of web services (Melzer *et al.*, 2010). WS-Security is also an OASIS standard[13].

4. EXAMPLE SOA: THE WEB-BASED LINGUISTIC CHAINING TOOL

The Web-Based Linguistic Chaining Tool (WebLicht; (Hinrichs *et al.*, 2010)) is a service-oriented architecture for linguistic analysis of text. A linguistic analysis of text usually starts with tokenizing a text before it can be part-of-speech tagged, parsed, and so on. In a desktop environment, this process is often time-consuming

[10] See http://www.oasis-open.org/specs/#wsbpelv2.0
[11] See http://www.oasis-open.org/specs/#samlv2.0
[12] http://www.verisign.com
[13] See http://www.oasis-open.org/specs/#wssv1.0

because a user has to install all necessary tools. Difficulties often arise when the result of one tool needs to be given as input to another tool. The problem stems from the fact that these tools do not share a common format for representing linguistic annotation layers. WebLicht solves this problem with a common, standard-conformant data format (see subsection 4.4) and allows the user to run several linguistic tools like tokenizers, taggers, or parsers – implemented as web services – to be executed in succession. It is possible to build a chain of linguistic web services (similar as with Gate and UIMA).

WebLicht is implemented as a web application so that users do not need to download and install any software on their own computers or to concern themselves with the technical details involved in building linguistic tool chains. The web application ensures that the tool chains are valid, generates the calls to the web services, and displays the results.

4.1. The Web Services

WebLicht provides several linguistic web services like tokenizers, taggers, or parsers for different languages. In all, there are about 70 web services for seven different languages[14]. The web services themselves are provided from many different service providers.

The web services in WebLicht make functionality of existing desktop applications and command line tools, which was only available locally, available through the web. WebLicht is part of a prototypical infrastructure that was developed to facilitate chaining of LRTs (language resources and tools) services. WebLicht allows the integration and use of distributed web services with standardized APIs. The nature of these open and standardized APIs makes it possible to access the web services from nearly any programming language or shell script.

All web services within WebLicht are implemented as RESTful web services. They are all independent of programming language and server environment. The communication between the web services is based on the TCF data format (Heid *et al*., 2010). (See subsection 4.4 for details about TCF.) This format is given as input to and produced as output from every web service (except the first one which takes plain text as input).

4.2. The Registry

Every web service in WebLicht is registered in a central registry, which is located in Leipzig. In this registry, metadata and processing information (with input and output specifications) about every registered web service are stored. The metadata includes *e.g*. information about creator, name, and address of a service.

[14] As of this writing, June 2010.

The input and output specifications of each web service are required in order to determine which processing chains are possible. Combining the metadata and the processing information, the repository is able to offer functions for the chain building process.

The registry consists of a relational database containing all the information about the web services. Metadata and processing information about every registered web service can be obtained through web services. Therefore, the registry itself is also realized as a RESTful web service.

4.3. The WebLicht User Interface

The WebLicht web application[15] is developed and hosted in Tübingen. The user interface is implemented with Web 2.0 technologies, using an AJAX driven toolkit. It incorporates the Java EE 5 technology and can be deployed in any Java application server.

Figure 5 shows a screenshot of the WebLicht web interface. The right panel entitled *Chain 1* offers the main functionality of WebLicht. It is possible to specify the input text and to orchestrate the chain of linguistic web services.

Input of plain text to the chain can be specified in three ways (in the *Input* tab, hidden in the screenshot[16]): i) entered by the user, ii) file upload from the user's local hard drive, or iii) selecting one of the sample texts offered by WebLicht. Various format converters can be used to convert uploaded files into the data exchange format (TCF) used by WebLicht. Input file formats accepted by WebLicht include plain text, Microsoft Word, and RTF.

The orchestration of the web services is done in the *Tools* tab. The panel *Next Choices* indicates which web services may be chosen next, depending on the previously chosen web services. This list is individually generated by inspecting the metadata of the tools which were already added to the chain. Web services already added to the chain are listed in the *Chosen Tools* panel below.

The chaining mechanism of WebLicht ensures that the list of possible next web services only contains web services that ensure a valid next step in the chain. For example, a part-of-speech tagger can only be added to a chain after a tokenizer was chosen. The metadata of each tool contains information about the annotations which are required in the input data of a web service and which annotations are added by that web service.

