Web-based technology for storage and processing of multi-component data in seismology

–

First steps towards a new design

zur Erlangung des Doktorgrades der Fakultät für Geowissenschaften

der Ludwig-Maximilians-Universität München

vorgelegt am 16. September 2009 von

Robert Barsch

1. Gutachter: Prof. Dr. Heiner Igel

2. Gutachter: Prof. Dr. Hans-Peter Bunge

Tag der mündlichen Prüfung: 21.12.2009

Summary

Seismic databases and processing tools currently available are mainly limited to classic three-component seismic recordings and cannot handle collocated multi-component, multi-disciplinary datasets easily. Further, these seismological databases depend on event-related data and are not able to manage state of the art continuous waveform data input as well. None of them allows for automated request of data available at seismic data centers or to share specific data to users outside one institute. Some seismic databases even depend on licensed database engines, which contradicts the open source character of most software packages used in seismology.

This study intends to provide a suitable answer to the deficiencies of existing seismic databases. SeisHub is a novel web-based database approach created for archiving, processing, and sharing geophysical data and meta data (data describing data), particularly adapted for seismic data. The implemented database prototype offers the full functionality of a native XML database combined with the versatility of a RESTful Web service. The XML database itself uses a standard relational database as back-end, which is currently tested with PostgreSQL (http://www.postgres.org) and SQLite (http://www.sqlite.org). This sophisticated structure allows for the usage of both worlds: on the one hand the power of the SQL for querying and manipulating data, and one the other hand the freedom to use any standard connected to XML, e.g. document conversion via XSLT (Extensible Stylesheet Language Transformations) or resource validation via XSD (XML Schema). The actual resources and any additional services are available via fixed Uniform Resource Identifiers (URIs), where as the database back-end stores the original XML documents and all related indexed values. Indexes are generated using the XPath language and may be added at any time during runtime. This flexibility of the XML/SQL mixture introduced above enables the user to include parameters or results as well as meta data from additional or yet unknown monitoring techniques at any time. SeisHub also comprises features of a "classical seismic database" providing direct access to continuous seismic waveform data and associated meta data. Additionally, SeisHub offers various access protocols (HTTP/HTTPS, SFTP, SSH), an extensible plug-in system, user management, and a sophisticated web-based administration front-end. The SeisHub database is an open source project and the latest development release can be downloaded via the project home page http://www.seishub.org.

The SeisHub database has already been deployed as central database component within two scientific projects: Exupéry (http://www.exupery-vfrs.de), a mobile Volcano Fast Response System (VFRS), and BayernNetz, the seismological network of the Bavarian Seismological Service (Erdbebendienst Bayern; http://www.erdbeben-in-bayern.de).

Acknowledgements

During the past few years as a PhD student at the Department of Earth and Environmental Sciences, Geophysics, LMU Munich, I have been given a lot of advice and support from various people around me. Without their kind and helpful input I would not have been able to finish this thesis, and I am very happy to express my gratitude towards them in this section.

First, I would like to express my deepest gratitude towards the people who provided scientific and financial support to made this thesis possible. I would like to thank my advisers Prof. Dr. Heiner Igel and Dr. Joachim Wassermann who initiated this project and supervised my work over the past three years. Financial support was granted by the German Research Foundation (DFG). Further, I especially like to thank Prof. Dr. Hans-Peter Bunge for giving me the opportunity and ongoing support to work in Munich. I also would like to express my gratitude towards the following people: Paul Käufl, Moritz Beyreuther, Lion Krischer, Dr. Jens Oeser, Sven Egdorf, and Prof. Dr. Rocco Malservisi.

During the past few years in Munich many people crossed my way, but only a few left a lasting impression. I really like to thank those people for just being around all this time and enriching my life: my “beloved” flat mates Marv and Markus (finally I may have time to look for my own apartment), all the “nerds” in my RPG groups (especially Frank, Tom, Marcus, Andi, and Andrea), Vonni & friends, Sandra, Tobi, Richie and Anita.

Last but by no means least, I like to thank my parents, my grandmas, my brother and his family, my cousins and all other close relatives for their never-ending encouragement and ongoing support.

Table of Contents

List of Figures

Listings

List of Tables

List of Abbreviations

AJAX	Asynchronous JavaScript and XML
API	Application Programming Interface
BayernNetz	Bavarian Seismological Network
BGR	Bundesanstalt für Geowissenschaften und Rohstoffe
BLOB	Binary Large Object
BOM	Byte Order Mark
CA	Certificate Authority
CDATA	Character Data
CERN	European Organization for Nuclear Research
CGI	Common Gateway Interface
CPAN	Comprehensive Perl Archive Network
CSS	Cascading Style Sheets
DinSAR	Differential Interferometric Synthetic Aperture Radar
DLL	Dynamic Link Library
DMC	Data Management Center
DOAS	Differential Optical Absorption Spectrometer
DTD	Document Type Definition
EIDAC	European Integrated Data Center
FDSN	International Federation of Digital Seismograph Networks
GBSAR	Ground Based Synthetic Aperture Radar
GEOFON	Geoforschungsnetz
GeoTIFF	Georeferenced Tagged Image File Format
GIANT	Graphical Interactive Aftershock Network Toolbox
GIS	Geographic Information System
GPS	Global Positioning System
GSE	Group of Scientific Experts
GUI	Graphical User Interface
HDD	Hard Disk Drive
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
HTTPS	HTTP Secure
IEC	International Electrotechnical Commission
INGV	Istituto Nazionale di Geofisica e Vulcanologia
IPGP	Institut de Physique du Globe de Paris

IRIS	Incorporated Research Institutions for Seismology
ISO	International Organization for Standardization
JSON	JavaScript Object Notation
KML	Keyhole Markup Language
LMU	Ludwig-Maximilians-Universität
NCSA	National Center for Supercomputing Applications
NERIES	Network of Research Infrastructures for European Seismology
NetDC	Networked Data Centers
OASIS	Organization for the Advancement of Structured Information Standards
ORFEUS	Observatories and Research Facilities for European Seismology
ORM	Object Relational Mapper
PEP	Python Enhancement Proposal
PinSAR	Polarimetric Interferometric Synthetic Aperture Radar
PITSA	Programmable Interactive Toolbox for Seismological Analysis
PNG	Portable Network Graphics
PyPI	Python Package Index
RDB	Relational Database
RDBMS	Relational Database Management System
RELAX NG	Regular Language for XML Next Generation
REST	Representational State Transfer
RPC	Remote Procedure Call
RSS	Really Simple Syndication
SAC	Seismic Analysis Code
SDS	SeisComP Data Structure
SEED	Standard for the Exchange of Earthquake Data
SFTP	SSH File Transfer Protocol
SMTP	Simple Mail Transfer Protocol
SOAP	Simple Object Access Protocol
SSH	Secure Shell
STS	Steiner Triple Systems
SVG	Scalable Vector Graphics
SVN	Subversion
TCP/IP	Transmission Control Protocol/Internet Protocol
TDD	Test-driven development
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
USGS	United States Geological Survey

UTC	Coordinated Universal Time
UTF	Unicode transformation format
VFRS	Volcano Fast Response System
W3C	World Wide Web Consortium
XHTML	Extensible Hypertext Markup Language
XLink	XML Linking Language
XML	Extensible Markup Language
XP	Extreme Programming
XPath	XML Path Language
XSD	W3C XML Schema Definition Language
XSLT	Extensible Stylesheet Language Transformation

1 General Introduction

The data volumes in observational and computational seismology are rapidly expanding. This development occurs partly due to ever increasing continuous data of global, regional, and local permanent station density, large scale experiments, and the increasingly important options to generate simulation data that should be stored with the same priority as observations. Furthermore, seismology is moving on beyond data reduction towards complete waveform processing and simulation. It is commonly accepted in the seismological community that the suite of databases and processing tools that were developed in the past decade is now rather outdated and requires novel approaches.

Current seismic databases and processing tools are mainly limited to classic three-component seismic recordings and cannot handle collocated multi-component, multi-disciplinary data, which play an increasingly important role in many fields of Earth sciences, easily. Examples are ground motion recordings combined with atmospheric observations (e.g., precipitation, pressure, temperature), other motion components (rotational motions, tilt, strain) or instrumental characteristics (accelerometer, GPS). Further, seismological databases mainly depend on event-based datasets not able to handle state of the art continuous waveform data input as well as classical seismic data. None of them allows for automated request of data available at seismic data centers or to share specific data to users outside one institute. Finally, some seismic databases even depend on licensed database engines, which contradicts the open source character of most software packages used in seismology.

The points indicated above suggest that it is time to reconsider the software requirements for seismic databases and processing tools from scratch. The software package SeisHub developed during this study aims to overcome many of the deficiencies on the database side. SeisHub is a novel web-based database prototype closely linking data archiving, distribution, and waveform processing with strong emphasis on the field of seismology. However, SeisHub is absolutely not limited to seismological datasets and may be applied in any other field of Earth sciences, mainly because of its very modular architecture, and the possibility to store and index additional or yet unknown data at any time.

SeisHub is written in the Python programming language, and builds on modern, standardized, and open source communication (HTTP-based Web service) and database (XML database interface on top of a relational database) technologies in order to be flexible enough to cope with current and future requirements. The general introduction in this chapter is followed by **Chapter 2**, which gives a compact introduction to the most important XML specifications and technologies used frequently during this study. Further sections cover the topics of Web services and XML databases in detail.

The insights of the second chapter are applied in **chapter 3** in order to introduce core classes and implementation details of SeisHub. The chapter starts with the modular, extensible architecture of SeisHub outlining the concept of services and the underlying component system. Furthermore, the XML database, its elements, and the usage of the associated Web service are explained. Another section covers the real-time indexing of a file-based seismic waveform archive, discusses synthetic seismograms, and gives a preview of recent developments of real-time waveform distribution on a European level. The chapter concludes with an excursion into Extreme Programming, a modern software development approach used during this study, and introduces ObsPy, a newly developed seismological Python package featuring a consistent interface for reading, writing, processing, and imaging seismograms of different standards. The open source library ObsPy is closely connected to SeisHub, but may also be used as standalone product by any observational seismologist.

Chapter 4 highlights two scientific projects in which SeisHub has been applied successfully as central database component. The first application is Exupéry, a mobile Volcano Fast Response System (VFRS) that can be deployed for volcanic monitoring in case of a volcanic crisis or volcanic unrest anywhere in the world. Within Exupéry, SeisHub handles various multi-disciplinary data types, such as event-related and continuous data, ground-based measurements and satellite data, time series, images and models (3D). The second application of SeisHub is BayernNetz, the seismological network of the Bavarian Seismological Service (Erdbebendienst Bayern). SeisHub is currently running parallel to, but will in the future replace the existing event-based seismic database system.

This thesis concludes in **chapter 5** by summarizing the achieved results and major contributions for the seismological community. It will close with ideas for future research or developments in the field of seismic databases.

2 Introduction to XML and Web Technologies

Introduction. The Internet has revolutionized the way the world accesses and shares information. Although it had evolved since the 1960s, the largest impact in its history was the development of the so called World Wide Web (commonly known as the Web) by Tim Berners-Lee and Robert Caillau at the European Organization for Nuclear Research (CERN) in 1991 combined with the release of the first non prototype web browser Mosaic by the National Center for Supercomputing Applications (NCSA) in 1993. The Internet's core components HTTP and HTML are in use on almost every computer nowadays, but several new standards and technologies have emerged over the last decade to fill gaps which cannot be covered by HTML and HTTP. This chapter helps the reader to become familiar with the most important Web and XML technologies, which will frequently appear in this study. Both technology trees are broad topics discussed in countless articles and books. This chapter will not cover every detail of each specification, but gives a brief overview of the most relevant aspects.

2.1 Extensible Markup Language (XML)

The Extensible Markup Language is a vendor-neutral, standardized, general-purpose framework for defining custom markup languages tailored for any kind of information. Markup is hereby defined as additional information appended to a document to enhance its meaning so that it identifies document elements and how they relate to each other [Ray 2001, p. 2]. In contrast to HTML, in XML there is no fixed set of tags or elements. Instead, XML allows developers to invent their own XML language which meeting their needs. The development of XML was started by the World Wide Web Consortium (W3C) in 1996, which resulted in an official W3C recommendation (synonymous with standard) in February 1998 [W3C 2008]. In a remarkably short period, XML has become the “lingua franca” for marking up traditional datasets and exchanging structured information among the Internet [Melton & Buxton 2006, p. xix].

2.1.1 XML Essentials

XML novices tend to have a hard time understanding XML by simply describing it verbally, therefore an example will be given how XML is applied. The following lines are taken from a so called Dataless SEED file – an standard format for seismological time series and linked meta data. Only a subset of this file defined as “Station Identifier Blockette” [IRIS 2009a] is presented.

```
1  000003S 0500096MANZ 049.9861980012.10830000635.00003000
2  Manzenberg,Bavaria, BW-Net~0013210102005,340~~NBW
3  …
```

Listing 2-1: *Excerpt of a Dataless SEED file (dataless.seed.BW_MANZ).*

Any person who is familiar with the (partly binary) SEED file format may recognize the content of listing 2-1 as meta information for a specific seismological station – in this case Manzenberg, Bavaria ("MANZ") of the Bavarian Seismological Network ("BW"). Data values for latitude, longitude, elevation, network code etc. can be extracted by applying knowledge of the underlying structure. People without that expertise have to rely on some form of documentation of the format.

The next example shows another representation of the above station information above in a XML markup called the XML-SEED format. Both file formats (SEED and XML-SEED) will be discussed in chapters 3.5.1 and 3.6 in more detail.

```
<?xml version="1.0" encoding="UTF-8"?>
<xseed version="1.0">
…
  <station_control_header>
    <station_identifier blockette="050">
      <station_call_letters>MANZ</station_call_letters>
      <latitude>+49.986198</latitude>
      <longitude>+12.108300</longitude>
      <elevation>+635.0</elevation>
      <site_name>Manzenberg,Bavaria, BW-Net</site_name>
      <network_identifier_code>1</network_identifier_code>
      <word_order_32bit>3210</word_order_32bit>
      <word_order_16bit>10</word_order_16bit>
      <start_effective_date>2005-12-06</start_effective_date>
      <end_effective_date></end_effective_date>
      <update_flag>N</update_flag>
      <network_code>BW</network_code>
    </station_identifier>
…
  </station_control_header>
</xseed>
```

Listing 2-2: *Excerpt of a XML-SEED file (dataless.seed.BW_MANZ.xml).*

Obviously, this dataset is very verbose, but much more readable, and therefore understandable for humans, which has been achieved by structuring data into a hierarchical tree and by adding descriptive tags around each single data value. This simple transformation process from a more or less obscured dataset to some general self-describing, human-readable text is called marking up data.

XML itself consists primarily of five basic items: elements, attributes, processing instructions, comments and entities. For understanding the basic structure of a XML documents, the following more general example will be used.

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <example>
3    <tag>Hello World!</tag>
4    <tag></tag>
5    <tag />
6    <tag id="test" style='bold'>Text</tag>
7    <!-- comment -->
8    <tag>23 &lt; 46</tag>
9    <tag>
10     <![CDATA[
11       function add(x, y) {
12         return x + y;
13       }
14     ]]>
15   </tag>
16 </example>
```

Listing 2-3: *A general XML example file.*

Element. The combination of opening tag, text and closing tag is called an element, e.g. "`<tag>text</tag>`". Empty elements have no data included, e.g. "`<tag></tag>`" or as a shortcut "`<tag />`". Elements may be nested within another element without overlapping. There must be only one root element in the whole document, in the document above "`<example>`".

Attribute. Elements may either have none, one or multiple attributes, additional information supplementing an element. A single attribute consists of a name, an equal sign and the value given inside apostrophes or double-quotes, e.g. "`id="test"`" or "`style='bold'`". Attributes must be placed in the start-tag and no attribute may appear more than once per tag.

Processing instructions. Text delimited with "`<?`" and "`?>`" contains processing instructions for applications. A very common instruction is the XML declaration shown in the first line of the example above. It reveals the used document encoding and XML version.

Comment. Any text delimited by "`<!--`" and "`-->`" is meant to be a human-readable annotation and will be ignored by any XML parser or tool.

Entity. Certain problematic characters such as the greater-than and less-than characters are illegal within the text values and must be escaped in a so called entity form, otherwise a XML parser could not distinguish between describing tags and contained data. An entity starts with an

ampersand followed by a name and a semicolon, e.g. "`<`" or "`&quot;`". XML also provides the option to store data without using entities by using a character data (CDATA) section. Everything within a "`<![CDATA[`" and "`]]>`" will be interpreted as a text only section by any XML parser. Nested CDATA sections are not possible.

XML documents are commonly classified into two major categories: **document-centric** and **data-centric** XML. One can imagine document-centric XML resources as textual documents such as a Microsoft Word or OpenOffice Writer document, where the order of elements matters. Data-centric XML documents are often used as interchange format between application; reordering the elements does not change the meaning of the document itself.

The next two sections elaborate key advantages and common criticism about XML. Further sections will introduce established XML extensions and provide an insight into the huge topics of validation, data binding and transformation of XML documents.

2.1.2 Advantages of XML

Well-formedness. The XML specification defines that any XML document must follow a set of minimum syntax requirements. A well-formed XML document must start with the XML declaration[1], may have only one single root element, other elements must nest properly within each other etc. Syntactical correctness does not imply that the containing data are valid, but it guarantees that every XML processor can read this document without errors. Non well-formedness is also a typical indicator for an incomplete or incorrect transmission of a XML document.

Application neutrality. As shown before, a human, may read data with a simple text editor, if necessary. However, machine readability is more important – a document in XML format allows tools to process the data in a standardized way. Using a binary format binds the user to a specific application domain within the lifetime of the data, whereas using XML, a standard syntax with verbose descriptions of contained data, allows easier application of the content into other domains.

International language support. XML supports the full Unicode Standard, a consistent digital representation of the world's alphabets (including ideograph and symbol collections) written today, by design. The current version 5.1 of the Unicode Standard consists of more than 100,000 unique characters and symbols [Maier et al. 2009, p. 157]. Unicode itself will not be written directly into a XML document; instead, every document must declare its encoding at the beginning of the document. There are three Unicode transformation format (UTF) algorithms defined for mapping Unicode characters into a sequence of bytes [Korpela 2006, pp. 293-303].

1 This is valid since XML version 1.1, before it was optional.

- UTF-8: variable length encoding from 1 to 4 bytes per character covers Latin based languages effectively.
- UTF-16: uses either 2 or 4 bytes per character.
- UTF-32: uses always 4 bytes per character; therefore it is very easy to process and handle; usable if disk or memory space is no concern and a lot of "uncommon" characters are used.

All three encoding forms represent the same character repertoire and can safely be transformed into each other. UTF-16 and UTF-32 encodings require an additional Byte Order Mark (BOM) in front of the XML declaration.

XML allows the usage of other standard encoding such as ISO/IEC 8859. For compatibility reasons developers should stay with UTF, preferably UTF-8, which is also the default encoding for XML if none is declared.

Platform neutrality. A XML parser needs to be able to process a well-formed XML document in any given byte order (little or big endian) or encoding. The XML standard also takes care of handling of whitespace characters, e.g. the common problem of line breaks in text files on different operating systems (Macintosh `CR`[2] only, Unix/Linux `LF`[3] only, Windows `CR` followed by `LF`). This ensures that any XML document may be written and processed on every computer platform and operating system.

Well supported. XML technologies are omnipresent nowadays: every XHTML conform web page and every news feed relying on the RSS or Atom format are in fact XML documents. There are countless XML based micro formats and markup languages used in (web) applications. Unsurprisingly, there are lots of XML tools, frameworks and libraries for almost every platform and programming language available, both open source and commercial.

Hierarchical structure. The basic syntax of XML allows developers to easily create their own markup language; they are not limited to standard sets of tags and may extend their language later if needed, knowing that all XML technologies will continue to work with it. One specialty of XML is the option to nest elements within other elements. This allows easy object serialization, a quite common application for XML.

Open standard. As already stated at the beginning of this chapter: XML is an open, vendor-neutral, fee-free standard, a point which cannot be stressed enough, especially for a general exchange format.

2 Carriage return, ASCII character `0x0D` or `'\r'`
3 Line feed, ASCII character `0x0A` or `'\n'`

2.1.3 Criticism on XML

Verbosity. XML documents are generally larger as a binary format containing the same data, mainly because XML is a plain text format with additional describing tags. This is particularly true for large redundant datasets such as tabular results from a relational database (RDB) or time series such as seismograms. Verbosity also results in larger processing times and occupation of more disk space. Although disk space is not an issue anymore today, web developers still have to keep track on the amount of data which is sent over the network. Fortunately, modern communication protocols such as HTTP allow to "compress data on the fly, saving bandwidth as effectively as a binary format" [Bos 2003].

Hierarchical structure. People used to working with data stored in a relational format may have a hard time mapping data into a nested, hierarchical structure of a XML document. In fact, XML is ill suited for large sets of normalized tabular data, for it actually encourages people to use non-relational structures (which is another related criticism). Developers need to overcome the temptation to map every piece of information into a single XML document and should use the advantage of interlinking between different resources, e.g. linking to the compact, binary form of time series, but inserting descriptive meta information within the actual XML document.

2.1.4 XML Extensions

The XML specification itself would not have its popularity as a standard exchange format if it was not for many supporting technologies layered above the XML core. This section will briefly introduce the common extensions XML Path Language (XPath), XML Linking Language (XLink), Cascading Style Sheets (CSS), and XML namespaces introduced through various recommendations from the W3C.

XPath. The XML Path Language is the standard technique to query for portions of a single XML document using a path like syntax, called XPath expression [W3C 1999a]. Such expressions are evaluated by simply traversing the hierarchical element tree from the top node down to the leafs and extracting the relevant values. The XPath language plays a key role in many other XML related extensions such as XSLT (chapter 2.1.6) or XML schemas (chapter 2.1.5).

In order to demonstrate the basic usage of this technology, table 2-1 visualizes a selection of XPath expressions, their actual meanings and the results after the expressions are applied on the XML document shown in listing 2-4. The latter contains some arbitrary names and number of inhabitants of fictional cities.

```
<?xml version="1.0" encoding="UTF-8"?>
<cities>
  <city id="1">
    <name>Springfield</name>
    <pop>7392</pop>
  </city>
  <city id="2">
    <name>Gotham City</name>
    <pop>8274527</pop>
  </city>
  <city id="3">
    <name>Bielefeld</name>
    <pop>324912</pop>
  </city>
</cities>
```

Listing 2-4: *XML example containing fictional cities.*

XPath Expression	**Meaning of the XPath expression**	**Return values**

`/cities/city/name`	Select all `<name>` elements that are children of `<city>` nodes of the root element `<cities>`.	`<name>Springfield</name>`
		`<name>Gotham City</name>`
		`<name>Bielefeld</name>`

`//@id`	Return values of all attributes named “id”.	`1`
		`2`
		`3`

`//city[@id="2"]`	Select `<city>` nodes that have the attribute “id” set to “2”.	`<city id="2">`
		`  <name>Gotham City</name>`
		`  <pop>8274527</pop>`
		`</city>`

`sum(//city[@id<3]/pop)`	Return sum of all inhabitants for cities with “id” less than “3”.	`8281919`
`count(//city[pop<500000])`	Count number of cities with less than “500000” inhabitants.	`2`

Table 2-1: *Examples of XPath expressions applied on listing 2-4.*

XLink. The XML Linking Language is an attribute based syntax to hyperlink other resources in XML documents [W3C 2001a]. The XLink specification defines unidirectional (simple) link types, similar to links used in HTML/XHTML documents among the Internet, and bi- and multidirectional links (extended type).

The example 2-5 contains multiple `<page>` elements containing simple XLink hyperlinks by adding the two attributes `xlink:href`, pointing to a web page, and `xlink:type="simple"`, declaring it as an unidirectional link type.

```
<?xml version="1.0" encoding="UTF-8"?>
<bookmarks xmlns:xlink="http://www.w3.org/1999/xlink">
    <page xlink:href="http://www.python.org"
          xlink:type="simple">Python</page>
    <page xlink:href="http://www.seishub.org"
          xlink:type="simple">SeisHub</page>
    <page xlink:href="http://www.postgres.org"
          xlink:type="simple">PostgreSQL</page>
</bookmarks>
```

Listing 2-5: *XML example containing categorized bookmarks using the XLink technology.*

XML namespaces. In addition to XLink, the last example also incorporates the XML namespaces technology. XML languages of different application domains often need to be combined in a single document. In order to prevent naming collisions of element tags or attributes with identical identifiers, the concept of namespaces has been introduced. XML namespaces were first introduced by the W3C in 2006 as an extension [W3C 2006] and has now even been included into the XML core specification since version 1.1.

In the example above the namespace "`xlink`" is declared by the reserved XML attribute "`xmlns`"and the URI "`http://www.w3.org/1999/xlink`". In this manner, any node or attribute starting with the identifier "`xlink`" followed by a colon and the actual node name belong to the same application domain.

CSS. Cascading Style Sheets is a language to describe the presentation layout of a XML document [W3C 2009]. CSS is widely used to style up web pages written in HTML/XHTML, but are not limited thereto. Attaching the style sheet in listing 2-6 to the XML document above results in a standard browser into a formated output similar to figure 2-1.