---

[15] See https://weblicht.sfs.uni-tuebingen.de

[16] The interested reader might take a look at the actual user interface at https://weblicht.sfs.uni-tuebingen.de to exactly follow all explanations that are not shown in the screenshot.

Valid workflows can combine web services from various service providers. There is no restriction to predefined combinations of web services. This allows users to compare the results of several tool chains and find the best solution for their individual use case.

On the left side, in the *Tool Descriptions* panel, the metadata of a selected web service (author, URL, description etc.) is shown. These data are extracted on-the-fly from the service registry. The final result of running the tool chain as well as each individual step can be visualized with an integrated annotations viewer or in XML (*XML View* panel in Figure 5), or downloaded to the user's local hard drive in WebLicht's own data exchange format TCF (see subsection 4.4).

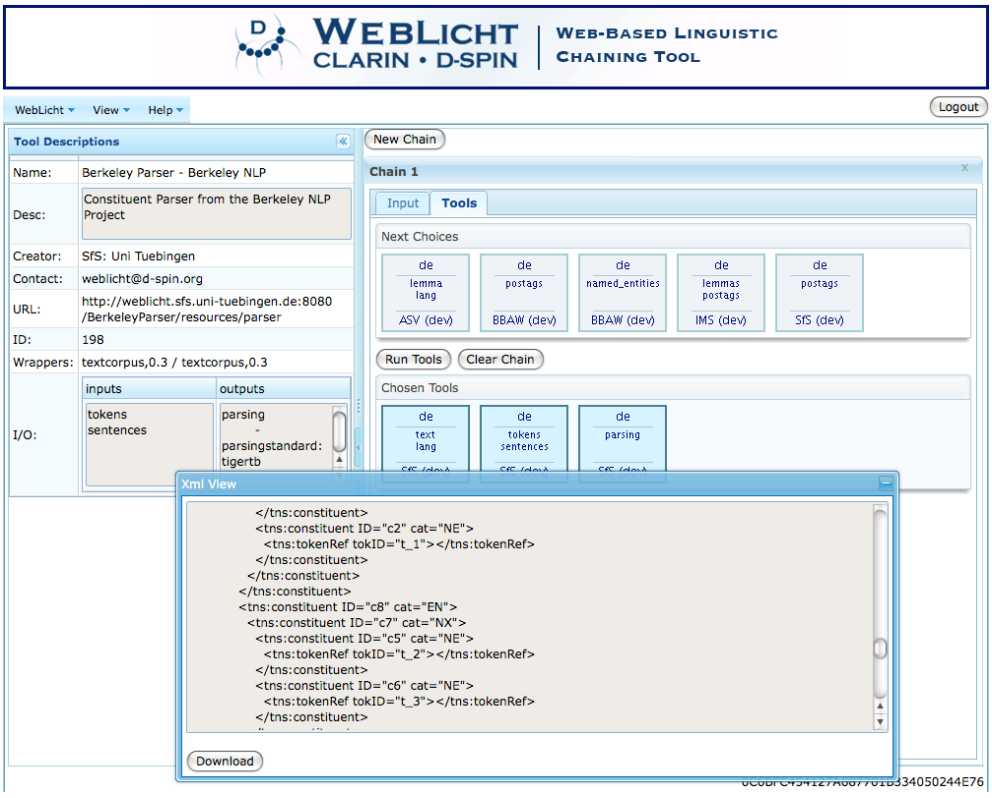

Figure 5. A screenshot of the WebLicht web interface.

4.4. The Text Corpus Format (TCF)

All web services within WebLicht use a common data format for input and output. This Text Corpus Format (TCF; (Heid *et al*., 2010)) is an internal data exchange format based on namespace-aware XML. TCF is conformant with GrAF (Ide & Suderman, 2007), which is an extension of the ISO standard LAF (ISO/DIS 24612:2009). It was designed to efficiently enable the seamless flow of data between the individual services of a service-oriented architecture.