```
page {
    display: block;
    border: 1px dashed black;
    margin: 5px;
    padding: 10px;
    background-color: #F0F0F0;
}
```

Listing 2-6: *Cascading style sheet example.*

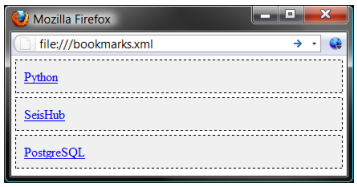

Figure 2-1: *CSS applied on a XML document.*

2.1.5 XML Schemas: Validation & Data Binding

XML itself is a toolkit allowing everyone to generate their own semantically rich markup language. This flexibility results, as intended, in many different types of XML languages and documents. But how does an application know if a class of XML document fits the markup it understands? Similar to human languages, there is a need for some sort of grammar or formal rules to enforce the designated syntax. This gap is filled by XML schema definition languages (XML schemas) which are usually provided as an extra document supplementing the original XML markup language. XML schemas are used to formalize a set of custom rules to constrain a XML document and enforce a specific vocabulary within the hierarchical structure. Additionally, it may be used to dictate the expected data types of elements.

XML schemas are used for two major application fields:

1. **Validation**, the automated verification of content of any XML document by simply applying one or multiple XML schemas (as long the schema language is understood), and
2. **Data binding**, essentially the idea to use the data type information of each element to automatically create mappings from XML documents to structures (usually objects or RDB tables) within an application.

In the past few years, dozens of XML schema definition languages have been designed. Today's most popular schema languages are DTD, XML Schema, RELAX NG, and Schematron. The following sections briefly describe those formats. It should be noted that only the last three schemas are relevant for this study. However, as DTD is the oldest XML schema language, it will be mentioned as well.

Document Type Definition (DTD). Document Type Definition is XML's native schema definition language directly included in the XML core, addressing the document-centric XML approach. Nowadays its popularity is passing as DTD is using a non-XML syntax; therefore an additional DTD parser is needed to extract the information. Also, it has no XML namespaces integration and almost no support for distinguishable data types contrary to the concept of data binding [Melton & Buxton 2006, p. 87; Daconta et al. 2003, p. 39]. Nevertheless, DTD is still

widely used as it is one of the easiest schema languages to apply, partly because of its limited capabilities.

XML Schema Definition Language (XSD or XML Schema). Shortly after the first release of the XML core specification was approved, the W3C started to create the next generation schema language in order to simplify schema generation and to tackle the drawbacks of DTD summarized above. The XML Schema language became an official W3C recommendation on May 2001 comprised of the three separated parts: Primer [W3C 2004b], Structure [W3C 2004c] and Datatypes [W3C 2004d]. Although the resulting specification succeeds in its given design goals to support data types and XML namespaces, it is widely criticized as a highly complex specification [Melton & Buxton 2006, pp. 100-101; Taylor & Harrison 2008, p. 55; Møller & Schwartzbach 2006, pp. 114-115]. Despite this issue, XML Schema is seen as the industry standard, basically because it is ratified by W3C, integrated in many applications and XML tools, and much of its complexity can be faced by using proper tools for schema generation.

Regular Language for XML Next Generation (RELAX NG). RELAX NG is an ISO/IEC certificated schema language which was developed within the Organization for the Advancement of Structured Information Standards (OASIS) as an alternative to W3C XML Schema in 2001 [ISO/IEC 19757-2]. It has been designed with the goal of simplicity and high readability of the format. RELAX NG itself addresses only validation, but within this field it is often considered as easier to understand and technically superior to W3C XML Schema [Evjen et al. 2007, p. 211; Taylor & Harrison 2008, p. 55], but unfortunately it lacks the support of major software vendors [Daconta et al. 2003, p. 39]. RELAX NG does not directly specify data typing, instead it allows the usage of external data type definitions such as the W3C XML Schema Datatypes approach.

Schematron. Schematron is an open source XML validation tool primary using a list of XPath expressions. If a document passes all XPath expressions in this list, it will be considered as valid. This simple, but powerful rule-based approach differs from the three grammar-based concepts introduced above. Schematron is an ISO/IEC certificated schema language [ISO/IEC 19757-3].

It should be noted that schema generation from the scratch can be an utterly complex process even for experienced users. Developers usually rely on some graphical XML schema editor such as Altova XMLSpy[4], Oxygen XML Editor[5], or Liquid XML Studio XSD Editor[6]. Those tools commonly use an instance of an XML document to automatically create a fitting schema. More complex and complete XML documents deliver better schemas. The resulting schema may then be adapted to their needs within the editor.

4 http://www.altova.com/de/xmlspy/xml-schema-editor.html
5 http://www.oxygenxml.com/xml_schema_editor.html
6 http://www.liquid-technologies.com/XmlStudio/Free-Xsd-Editor.aspx

2.1.6 Extensible Stylesheet Language Transformation (XSLT)

One of the most impressive features of the XML technology tree is the Extensible Stylesheet Language Transformation (XSLT) standard, a specification introduced by the W3C in 1999 to convert XML document into another document of even completely different media types [W3C 1999b]. XSLT itself is a functional programming language expressed in XML. Ordinary XML tools may be used to manipulate XSLT documents. The following figure illustrates the basic process flow to produce an output document using a XSLT processor.

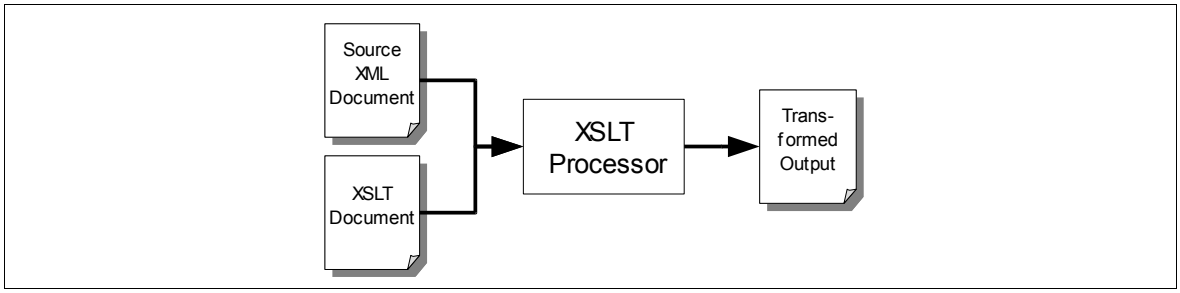

Figure 2-2: *The XSLT transformation process [mod. from Jones & Drake 2002, p. 126].*

It should be noted that there is a significant difference between CSS, introduced in chapter 2.1.4, and the XSLT technology. Cascading Style Sheets simply define look and feel of elements of a markup language, whereas XSLTs are used to transform XML documents into complete new representations not necessarily bound to a XML markup. Resulting documents of a XSLT processor may either be XML-based documents, e.g. XHTML or Scalable Vector Graphics (SVG) images, or plain text-based documents such as tab-delimited text files or SQL statements.

The next two listings contain a simple XML document and a transformation style sheet. The latter is used to convert the XML resource into a SVG graphic (essentially another XML document), which can be viewed in any modern web browser as indicated in figure 2-3.

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <chart>
3    <item>112.98</item>
4    <item>43.23</item>
5    <item>12.98</item>
6    <item>74.00</item>
7    <item>12.98</item>
8  </chart>
```

Listing 2-7: *XML example file for the XSLT technology.*

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0"
                xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
                xmlns="http://www.w3.org/2000/svg">
  <xsl:template match="/chart/item">
    <rect stroke="none" x="50" height="30" fill="blue">
      <xsl:attribute name="width">
        <xsl:value-of select="." />
      </xsl:attribute>
      <xsl:attribute name="y">
        <xsl:value-of select="position()*18-18" />
      </xsl:attribute>
    </rect>
    <xsl:text>&#10;</xsl:text>
    <text x="10" font-size="12">
      <xsl:attribute name="y">
        <xsl:value-of select="position()*18" />
      </xsl:attribute>
      <xsl:value-of select="." />
    </text>
  </xsl:template>
  <xsl:template match="/chart">
    <svg width="100%" height="100%" version="1.1">
      <xsl:apply-templates />
    </svg>
  </xsl:template>
</xsl:stylesheet>
```

Listing 2-8: *XSLT document to generate a SVG image.*

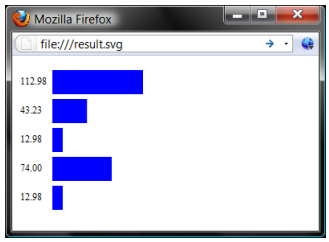

Figure 2-3: *SVG image after transformation from a XML document using XSLT.*

Conversion between different document formats of the same category can be very advanced using the XSLT technology. Instead of converting any format into any other format, which requires up to n*(n-1) different conversion procedures, an intermediary XML document format can be used. XSLT style sheets converting from and into this intermediary format will limit the number of used conversions to 2*n [Buxmann et al. 2003]. New formats can be added at any time with two new style sheets, allowing to convert from and into every other supported format connected to the intermediary format.

2.2 Web Services

There are numerous definitions for Web services in the literature or among the Internet, each slightly different from one another. The World Wide Web Consortium delivered in 2004 the following definition for Web services [W3C 2004a]:

> *A Web service is a software system designed to support interoperable machine-to-machine interaction over a network. It has an interface described in a machine-processable format (specifically WSDL [Web Services Description Language]). Other systems interact with the Web service in a manner prescribed by its description using SOAP-messages [Simple Object Access Protocol], typically conveyed using HTTP with an XML serialization in conjunction with other Web-related standards.*

This definition actually covers only the initial, service-orientated concept of Web services, which is closely coupled to the XML-based SOAP (Simple Object Access Protocol) messaging framework standardized by the W3C [W3C 2007a; W3C 2007b; W3C 2007c]. Such Web services are essentially network-accessible endpoints to Remote Procedure Calls (RPCs) exposed to the clients via a single Uniform Resource Identifier (URI) [Berners-Lee et al. 2005]. Interfacing between Web service and client is achieved using SOAP messages on top of an arbitrary transport protocol such as SMTP or more commonly HTTP as ubiquitous communication protocol in the Web [Graham et al. 2004, pp. 112-114]. If the latter is used as transport protocol, messaging is handled only via the HTTP POST method (see figure 2-4). Because of the vast amount of additional standards supplementing SOAP-based Web services – currently more than 80 specifications partly complementing, overlapping, or competing with each other [Wikipedia 2009] – those are also labeled with the "not entirely kind, but fairly mild nickname: Big Web services" [Richardson & Ruby 2007, p. 299].

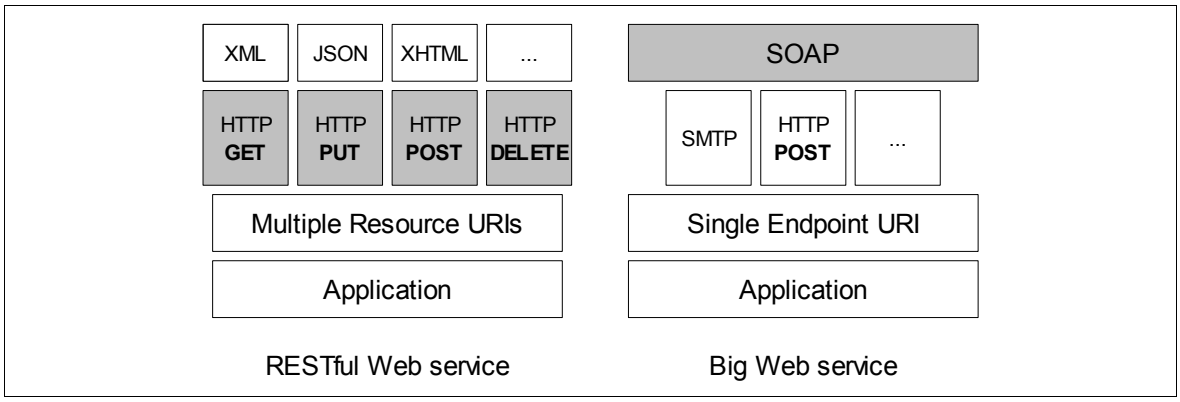

Figure 2-4: *Protocol layering of RESTful (l) and Big Web services (r.) [mod. from Pautasso et al. 2008].*

Recent definitions of Web services include a fundamental different approach based on the REST (Representational State Transfer) architectural style introduced by Roy Fielding in his dissertation [Fielding 2000]. This style follows the resource-orientated structure of the Internet, considering data bound to a specific URI as an unique resource. Communication with such resources is done via a fixed set of operations. Web services implementing Fielding's architectural style on top of the HTTP protocol are called RESTful Web services. Instead of wrapping options and parameters for a remote procedure call into a XML-based SOAP message, RESTful Web services facilitate the standard HTTP methods GET, PUT, POST, and DELETE (see figure 2-4 and table 2-3) on fixed URIs to communicate with a service. It is the responsibility of the Web service to map the combination of HTTP method and URI to a specific functionality within the application. The RESTful approach has got quite popular within the last few years as it allows requesting XML-based data with standard HTTP without the need to apply an additional message layer – essentially the more “natural” way the Internet has been used for decades.

Finally, there are countless RPC-REST hybrid Web services which take paradigm of both worlds and combine them, e.g. presenting the data in REST style, but modifications are achieved using RPC methods. Also, Web services using only the HTTP GET method for resource or application interaction are often named “GETful” Web services [Davis 2008]. However, the author considers any application or software component reachable via a fixed URI over the Internet and providing services for other client applications as a Web service.

Most Web services are not built communicate directly with humans using a web browsers as client. Instead, Web services benefit from the principles, ideas and especially standards of the Internet allowing them to establish interaction between different (web) applications. Whatsoever, web browsers are great testing and demonstration clients for Web services. Although the name may imply it, Web services actually do not necessarily need to run via the Internet. The term “web” refers to the technology and standards used. In fact, many of today's Web services are deployed in an Intranet environment or run as local services. Similar to the relationship between web servers (e.g. Apache HTTP Server[7]) and web browsers, underlying software or hardware details are not relevant for the network interfacing process between server and client. Both may be written in any programming language offering basic HTTP networking support, but they do not have to know about the implementation and deployment details of their counterparts.

As mentioned above, Web services can roughly be categorized into two basic categories: Big (SOAP-based) and RESTful Web services. Both groups will be discussed within the following two subsections. RESTful Web service will be covered in more detail as it is used within this study. The chapter will be concluded by supplementing technologies for Web services.

7 http://httpd.apache.org

2.2.1 Big Web Services

As indicated above, Big Web services are software components which can be evoked usually at a single URI over the network. Communication with such a Web service is done via SOAP messages, basically XML documents wrapping the actual content, processing instructions, error messages, etc. The basic work-flow for a Big Web services using HTTP as transport protocol can be seen in figure 2-5 (using data from example 2-4). SOAP is essentially another RPC specification, similar to CORBA[8] (Common Object Request Broker Architecture) or Microsoft's DCOM[9] (Distributed Component Object Model), but built on top of the XML technology tree.

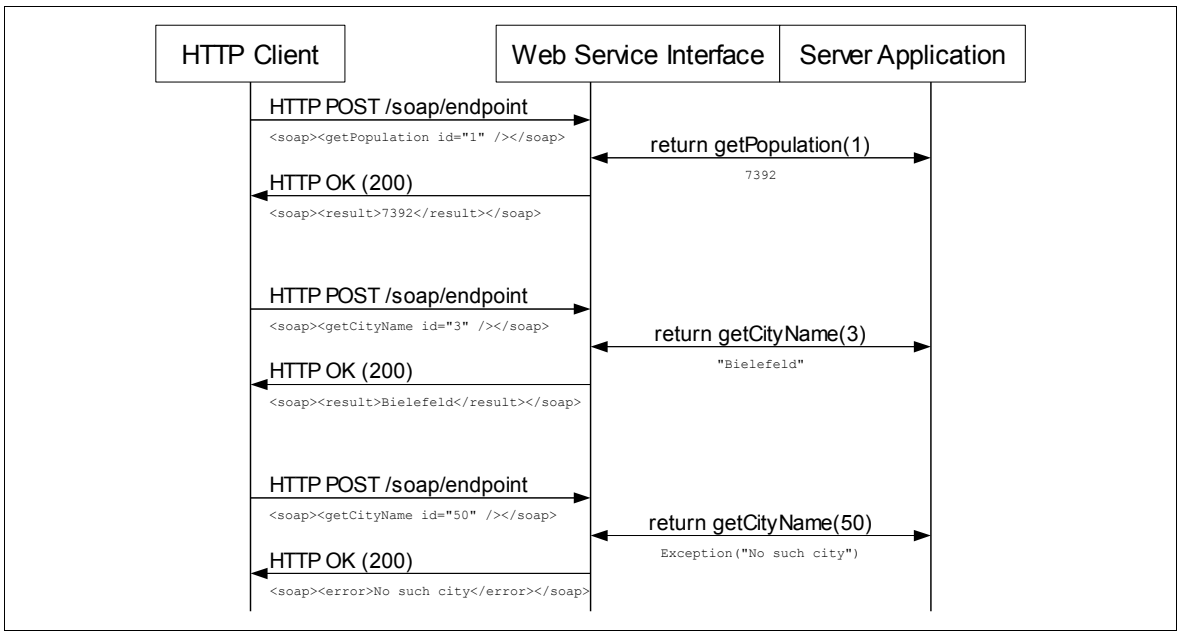

Figure 2-5: *Schematic work-flow of a Big Web service using the HTTP transport protocol. Communication is completely realized using the HTTP POST method and SOAP messages.*

The functionality of Big Web services is described through a XML document written in the Web Services Description Language (WSDL) [W3C 2001b]. Such a WSDL document specifies at what URL the service is accessible, the available transport protocols, the arguments expected by the Web service, and which data will be returned [Buxmann et al. 2003]. WSDL service descriptions can be registered by their service provider within a central UDDI (Universal Description, Discovery and Integration) directory [OASIS 2004]. The idea of such directory was to allow end users or services to discover and even deploy Web services meeting their needs. However, automatic Web service deployment (on global scale) proved as not realizable as there is no standardized way to describe the functionality of a service [Kulchenko & Ray 2002, pp. 5-6].

8 http://www.omg.org/corba/
9 http://msdn.microsoft.com/en-us/library/ms878122.aspx

In order to briefly demonstrate the concepts discussed above, a trivial "Hello" Web service offered by the Faculty of Computer Science, University Vienna will be used as an example. The service endpoint is reachable via http://almighty.pri.univie.ac.at/~mangler/helloService.php offering the RPC method "sayHelloTo" which accepts an arbitrary string as input element and returns "Hello" and the given input text. The corresponding WSDL document specifying this service can be found at http://almighty.pri.univie.ac.at/~mangler/helloService.wsdl (see Appendix A.3). As one can see, a WSDL document can be very complex, even for such simple task. On the other hand, they are detailed enough to automatically create a user interface, e.g. using tools like the web-based Generic SOAP Client (http://www.soapclient.com/soaptest.html) or by the application soapUI (http://www.soapui.org). Such generated clients can never cope with the requirements of a real user interface, however, they may be used to understand or test the functionality of Web services and to inspect the actual SOAP messages. The transmitted SOAP messages for the request (containing the string "TEST") and the corresponding response ("Hello TEST") are shown in Appendix A.3. Manual handling of such messages can be minimized by using SOAP libraries mapping between message entities and program variables.

2.2.2 RESTful Web Services

The REST architectural style considers a resource as basic element of information in a distributed hypermedia system, which is referenced by a globally unique resource identifier. Clients and server retrieving or manipulating such resources communicate over an uniform interface exchanging purely representations of resources. The REST approach is basically a "big picture" of the Internet [Costello 2009] and a guide for "design and development of the architecture for the modern Web" [Fielding 2000, p. 4], but actually not restricted to the Internet or HTTP. Table 2-2 summarizes REST data elements and their complementing elements in the Web.

Data Element	**Modern Web Example**
Resource	The indented conceptual target of hypertext reference
Resource identifier	Uniform Resource Locator (URL)
Representation	HTML document, JPEG image
Representation meta data	Media type, last-modified time

Table 2-2: *REST's data elements [mod. Fielding 2000, p. 88].*

The REST architectural style and the Internet share the following constraints and characteristics stated by Fielding [2000, pp. 76-78]:

- **Client-server architecture.** A separation of concern using a unified interface effectively allows each component to evolve independently.

- **Stateless interaction.** Each request from clients to the server contain all necessary information to understand the request. No information about the client (state) is stored on the server.

- **Cacheable resources.** Stateless communication allows caching to improve network efficiency. Server responses may be marked as cacheable or non-cacheable.

- **Uniform resource interface.** Interaction with resources is done via a uniform generic interface, such as GET, POST, PUT, DELETE for HTTP [RFC 2616, c. 9].

- **Addressable resources.** The system is comprised of named resources defined by uniform resource identifier, such as URL for the Internet.

- I**nterconnected resource.** Resources are interconnected using hyperlinks allowing to navigate between resources.

- **Layered system.** Immediate components such as proxy or cache servers, load balancers, etc., can be layered between clients and resources.

Web services implementing those REST principles and using HTTP as transport protocol are called RESTful Web services. Functionality of such services is exposed via resources on fixed URIs and the four HTTP methods as seen in table 2-3. A basic work-flow for requesting a RESTful Web service is shown in figure 2-6 (again using data from listing 2-4). As can be seen, HTTP status codes [RFC 2616, c. 10] are used within the response. A RESTful service client knows instantly about the status of its request without the need to look into the content of the response.

HTTP Method	**Description**
GET	Retrieve a representation of a known resource.
POST	Create a new, dynamically named resource.
PUT	Modify a known resource. It is not used for resource creation.
DELETE	Remove a known resource.

Table 2-3: *HTTP methods and description [mod. RFC 5023].*

Similar to WSDL, description languages for RESTful Web services have been proposed, like the more recent Web Application Description Language (WADL) [Hadley 2009]. However, the best form of service description is given via well-written human-readable API documentation, proven by many successful services such as Google Maps API[10] or Amazon Web services API[11].

10 http://code.google.com/intl/en-EN/apis/maps/documentation/
11 http://aws.amazon.com/documentation/

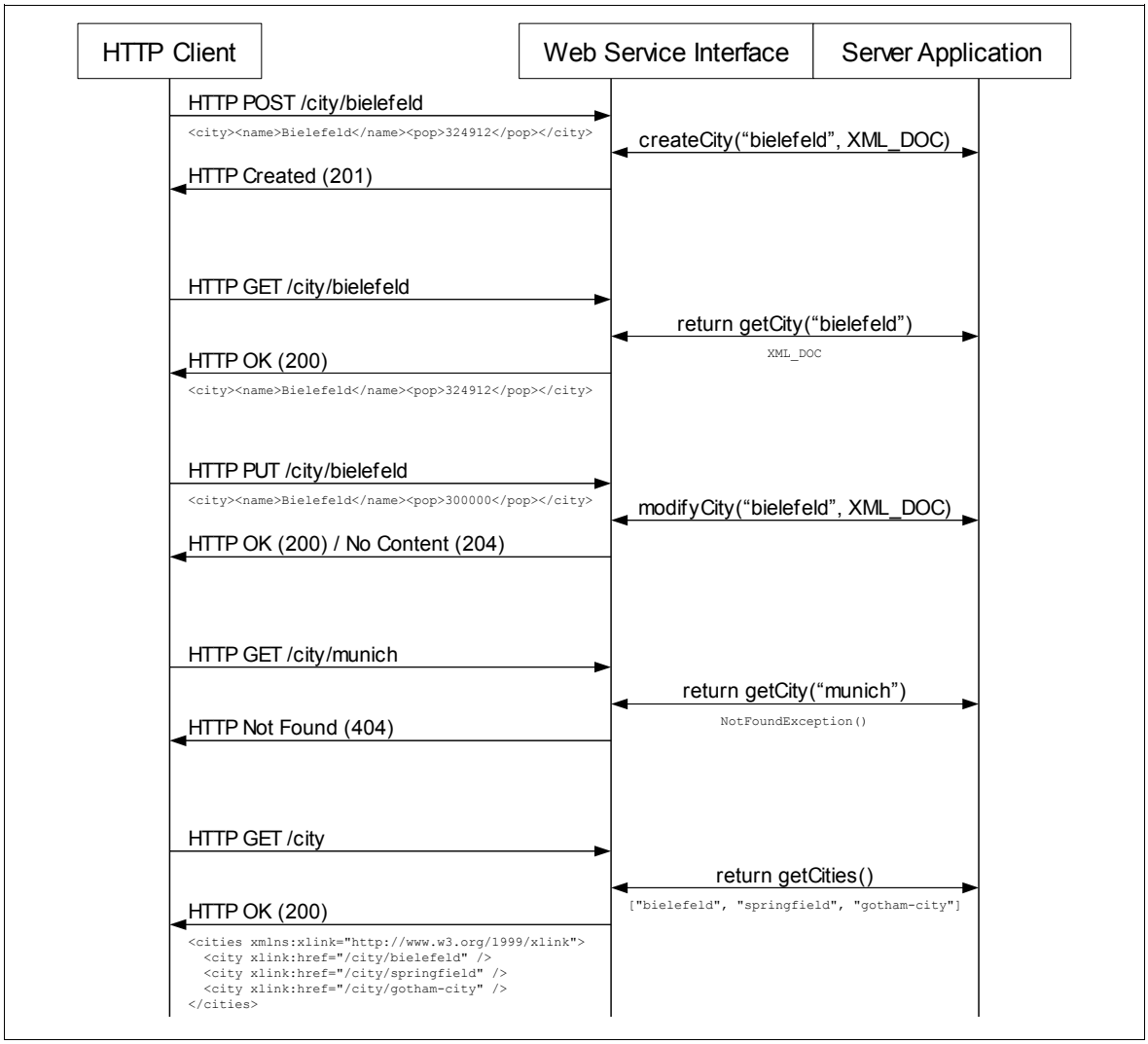

Figure 2-6: *Schematic work-flow of a RESTful Web service. Communication is realized by applying different HTTP methods and HTTP status codes.*

2.2.3 Supplementing Technologies

This chapter includes a few techniques used in common browsers in order to interact with Web services: Ajax and JSON (JavaScript Object Notation).

Ajax. It has been stated before that Web services are not built to communicate directly with a human. The end user is not supposed to call URIs with certain parameters (RESTful style) in a browser or even manually generate SOAP messages for a Big Web service. However, modern browsers are nowadays able to behave like full-fledged desktop applications dynamically and moreover asynchronously interacting with Web services (see figure 2-7). Any such client-side technology communicating with a Web service is summarized as Ajax. Most prominent representatives of Ajax are JavaScript (the standard programming language within browsers), Adobe Flash (browser plug-in), and Java applets (Java programs executed within the browser). The

term Ajax is originally derived from the acronym AJAX (Asynchronous JavaScript and XML), but as the technique is not fixed to JavaScript, XML, and asynchronous request anymore, this term had been decommissioned [Richardson & Ruby 2007, pp. 315-316]. Ajax is the driving technology behind the success of many dynamically rich, interactive web applications such as Google Maps (http://maps.google.de), YouTube (http://www.youtube.com), or the new seismic data portal prototype developed by the Network of Excellence of Research and Infrastructures for European Seismology (NERIES; http://193.52.21.80/jetspeed/portal/).

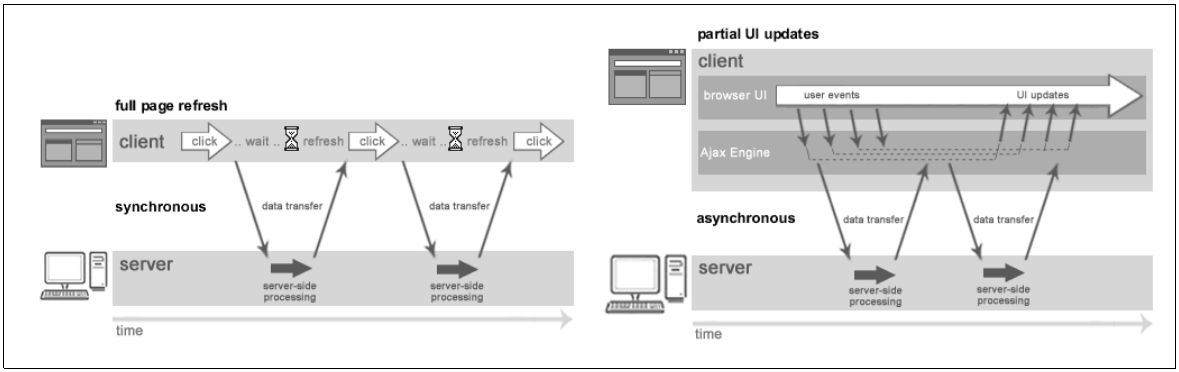

Figure 2-7: *Classic web model (l.), where the user had to refresh the full page manually (synchronous communication) vs. Ajax model (r.) supporting partial web page updates by the browser (asynchronous communication) [Wei 2005].*

JSON. JavaScript Object Notation is a data serialization format originally declared within the JavaScript programming language [ECMA 1999]. It is a language-independent, text-based format supporting simple data types including associative arrays. Today it is often used as a lightweight alternative to XML for data exchange with Web services. Many RESTful Web services offer both JSON and XML-based results. For such services, JSON is commonly applied by Ajax applications as it avoids the more time intensive parsing process of XML-based results. A simple example for the JSON format is given in listing 2-9 showing the same content of a XML document introduced before (listing 2-4).