The format stores all linguistic annotation layers in a single file. Annotation layers are accumulated in the outputs of each applied web service call, which means that the format is incrementally enriched by one or more additional layers for every executed web service. The annotation layers are rendered in a stand-off annotation format. Each web service is permitted to add an arbitrary number of layers, but it is not allowed to change or delete any existing layer. Thus, each layer in TCF represents the linguistic annotation result of a single web service and it is possible to trace back the results of every step in the chain.

Figure 6 illustrates the incremental approach of TCF: In step 1, there is only plain text in the TCF document. Step 2 also contains tokens – the linguistic annotation result of a tokenizer – in addition to the plain text. In step 3, a third layer containing part-of-speech tags – the result of a part-of-speech tagger – is shown. Step 4 adds a parse tree layer to the TCF document.

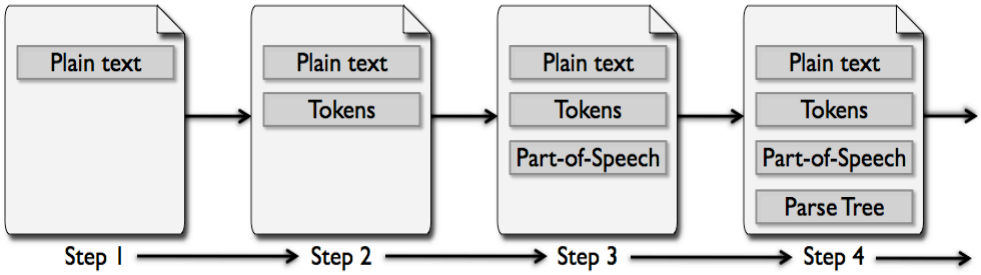

Figure 6. Each web service incrementally adds an annotation layer to TCF.

An example of the Text Corpus Format containing the first three layers is shown in Figure 7. The first layer with the plain text is represented in line 7. Lines 8 to 17 encode the tokens layer, where each token has an `ID` attribute that serves as a unique identifier in other annotation layers. The tokens constitute the central atomic elements in TCF to which other annotation layers refer. The third layer containing the part-of-speech tags (lines 18 to 27), for example, refers to the tokens by the `tokID` attribute.

```
 1 <?xml version="1.0" encoding="UTF-8"?>
 2 <D-Spin xmlns="http://www.dspin.de/data" version="0.3">
 3   <tns:MetaData xmlns:tns="http://www.dspin.de/data/metadata">
 4     <tns:source></tns:source>
 5   </tns:MetaData>
 6   <tns:TextCorpus xmlns:tns="http://www.dspin.de/data/textcorpus"
lang="en">
 7     <tns:text>He buys an apple and a drink.</tns:text>
 8     <tns:tokens>
 9       <tns:token ID="t_0">He</tns:token>
10       <tns:token ID="t_1">buys</tns:token>
11       <tns:token ID="t_2">an</tns:token>
12       <tns:token ID="t_3">apple</tns:token>
13       <tns:token ID="t_4">and</tns:token>
14       <tns:token ID="t_5">a</tns:token>
15       <tns:token ID="t_6">drink</tns:token>
16       <tns:token ID="t_7">.</tns:token>
17     </tns:tokens>
18     <tns:POStags tagset="PennTB">
19       <tns:tag tokID="t_0">PP</tns:tag>
20       <tns:tag tokID="t_1">VBZ</tns:tag>
21       <tns:tag tokID="t_2">DT</tns:tag>
22       <tns:tag tokID="t_3">NN</tns:tag>
23       <tns:tag tokID="t_4">CC</tns:tag>
24       <tns:tag tokID="t_5">DT</tns:tag>
25       <tns:tag tokID="t_6">NN</tns:tag>
26       <tns:tag tokID="t_7">.</tns:tag>
27     </tns:POStags>
28   </tns:TextCorpus>
29 </D-Spin>
```

Figure 7. TCF example with three layers: plain text, tokens, and part-of-speech tags.

5. CONCLUSION

Man-machine interaction has come a long way. Punch cards were used to interact with the very first computers. These were gradually replaced by command line interaction and graphical user interfaces. Now, there are service-oriented architectures that seem to be the logical next step in the evolution. Service-oriented architectures are widely used in business applications but are also of interest in scientific computing in all areas of science. This includes the humanities and social sciences – especially because of the heterogeneity of their research paradigms and accompanying tools and systems. In a SOA the different tools can be made interoperable and can be loosely coupled with each other. This makes it possible to create synergistic workflows which in turn facilitate deeper data analysis and interpretation.