```
1  {
2    "city": [
3      { "id": "1", "name": "Springfield", "pop": "7392" },
4      { "id": "2", "name": "Gotham City", "pop": "8274527" },
5      { "id": "3", "name": "Bielefeld", "pop": "324912" }
6    ]
7  }
```

Listing 2-9: *JSON example containing fictional cities.*

2.3 XML Databases

Standard databases applying the relational database model [Codd 1970] use tables to store information. Each data field is represented by a column, while datasets are stored in different rows of the table. Additionally, such database system allows the database developer to define relations between two fields of different tables. Storing hierarchically XML documents in a standard relational database is rather problematic because of their irregular structure. The need for databases specialized on handling such XML resources emerged with today's increased usage of XML documents as a common data exchange format. Databases able to directly store and index such documents are commonly called XML databases. There are two major groups of XML databases: native and XML-enabled databases.

XML-enabled databases. In XML-enabled databases, XML documents are transferred into the internal database structure of a traditional database [Bourret 2009] such as tables and fields of a relational database. The transfer process requires mapping from a document specific schema, e.g. using XML Schema, into a database specific schema. This approach depends on heavily structured data and is therefore better suited for data-centric XML documents (see also chapter 2.1.1).

Native XML databases. Semi-structured or document-centric XML resources are better handled by databases using the whole unmodified XML resource as a fundamental storage unit, similar to files in a file system. The underlying database model (relational, object-orientated, etc.) is not relevant for native XML databases.

Additionally, most XML databases offer XPath for querying resources or collection of resources, but many other standards and technologies like XQuery (XML Query Language) [W3C 2007d], XUpdate (XML Update Language) [Laux & Martin 2000], SQL/XML (SQL extension for using XML combined with SQL) [ISO/IEC 9075-1], or XML:DB API (or XAPI) [Staken 2001] are also supported. Common XML technologies for document validation (XML schemas) and transformation (XSLT) as introduced throughout this chapter are usually also used directly within the XML database systems.

SeisHub, the database system developed by the author, was built on the concept of a native XML database. This concept allows storing of completely flexible data structures using XML resources, which was a major goal of this project. The next chapter will elaborate the native, web-based database SeisHub in more detail.

3 SeisHub: A Web-based Database for Seismology

Introduction. The general points and issues raised in the first chapter illustrate the urgent need of a newly designed database structure in seismology. SeisHub, a prototype of a web-based, native XML database developed by the author, aims to be a suitable answer to overcome many of the deficiencies of existing databases. This chapter will briefly introduce the basic concepts of SeisHub and give a comprehensive insight into the architectural and implementation details of its core classes. Further sections will familiarize the reader with the two underlying database concepts for handling XML resources and seismic waveform data. The chapter concludes with related topics: ObsPy, a vital dependency partly developed for SeisHub within this study, and a short excursion into Extreme Programming, a modern software development approach used throughout the whole project. The two projects Exupéry and Bavarian Seismological Network (BayernNetz) and related datasets are used multiple times as examples throughout the whole chapter. Both projects will be discussed in detail in the following chapter "Scientific Application of SeisHub".

3.1 What is SeisHub?

SeisHub is a novel web-based database approach created for archiving, processing and sharing geophysical data and meta data (data describing data), particularly adapted for seismic data. It mainly offers the full functionality of a native, document-centric XML database coupled with the versatility of a HTTP(S)-based RESTful Web service. The XML database itself uses a standard relational database as back-end, currently tested with PostgreSQL[12] and SQLite[13]. This sophisticated structure (see also figure 3-1) allows the usage of both worlds: the power of the SQL for querying and manipulating data and any standard connected to XML, e.g. document conversion via XSLT or resource validation via XML schemas.

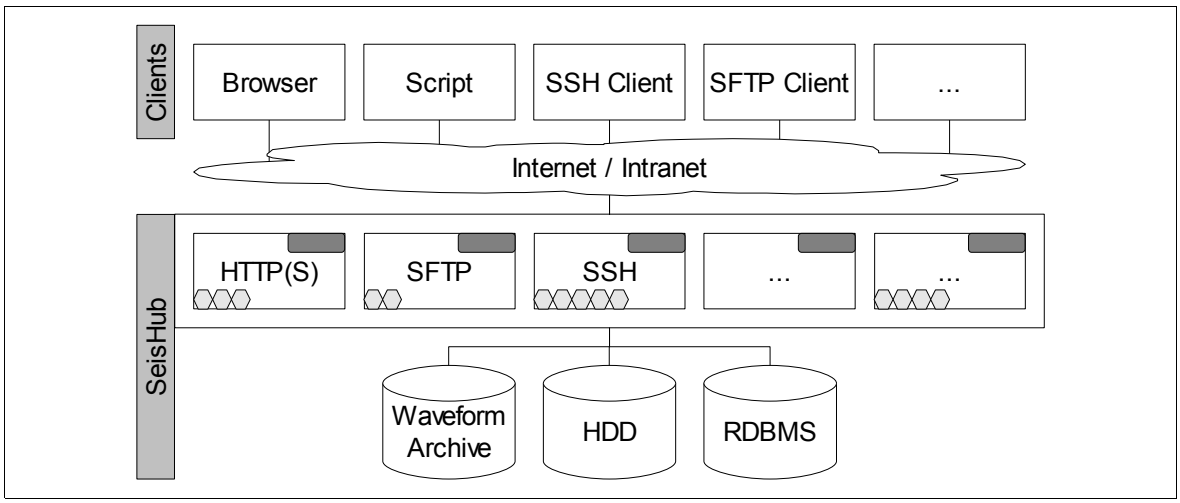

Figure 3-1: *Basic technical architecture of SeisHub.*

12 http://www.postgres.org
13 http://www.sqlite.org

SeisHub further incorporates features of a “classical seismic database” providing direct access to continuous seismic waveform data and associated meta data. Waveform data is handled via the file system, as storing huge amounts of binary and compressed data directly into a relational database is viewed as not very efficient, due to additional database overhead (e.g. requesting time) and limitations (e.g. database size limits).

3.2 Why XML?

The most pressing drawback of current seismic databases is the rigidity of their data structure. As an example, classical Relational Database Management Systems (RDBMS) require a well thought, predefined database schema – a formal definition of tables, fields in each table and relationships between fields and tables – before actually storing any data. Later modifications of this underlying structure, except for simple addition of new fields or tables, is usually difficult to achieve. It requires a person with detailed knowledge of the underlying database system and the used programming language. One could start to create a database schema which fits all possible datasets expected in the near future, but this schema would be outdated as soon as new information has to be incorporated. This deficiency conflicts with the goal of a flexible, future-proof concept for storing collocated, multi-component, multi-disciplinary data within the same database system.

SeisHub chooses to follow a different approach by offering characteristics of a native XML database using whole XML documents as fundamental unit of storage. The actual XML resources are saved unmodified in a single Binary Large Object (BLOB) database field, and only preselected parts of the document are used as search indexes. This allows using the given document structure defined by the resource provider themselves. The documents can be limited or extended from the document provider at any time without necessarily touching the underlying database, e.g. modifying the database schema.

Another reason to work with XML documents as basic storage elements is indicated in the second chapter: XML technology plays nowadays an increasing, if not the leading role as standard exchange language. This also applies in the field of seismology. Best examples for this are the two proposed standards XML-SEED[14], a XML representation of SEED volumes [Tsuboi et al. 2004] and QuakeML[15], at the current state a XML description of seismic events [Schorlemmer et al. 2004]. Using standardized XML documents as storage and exchange resources reduces effectively the number of data format conversion, which is usually not a trivial procedure. Even if conversion from one XML document format into another is required, XSLT as introduced in chapter 2.1.6 offers a unified way to solve that issue.

14 http://www.jamstec.go.jp/pacific21/xmlninja/
15 http://www.quakeml.org

3.3 Technical Details

SeisHub has a modular architecture consisting of a single core class, labeled environment, a handful of distinct services, and many components, also called extensions or plug-ins. The environment object takes care of all start-up procedures, e.g. parsing the configuration file, initializing the connection to the database back-end, and managing all available components. Services are classes implementing either a single network protocol, e.g. HTTP, or processes which have to be called periodically such as a file monitor. Plug-ins are extending one or multiple services with either an URL-based resource such as another administration web page for the HTTP service, or further functionalities, e.g. a new command for the SSH service. It should be noted that not all services offer interfaces for extensions.

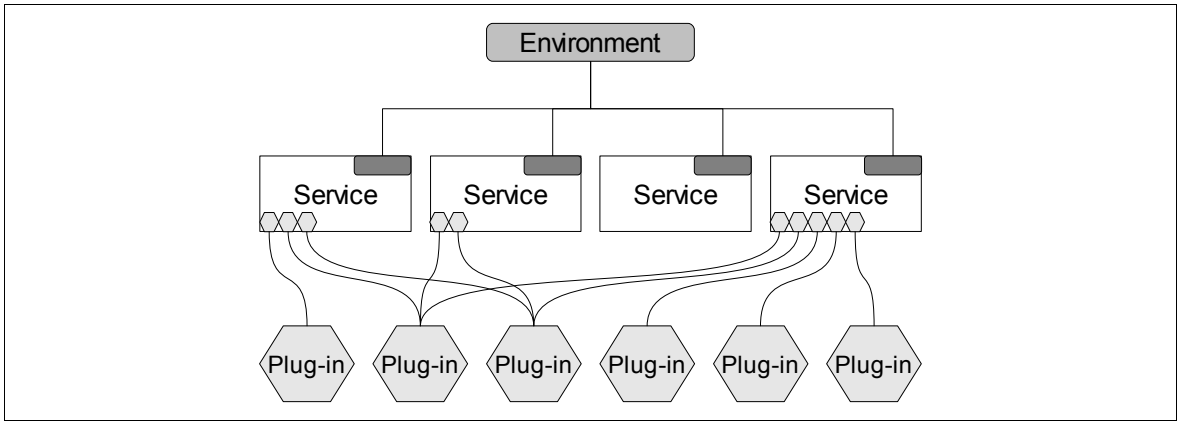

Figure 3-2: *Schema of SeisHub's modular architecture.*

SeisHub itself is written in Python[16] [Van Rossum et al. 1990], an object-oriented, open source, high level programming language available for all major operating systems. Among others, Python is known for its remarkable simple and readable syntax, partly because of its unique indentation style – using white spaces as block delimiter - and its large, comprehensive standard library providing tools for all kind of tasks, commonly referred to as “batteries included” [Lutz 2006, p. 91; Martelli et al. 2005, p. 526]. It has been chosen by the author as the preferred programming language after excellent experiences with the popular web-based Python products Trac[17], Zope[18] and Plone[19]. SeisHub also relies on a few additional Python modules, nevertheless the number of external dependencies has been kept as low as possible. Most external modules will be introduced in detail throughout the next sections.

16 http://www.python.org
17 http://trac.edgewall.org
18 http://www.zope.org
19 http://www.plone.org

For better comprehension of the whole chapter, it may be beneficial to have a SeisHub server instance installed and running by now using the SeisHub Installation Guide in Appendix A.1. Web addresses (URL) in the following images and examples referring to the default host “localhost” and port “8080” must be adapted to the particular setup, if modified. The running server can be tested by calling SeisHub's web-based management interface which is reachable with any standard browser via http://localhost:8080/manage/. The entry point of the RESTful XML resource repository can be found under http://localhost:8080/xml/[20].

Additionally, it should be noted that actually two separate SeisHub processes are spawned using the start-up script in `trunk/bin`. The main process (using process file `trunk/seishub/seishub.tac`) includes all services and extensions described throughout this chapter. The secondary process (using `trunk/seishub/seedmon.tac`) is limited to the waveform archive crawling service by disabling all services except SEEDFileMonitor (see chapter 3.5.4). This split into two processes allows better resource usage for multiple processors. It also enables SeisHub to have two file crawlers running in parallel: one is optimized for processing recent files more often, the other one for walking the whole data archive over time. The actual crawling process and the associated SEEDFileMonitor service will be covered in section 3.3.2, and more specifically in section 3.5.

The next three subchapters give a short introduction to core elements of SeisHub: environment, services and components in general. Further sections within this chapter look into a few specific components such as packages, resource types, and mappers.

3.3.1 Environment

The environment class[21] `trunk/seishub/env.py` is initialized exactly once at the start-up of a SeisHub server and shared with every single service and extension. A SeisHub environment consists of a configuration handler `env.config`, a logging system `env.log`, a relational database manager `env.db`, a XML catalog handler `env.catalog`, a package handler `env.registry`, a component manager `env.compmgr`, a resource tree `env.tree`, and an user management handler `env.auth`.

`env.config`. The configuration handler parses and manages the central configuration file `trunk/conf/seishub.ini`. Most options which are changeable via the administrative interface will be stored into this file. Services and plug-ins may define their own options providing default values, e.g. the HTTP service needs a port which defaults to 8080 if no port is defined. Unset

20 For better browsing experience the format option may be used: http://localhost:8080/xml?format=xhtml.
21 Internally named as “the class that rules them all”.

options with default values will automatically be created at server startup. There is no initial configuration file delivered with SeisHub, instead, the configuration file will be created with all default values when the server is started the first time. A full list of all possible configuration options currently available is given in Appendix A.2.

`env.log`. The logging handler is used to collect and primary filter log messages using the log level defined in SeisHub's configuration file.

`env.db`. This is the central database manager connecting SeisHub with the relational database back-end. The manager itself is sported by SQLAlchemy[22], a Python SQL toolkit and Object Relational Mapper (ORM) supporting many major databases, such as SQLite, PostgreSQL, MySQL, Oracle, MS-SQL, Firebird, MaxDB, MS Access, Sybase, Informix, and DB2. The database manager is able to access SeisHub's resources via classical SQL queries or by using the more "pythonic"[23] way of ORM – here, tables and relations are mapped into Python objects. The default database is set within the `[db]` section of SeisHub's configuration file. If no database is preset, SQLite will be used. SQLite is a file-based, transactional SQL database engine, which is a standard module of Python version 2.6 and above. However, it should be replaced by PostgreSQL in a productive server environment, because SQLite has a low level of concurrency allowing only a single process to write into the database at the same time, which results into a performance loss. But it is the perfect choice for evaluating SeisHub as it prevents installation of an extra database system, which can be a quite taxing procedure.

`env.catalog`. The catalog handler gives an object-orientated way of managing SeisHub's XML resources by offering various object-orientated helper methods.

`env.registry` and **`env.compmgr`.** The component manager detects and initializes SeisHub components/plug-ins and passes new or modified components to the registry class. The registry takes care of synchronizing all component options with SeisHub's internal database.

`env.tree`. The resource tree is a hierarchical representation of activated components such as packages, resource types, and mappers. Essentially the same tree structure is used within the HTTP and the SFTP services. The tree structure is automatically extended (or reduced) by activating (or deactivating) plug-ins. Building up the resource tree only once at startup or on explicit request increases the overall performance dramatically. The actual resource tree is visualized in the component browser panel at http://localhost:8080/manage/components/browse-components.

22 http://www.sqlalchemy.org
23 The term "pythonic" is a rather vague concept. Faassen 2005 suggests to start up a Python interpreter and type "`import this`" to see one possible perspective.

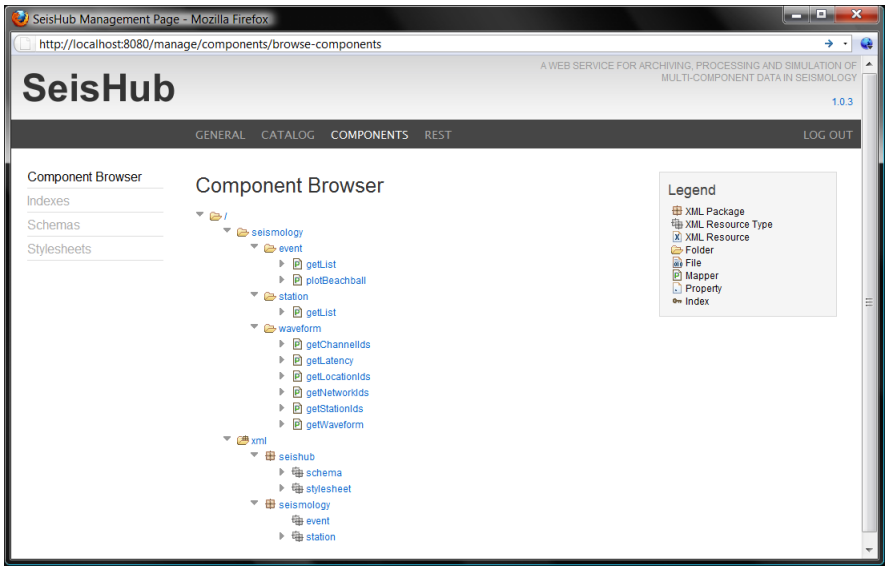

Figure 3-3: *SeisHub's component browser.*

`env.auth`. SeisHub includes a basic user management in order to protect the usage of services and resources from untrusted sources. User data and authentication information are stored in a separated SQLite database situated at `trunk/db/auth.db`, which will automatically be created if no such file exists. The default login name and password are “admin” (without the quotes) and should be changed immediately for any productive system. The `env.auth` handler gives a helper interface to validate authentication requests or manipulate existing user information. The actual authentication routines are implemented within the service modules using protocol specific authentication standards, e.g. Basic Authentication [RFC 2617] for the HTTP protocol and the key-based Secure Shell Authentication [RFC 4252] for SSH and SFTP. A web-based interface for the user management is given at http://localhost:8080/manage/admin/permission-users/.

3.3.2 Services

Services are core classes to handle all network or time-based events within SeisHub such as a basic resource requests from a HTTP client or a periodically call of a function. The current implementation of all services can be found at the `trunk/seishub/services/` directory. They are syntactically complex classes tightly coupled with the Twisted[24] networking library, an open source framework for building network applications in Python. At its core, Twisted is an asynchronous and event-based networking framework. Both techniques optimize the usage of available resources and effectively reduce the need of threads [Fettig 2005, p. xiii]. Python has an excellent and remarkable easy support for threads, but its implementation is known as not truly

24 http://twistedmatrix.com

thread safe – only one single thread is ever executed at any time managed through a single shared lock, called the global interpreter lock (GIL) [Lutz 2006, pp. 183-185]. Therefore, highly threaded Python applications suffer performance issues compared to asynchronous single-thread programming as Twisted provides. A comprehensive insight into this programming art is given by the Twisted core documentation [Twisted 2008].

Despite their complexity, services incorporate the component system stated above, which significantly simplify the creation of extensions. Plug-in developers do not have to know to much about the underlying network protocol and its implementation details anymore. Instead they write a new plug-in by inheriting from a single base class, choosing the interface classes of services they want to extend and providing only the desired functionality with a few lines of source code. The components system and the generation of extensions will be discussed in more detail in the next section. However, for the sake of completeness, service relevant interface classes are listed within this section

The following services are currently available in SeisHub: HTTP/HTTPS, SFTP, SSH, SEEDFileMonitor, and Manhole.

HTTP/HTTPS. This is the main service of SeisHub implementing the network protocols HTTP (default port 8080) and its secure variant HTTPS (default port 8443) deployed by the RESTful Web service and the web-based management interface. This core component of SeisHub should not be disabled.

The management interface allows easy configuration of SeisHub via a standard web browser. New management web pages can be added by writing extensions using one of the following Interfaces: `IAdminPanel`, `IAdminStaticContent`, or `IAdminTheme`. Examples for all current panels can be found in the `trunk/seishub/packages/admin/web` directory.

The RESTful Web service contains all elements of the resource tree provided to the clients in a XML representation. Both the Web service and the representation format are discussed in chapter 3.4. The resource tree itself can be extended by creating new plug-ins implementing the `IPackage`, `IResourceType`, or `IMapper` interface classes. All three interfaces will be covered in detail within chapters 3.4.3 and 3.4.4.

Further, it is possible to add whole disk-based directories into the resource tree by using the `[fs]` section in SeisHub's configuration file (see Appendix A.2). Those directories may additionally contain specific Python scripts with the file extension ".rpy", which are directly executed by a request returning any results to the RESTful Web service. This feature is similar to CGI programs commonly used in standard web servers. An example script can be found at `trunk/seishub/processor/tests/data/filesystem/scripts/test.rpy`.

Any client activity using the HTTP or HTTPS protocol is logged into separate access log files which can be found in the log directory `trunk/logs`.

SFTP. A convenient way to access SeisHub's resources tree is the integrated SFTP server. The same hierarchical structure reachable via the RESTful Web service is hereby mapped into a virtual file system. Plug-ins extending the resource tree (as described in the paragraph above) will also be applied to the virtual file system. The SFTP service is yet an experimental service, as every client tested by the author so far slightly differs from one other and a consistent implementation is therefore difficult to achieve. However, it massively simplifies uploading, downloading, and managing of XML resources, which is especially true for file operations on multiple documents. It should be noted that accessing very large XML repositories (single directories with a huge amount of resources) will eventually result into a timeout error, depending on the client settings. This service should be disabled in a productive environment because of its experimental nature.

SSH. The SSH service (default port 5001) is an experimental secure interface to a terminal based management console. It does not offer access to the actual XML resources like the SFTP service above. Instead, it implements a set of commands for the administration of the SeisHub server. Typing “help” into the command line lists all currently available key words. As this service is experimental and not considered as integral part of SeisHub, the current number of commands is limited. New commands can be introduced by writing a plug-in using the `ISSHCommand` interface class as demonstrated in `trunk/seishub/packages/admin/ssh/general.py`.

SEEDFileMonitor. The SEEDFileMonitor service periodically crawls any given directory for MiniSEED files and synchronizes them with SeisHub's internal database. The crawling process itself and the associated database structure will be introduced in chapter 3.5 and more specifically in section 3.5.4. As this service is a purely seismological extension for SeisHub, it may be disabled safely in any non-seismological environment.

Manhole. This service provides remote access to an interactive, administrative Python Shell through SSH (default port 5002). It allows the user to introspect and manipulate Python objects inside of a running SeisHub instance. The Manhole service is the perfect debugging environment, and, as the name already implies, it should never be activated in a productive mode.

SFTP, SSH and Manhole services are using key-based authentication protocols. SeisHub will automatically generate an unique set of public and private keys for each service at the initial start-up if no keys exist. The same procedure applies for the web server certificate of the HTTPS service. All created keys and certificates can be found within the `trunk/conf/` directory. It should

be noted that modern browsers explicitly warn about and even block pages using self-signed[25] HTTPS certificates, which is actually the correct behavior to ensure secure data transaction in the Internet. For a public, productive, multi-user environment it may become handy to provide a valid HTTPS certificate signed by a trusted certificate authority (CA). However, most browsers also offer the possibility to permanently accept such self-signed certificates.

Each service has its own unique configuration section within SeisHub's initialization file `trunk/conf/seishub.ini`. Possible setup options are summarized in Appendix A.2.

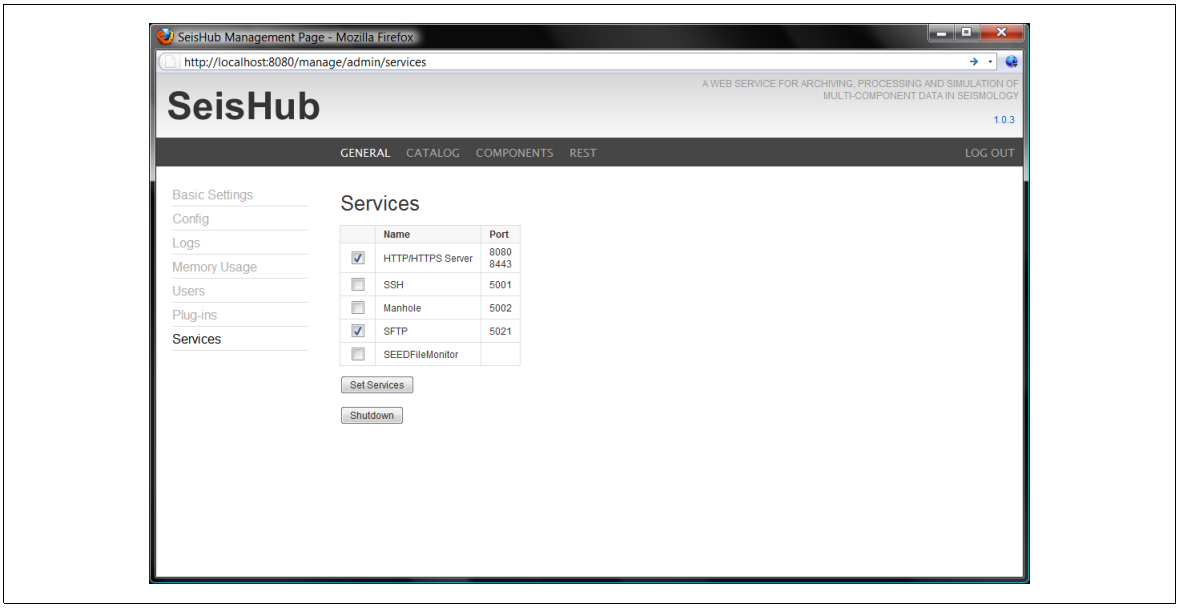

Figure 3-4: *Web-based administration interface for managing SeisHub services.*

Services and therefore linked plug-ins can be activated or deactivated at any time during the runtime of a server. Please note that by deactivating a service, it will not terminate immediately as long as a single client is still connected. Instead it will shut down after a certain grace period. Figure 3-4 shows the corresponding web-based service administration interface, which is reachable by calling http://localhost:8080/manage/admin/services in a browser.

3.3.3 Components/Plug-ins

Components are Python classes extending one or multiple services with new functionalities. Extension points are declared in interface classes. An example of an interface declaration for the SSH service can be seen in listing 3-1 taken from `trunk/seishub/packages/interfaces.py`. An interface class is basically a template or documentation of expected methods and attributes for a plug-in extending SeisHub. Any plug-in implementing an interface class promises to provide all specified methods and attributes.

25 A certificate which is signed by its own creator.