In computational linguistics, language resources and tools can now be made available with service-oriented architectures for a broader range of users than before. The download-and-install-first paradigm is avoided as well, since the functionality in SOAs now allow easy maintenance, updating, and backup. Service-oriented architectures are platform and language independent and make it easily possible to combine distributed services. In contrast to these advantages, the main disadvantage of a SOA is that functionality is only usable online – it binds users to the web. But, in turn, this enables access to cutting edge technologies such as cloud computing.

REFERENCES

1. BURKE B., *RESTful Java with JAX-RS*, O'Reilly Media Inc., Sebastopol, CA, USA, 2009.
2. CERAMI E., *Web Services Essentials – Distributed Applications with XML-RPC, SOAP, UDDI & WSDL*, O'Reilly & Associates, Inc., Sebastopol, CA, USA, 2002.
3. FIELDING R.T., *Architectural Styles and the Design of Network-based Software Architectures*, dissertation, University of California, Irvine, UK, 2000.
4. HEID U., SCHMID H., ECKART K., and HINRICHS E., A Corpus Representation Format for Linguistic Web Services: the D-SPIN Text Corpus Format and its Relationship with ISO Standards. *Proceedings of the Seventh Conference on International Language Resources and Evaluation (LREC 2010)*, Valletta, Malta, 2010.
5. HINRICHS E., HINRICHS M., and ZASTROW T., WebLicht: Web-based LRT services for German. *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics (ACL 2010)*, Uppsala, Sweden, 2010.
6. IDE N. and SUDERMAN K. GrAF, A Graph-based Format for Linguistic Annotations. *Proceedings of the Linguistic Annotation Workshop at ACL 2007*, Association for Computational Linguistics, Prague, Czech Republic, 2007.
7. JOSUTTIS N.M., *SOA in Practice – The Art of Distributed System Design*, O'Reilly Media Inc., Sebastopol, CA, USA, 2007.
8. MELZER I *et al*., *Service Service-oriented Architectures with Web Services*, 3rd ed., Spektrum Verlag Heidelberg, Germany, 2008.
9. MELZER I *et al*., *Service-orientierte Architekturen mit Web Services: Konzepte – Standards – Praxis*, 4th ed., Spektrum Akademischer Verlag Heidelberg, Germany, 2010.
10. RICHARDSON L. and RUBY S., *RESTful Web Services – Web services for the real world*, O'Reilly Media, Inc., Sebastopol, CA, USA, 2007.
11. SHKLAR L. and ROSEN R., *Web Application Architecture: Principles, Protocols and Practices*, 2nd ed., John Wiley & Sons Ltd, Chichester, England, 2009.

ABBREVIATIONS, KEY TERMS, AND DEFINITIONS

- **AJAX**: Asynchronous JavaScript and XML, a Web 2.0 technology for developing dynamic web sites.
- **API**: Application Programming Interface

- **BPEL**: Business Process Execution Language, is an XML-based OASIS standard that allows the specification of web service interactions. With the help of BPEL, workflows consisting of web services can be described.
- **Certificates**: Web services can be made secure with digital certificates that verify the identity of a service consumer through a third-party such as Verisign[17]. In case the service consumer requests a web service, the web service checks the certificate and reacts accordingly.
- **CGI**: Common Gateway Interface
- **COM**: Component Object Model
- **CORBA**: Common Object Request Broker Architecture, is a standard architecture for distributed object systems.
- **Desktop tool**: A tool that is installed and executed locally on a users' computer.
- **DWDS**: Das Wörterbuch der deutschen Sprache des 20. Jahrhunderts, see http://www.dwds.de
- **ESB**: Enterprise Service Bus
- **GrAF**: Graph Annotation Format, is an extension of the ISO standard LAF (ISO/DIS 24612:2009).
- **HTML**: HyperText Markup Language, a popular markup language for creating web pages.
- **HTTP**: Hypertext Transfer Protocol, a standard transfer protocol in the World Wide Web.
- **LAF**: Linguistic Annotation Format, is an ISO standard (ISO/DIS 24612:2009).
- **LRT**: Language Resources and Tools
- **REST**: Representational State Transfer, is an architectural style for distributed hypermedia systems, and one way to implement web services.
- **RPC**: Remote Procedure Call
- **SAML**: Security Assertion Markup Language, is an XML-based OASIS standard for exchanging authentication and authorization data between a service consumer and a service provider.
- **Service consumer**: part of a service-oriented architecture, searches for services and uses them.
- **Service provider**: offers the proper services in a service-oriented architecture.
- **Service registry**: stores information of all services in a service-oriented architecture and offers the service consumer the possibility to find services.