```
from seishub.core import Interface, Attribute

class ISSHCommand(Interface):
    """
    Interface for adding commands to the SSH service.
    """

    command_id = Attribute("""
        The SSH command.
        """)

    def executeCommand(request, args):
        """
        Processes a command line.

        Request object and a list of arguments are given.
        """
```

Listing 3-1: *Interface declaration `ISSHCommand` of the SSH service.*

Listing 3-2 demonstrates how a new SSH command "`Demo`" is implemented in SeisHub using the interface description above. Please note that a component developer does not need to know about the underlying SSH protocol. Instead he has to provide the command keyword and basic functionality within the method "`executeCommand`" - here simply writing some messages back to the SSH client.

```
from seishub.core import Component, implements
from seishub.packages.interfaces import ISSHCommand

class MyDemoCommand(Component):
    implements(ISSHCommand)

    command_id = "Demo"

    def executeCommand(self, request, args):
        msg = "You just called DEMO with %d arguments" % len(args)
        request.writeln(msg)
        request.writeln("Try again!")
```

Listing 3-2: *Example plug-in extending the SSH service with the keyword "`Demo`".*

A plug-in may also implement multiple interfaces within the same class by providing methods and attributes of all used interface declarations. Components can be enabled or disabled during run-time using the web-based administration page at http://localhost:8080/manage/admin/plug-ins as seen in figure 3-5.

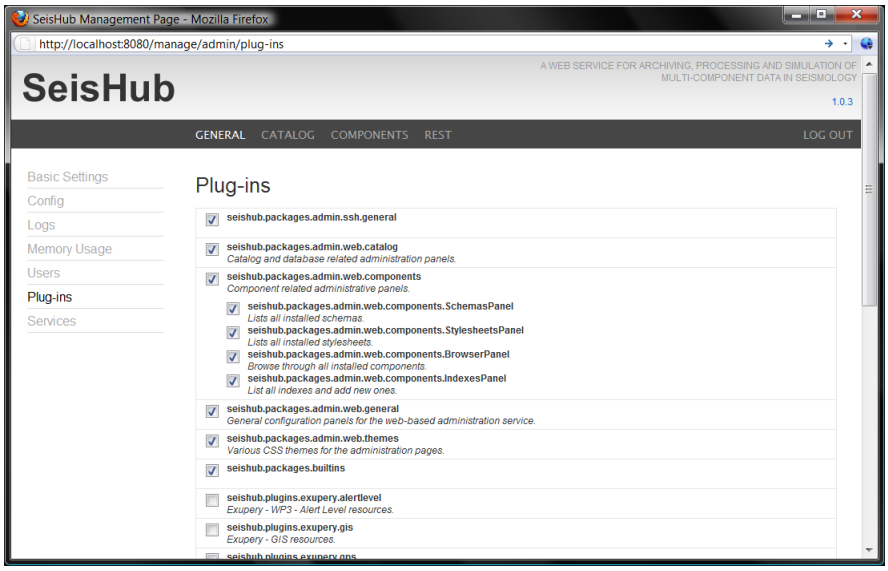

Figure 3-5: *Web-based administration interface for SeisHub plug-ins.*

All possible extension points for components are briefly summarized in the next table. Please refer to `trunk/seishub/packages/interfaces.py` for a full description of each interface. Examples using those interfaces can be found within the `trunk/seishub/packages` directory.

Service(s)	**Interface**	**Description**
HTTP/HTTPS	`IAdminPanel`	Extends the web-based administration interface with a new panel.
	`IAdminTheme`	Adds a new CSS-based layout for the management pages.
	`IAdminStaticContent`	Adds static content, e.g. images.
SSH	`ISSHCommand`	Adds a new SSH command.
HTTP/HTTPS SFTP (RESTful Web service)	`IPackage`	Defines a new package for the XML repository.
	`IResourceType`	Defines a new resource type for the XML repository.
	`IMapper`	Defines a procedure addressable on a fixed URI within the resource tree.
All	`ISQLView`	Creates a fixed SQL View within the relational database back-end.
All	`IProcessorIndex`	Creates a programmable index type.

Table 3-1: *SeisHub interfaces implementable by plug-ins.*

Another feature of the implemented component architecture is the integration of the Python packaging management systems "distutils" and "setuptools" within SeisHub. Distutils[26] is the standard way to build and install Python packages within a Python instance. Setuptools[27] is a powerful package distribution library which automates the find, download, install, and upgrade process of packages and dependencies by using a simple command line management tool called "`easy_install`". The latter accepts either plain URLs (supported protocols are HTTP, FTP, SVN) pointing to Python packages or it searches those in the central Python Package Index[28] (PyPI), a web-based package repository similar to CPAN[29] for Perl. Packages are usually shared in a single-file directly importable distribution format, called Python Eggs (file extension ".egg"). A Python Egg is created for a specific target platform and can directly be used without the need for any additional compilers. Another distribution format are plain packed Python source files.

SeisHub automatically searches at start-up for any modules in the Python system path and the `trunk/plugins` directory for Python modules implementing any interfaces from SeisHub. If such a module is found, it will automatically be available within SeisHub and can be activated via the web-based management interface. A plug-in developer can distribute extensions for SeisHub using PyPI. Moreover, any SeisHub operator can easily search for new or update existing plug-ins by using the tool "`easy_install`".

26 http://docs.python.org/library/distutils.html
27 http://pypi.python.org/pypi/setuptools/
28 http://pypi.python.org/pypi/
29 http://www.cpan.org

3.4 XML Database & RESTful Web Service

This chapter concentrates on the XML database component of SeisHub explaining the concept of resources and the associated indexing process of such XML documents. Additional sections introduce the three main components packages, resource types, and mappers. All elements will be explained further by using examples applied via the RESTful Web service interface.

3.4.1 Resources

As stated earlier, the fundamental storage units (for all non waveform data) are valid XML documents (also called resources). In order to create or modify data in SeisHub, one has to upload a whole XML document to the server using a supported network protocol. Each stored resource has got an unique Uniform Resource Locator (URL), which consists of the root of the XML repository (server host, port, and the fixed folder name "xml", e.g. http://localhost:8080/xml/), a resource specific package, resource type, and resource name. The resource name is either set by the uploading application or automatically generated by SeisHub. Package and resource type are both explained in the next chapter, but can for now be imagined as sub-folders used to further structure the storage of XML resources. As an example, one of SeisHub's build-in transformations style sheets (XSLT document) named "seishub_stylesheet_resourcelist_xhtml.xslt" can be found in the package "seishub" within the resource type "stylesheet". Altogether this resolves into the URL http://localhost:8080/xml/seishub/stylesheet/seishub_stylesheet_resourcelist_xhtml.xslt.

SeisHub's RESTful Web service allows creating (HTTP POST), modifying (HTTP PUT), downloading (HTTP GET), and deleting (HTTP DELETE) of XML resources (see table 2-3, chapter 2.2.2). The following paragraphs will briefly demonstrate each method on a XML-based seismic station resource "dataless.seed.BW_MANZ.xml" using the program cURL, a platform independent command line tool for transferring files with URL syntax [Stenberg et al. 2009]. Please make sure both resource type "station" and package "seismology" are enabled within the web-based management screen in order to reproduce the following examples.

```
curl -v --data-binary @dataless.seed.BW_MANZ.xml -u admin:admin
     -X POST http://localhost:8080/xml/seismology/station/MANZ
```

Listing 3-3: *Creating a named seismic station XML resource.*

The command above uploads the local file "dataless.seed.BW_MANZ.xml" into the resource type "station" within the package "seismology", and under the resource name "MANZ". The option "-u" is used for submitting a user name ("admin") and password ("admin"), and the parameter "-v"

generates a more verbose output. Calling the program once should upload the XML document under the stated URL and returns a HTTP status code 204 (OK).

Please note how the server responses with a status code 403 (Forbidden) complaining that the resource already exists, if the same command above is called again. Also be aware that if the POST method is used without defining an actual name (as shown in listing 3-4), the resource name will automatically be generated and revealed in the "Location" header field of the response message.