[17] http://www.verisign.com

- **SOA**: Service-Oriented Architecture, is a system architecture, which represents diverse, different, and eventually incompatible methods or applications as reusable and public accessible services, and thereby offers a platform and language independent usage and reuse. (Melzer, 2008)
- **SOAP**: Simple Object Access Protocol (former)
- **UDDI**: Universal Description, Discovery and Integration, describes a discovery service for web services based on SOAP.
- **URL**: Uniform Resource Locator, specifies the location of a resource. A popular example of a URL is the address of a web page in the World Wide Web, *e.g.* the URL to access WebLicht is https://weblicht.sfs.uni-tuebingen.de
- **WADL**: Web Application Description Language, the equivalent of WSDL for RESTful web services.
- **Web 2.0**: In this chapter, Web 2.0 stands for a new trend in the online world – including new interactive web applications with user-generated content and collaborative techniques for updating web sites.
- **Web application**: is a *client-server* application that uses a web browser as its client program. It delivers interactive services through web servers distributed over the Internet (or intranet). A web site simply delivers content from static files. A web application can present dynamically tailored content based on request parameters, tracked user behaviors, and security considerations. (Shklar & Rosen, 2009)
- **Web service**: A web service is any service that is available over the Internet, uses a standardized XML messaging system, and is not tied to any one operating system or programming language. (Cerami, 2002)
- **WebLicht**: Web-Base Linguistic Chaining Tool, see https://weblicht.sfs.uni-tuebingen.de
- **WSDL**: Web Service Definition Language, is a standard to describe the interface of a web service. WSDL files are encoded in XML and describe the functionalities of SOAP-based web services.
- **WS-Security**: Web Services Security, is an extension to SOAP that allows an easy and flexible use of existing security techniques of web services (Melzer *et al.*, 2010). WS-Security is also an OASIS standard.
- **XML**: Extensible Markup Language

APPENDIX: WSDL EXAMPLE

This WSDL example is taken from Cerami (2002) and Ethan Cerami's web site[18]. See that book for descriptions.

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions name="WeatherService"
  targetNamespace="http://www.ecerami.com/wsdl/WeatherService.wsdl"
  xmlns="http://schemas.xmlsoap.org/wsdl/"
  xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
  xmlns:tns="http://www.ecerami.com/wsdl/WeatherService.wsdl"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema">

  <message name="getWeatherRequest">
    <part name="zipcode" type="xsd:string"/>
  </message>
  <message name="getWeatherResponse">
    <part name="temperature" type="xsd:int"/>
  </message>

  <portType name="Weather_PortType">
    <operation name="getWeather">
      <input message="tns:getWeatherRequest"/>
      <output message="tns:getWeatherResponse"/>
    </operation>
  </portType>
  <binding name="Weather_Binding" type="tns:Weather_PortType">
    <soap:binding
      style="rpc" transport="http://schemas.xmlsoap.org/soap/http"/>
    <operation name="getWeather">
      <soap:operation soapAction=""/>
      <input>
        <soap:body
          encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
          namespace="urn:examples:weatherservice" use="encoded"/>
      </input>
      <output>
        <soap:body
          encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
          namespace="urn:examples:weatherservice" use="encoded"/>
      </output>
    </operation>
  </binding>
```

[18] See http://www.webservicex.net/WS/faq.aspx

```
  <service name="Weather_Service">
    <documentation>WSDL File for Weather Service</documentation>
    <port binding="tns:Weather_Binding" name="Weather_Port">
      <soap:address
        location="http://localhost:8080/soap/servlet/rpcrouter"/>
    </port>
  </service>
</definitions>
```