```
curl -v --data-binary @dataless.seed.BW_MANZ.xml -u admin:admin
     -X POST http://localhost:8080/xml/seismology/station/
```

Listing 3-4: *Creating a unnamed seismic station XML resource.*

In order to modify a resource, the HTTP PUT method is used. Listing 3-5 shows how to update an existing resource with new data.

```
curl -v --data-binary @dataless.seed.BW_MANZ.xml -u admin:admin
     -X PUT http://localhost:8080/xml/seismology/station/MANZ
```

Listing 3-5: *Updating a seismic station XML resource.*

Fetching and deleting of resources using their URL is shown in listing 3-6 and 3-7. Of course one could also use any modern browser to view the created resource by calling the URL of the resource http://localhost:8080/xml/seismology/station/MANZ, or see the overview of all possible stations via http://localhost:8080/xml/seismology/station/. Viewing or removing a non-existing resource will result in a HTTP status code 404 (Not Found). Python examples interfacing with SeisHub's RESTful Web service can be found in the directory `trunk/contrib/restscripts`.

```
curl -v -u admin:admin
     -X GET http://localhost:8080/xml/seismology/station/MANZ
```

Listing 3-6: *Retrieving a seismic station XML resource ("`-X GET`" is actually optional).*

```
curl -v -u admin:admin
     -X DELETE http://localhost:8080/xml/seismology/station/MANZ
```

Listing 3-7: *Deleting a seismic station XML resource.*

Using only resource names to identify such XML resources is rather useless in any database system. Therefore, every resource creation or modification triggers the indexing system of SeisHub, which will be elaborated in the next section. However, resources should be labeled with feasible resource names in order to ease the handling, similar to file names in the file system.

3.4.2 Indexing

Indexing in computer science refers to the process of extracting and storing certain identifiers of a dataset for fast lookup of those values in order to retrieve the original dataset in a very time-efficient way. In a classical RDBMS certain table fields are marked as an index field, which is handled privilegedly within the database compared to other fields. SeisHub, as a XML database, chooses a different approach: XML documents as the fundamental unit of storage, are saved unmodified in a single Binary Large Object data field accompanied by some meta data like file size, creation time, or the user identifier of the creator. XML documents are indexed by extracting predefined data elements using XPath expressions of the document and storing them separately from the resource itself. Additionally, the database can store values, which are gained by XPath specific processing functions like calculating the minimum, maximum, sum, or average of a number of values or counting certain elements within the document (see also chapter 2.1.4). All indexed values may later be used for data queries. The index process itself is immediately started after the first upload of a XML resource or every time a document is modified.

Meta information of any resource stored within SeisHub can be fetched by appending "/.meta" to the resource URL. Using "/.index" instead reveals all indexed values of this resource. Both requests are demonstrated using the example resource uploaded in the chapter before. The option "-o" followed by a file name stores the fetched document in the file system.

```
1  curl -v -u admin:admin -o meta.out
       http://localhost:8080/xml/seismology/station/MANZ/.meta
```

Listing 3-8: *Retrieving meta information of a seismic station XML resource.*

```
1  curl -v -u admin:admin -o index.out
       http://localhost:8080/xml/seismology/station/MANZ/.index
```

Listing 3-9: *Retrieving indexed values of a seismic station XML resource.*

Indexes are bound to a resource type either hard-coded in the resource type component itself or may be added or removed during runtime via the web interface. This flexibility allows adding of new index parameters at any time, but requires a reindexing of all resources of this resource type, which may need a while for big collections of resources. Indexes are always connected to a single data type. SeisHub basically stores indexed data in type-specific SQL tables in order to increase performance at database operations. For example, all indexed values of the datetime[30] data type are collected in a single SQL table called "default_datetime_index". Further supported data types are

30 Data type for a date and time combination.

boolean, text, float, date, and integer. Each index table contains mainly four columns: a document identifier pointing to the original XML document, the index identifier pointing to the actual index definition, the indexed value, and its document position allowing to distinguish between multiple entries within the same document.

Querying values using the database structure above is complicated for developers as indexes are distributed among multiple tables. Therefore, SeisHub creates for each single resource type an additional SQL View, basically a virtual table, containing all indexes of one resource type. Those tables are labeled as "/package/resourcetype" within the relational database and can be seen via the management interface calling http://localhost:8080/manage/catalog/db-query.

Each row of such a virtual table represents the cross product of all indexed values of a single XML resource. This works perfectly for values extracted only once within a document. However, any XPath expression found more than once within the same document will result in multiple rows. This procedure is imperfect for storing and indexing large series of values such as time series because this would result in huge SQL Views. This problem is not restricted to SeisHub – in fact it makes no sense at all to index each value of a time series within a database. However, in order to solve this problem, one could either index values which occur only once within the document or use the features of XPath to extract certain search parameters such as the minimum and maximum of a time series.

An issue which might occur are values bound to shared elements, e.g. data of multiple channels within a single station resource. SeisHub allows the definition of grouping elements (XPath expression of a parent element) for contained elements. All data found within such grouped elements will be represented in a single line in the SQL View table, e.g. for the example above, a single row per channel per station. Please be aware that grouping is a complex, time intensive process within the database and should be avoided if possible, especially grouping over multiple levels. The latter should better be separated into multiple resource types interconnected with the XLink technology.

One last restriction to the indexing process is linked to XML namespaces. SeisHub at the current state completely ignores such namespaces. Resource providers should try to avoid XML namespaces for elements which are going to be indexed. Validation schemas restricting documents of a resource type can further ensure that uploaded documents fulfill this requirement.

3.4.3 Package & Resource Types

Packages and resource types are components used to further structure XML resources into categories. Both components behave like virtual containers having its own unique URL comparable to directories on a file system. A package always consists of resource types, and the latter are used for the actual resources. The root of the whole XML repository containing all available packages can be seen via http://localhost:8080/xml/. A specific package is accessed by combining the repository root with the package name, e.g. http://localhost:8080/xml/seishub/ for the built-in package "seishub". The same applies for resource types and XML resources.

Packages and resource types are components and can be activated or deactivated on demand via the plug-in manager. By disabling a package, all connected resource types are disabled as well. Disabling a resource type prevents access to contained XML resources.

Next to their structural function, packages and resource types are also used to associate validation schemas (resource type only), transformation style sheets and indexes (resource type only; see chapter 3.4.2). Those may be defined directly into the actual component class or generated interactively via the web-based management interface. The latter should be used for testing purposes and be hard-coded afterwards into the plug-in class. Otherwise they will not be available for any new SeisHub instance using this component.

Validation schemas associated with a resource type are always applied on resource creation or modification requests within this resource type. SeisHub supports the following XML-based schema languages: XML Schema (XSD), RELAX NG, and Schematron (see chapter 2.1.5). Resource validation is started before the indexing process and includes all defined schemas. If a single schema validation fails, the request will be canceled and an appropriate error message is sent back to the client. One may upload arbitrary XML documents, if no validation schema is defined. XML schemas essentially restrict upload and storage of XML documents.

The example of uploading station resources above already used a XML Schema in order to validate a uploaded XML document. Uploading an invalid document, like shown in listing 3-10, will result in the HTTP status code 409 (Conflict) associated with a full error message why the validation process aborted the upload process.

```
1  curl -v --data-binary @index.out -u admin:admin
       -X POST http://localhost:8080/xml/seismology/station/NOXML
```

Listing 3-10: *Uploading a invalid seismic station XML resource.*

Transformation style sheets can either be applied before upload or more commonly at the retrieval of a resource. Style sheets are evoked by appending the "format" option combined with a short style sheet specific label to the request URL. It is also possible to build up a transformation chain by adding multiple format parameters to the resource URL. Beside the basic XML output format, all packages and resource types support the transformation into XHTML (format=xhtml) and JSON (format=json). The latter is used by the Ajax scripts within the web-based management interface. Unknown formats are ignored within SeisHub.

Fetching of the content of the "seismology" package in the JSON format is shown in listing 3-11.

```
curl -v -u admin:admin -o xslt-seismology.json
     http://localhost:8080/xml/seismology/?format=json
```

Listing 3-11: *Retrieving the content of the package "seismology" in the JSON format.*

The "station" resource type of the package "seismology" has an associated style sheet labeled as "map". This transforms a XML station resource into a XHTML document containing a table with a few station information such as latitude, longitude, elevation, channels, etc., and embeds a small map showing the location of the station. Listing 3-12 retrieves this representation for the uploaded station example used before and figure 3-6 shows the resulting file "xslt-map.html" viewed in a standard browser (or via http://localhost:8080/xml/seismology/station/MANZ?format=map).

```
curl -v -u admin:admin -o xslt-map.html
     http://localhost:8080/xml/seismology/station/MANZ?format=map
```

Listing 3-12: *Retrieving a seismic station XML resource in a simple XHTML format.*

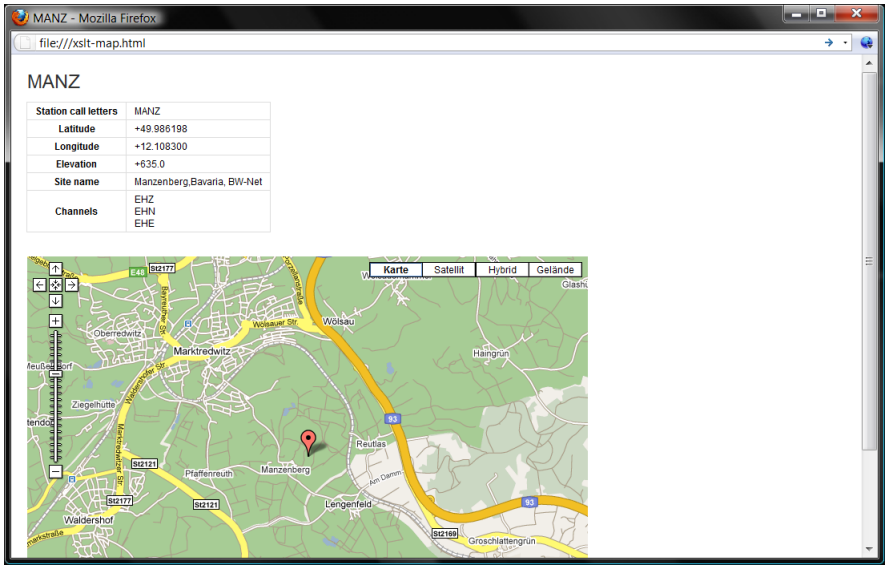

Figure 3-6: *Seismic station XML resource in a simple XHTML format.*

3.4.4 Mapper

Mappers are Python classes accessible via a user-defined, fixed URL and one of the four HTTP methods. Those classes may be used for writing a full featured RESTful resource interface or more commonly for simply querying the internal database and returning a formated output (similar to a GETful approach). Mappers are important for a purely HTTP-based access of data within SeisHub, where a direct SQL connection to the relational database back-end is not possible or not wanted. Generally, direct access of the back-end should be prevented for security and stability reasons. Instead, well defined “questions” or database queries, and the expected output format are to be defined beforehand and implemented within a mapper class. Mappers are components, which again can be enabled or disabled during runtime.

A simple GETful mapper situated at http://localhost:8080/seismology/station/getList is used to demonstrate the concepts of a mapper. By calling the URL above in a browser, it is possible to retrieve a XML document containing indexed information of all station resources. However, the mapper class evaluates additional parameters given to the URL in order to further filter this output list, e.g. “network_id”, “station_id”, “start_datetime”, “end_datetime” etc., or to generate a specific output format, here allowing XML (default), XHTML (format=xhtml) or JSON (format=json). In order to fetch all station resources within the seismic network BayernNetz formated in the JSON format, append “?network_id=BW&format=json” to the URL above.

Please note that the implementation of the actual database query and formating of the output is up to the developer of the mapper. He may rely on the catalog functions of the central environment object (see 3.3.1) or call SQL directly on the database back-end to retrieve data. However, he is able to request data from other existing web services or mappers. The same is true for the output of data: he may rely on internal style sheets registered within SeisHub or external style sheets reachable via an URL in order to transform the result.

However, this freedom comes with a price. Mapper developers have to make sure that all incoming parameters are securely mapped to database queries avoiding huge querying times or even worse SQL injections, an exploiting technique to pass SQL commands over the URL into the database query. SeisHub operators should always check mapper classes from unknown sources before using them in their own server instance.

3.5 Waveform Database

Another main component of SeisHub is the waveform database, which automatically observes any given directory and its subdirectories for seismological waveform data in the MiniSEED or full SEED standard. Although there are various file formats in seismology, SEED has been chosen as the default format for time series within SeisHub. Several reasons for this decision will be given by introducing the SEED standard within the following section. This chapter will also discuss the usage of SEED for synthetic seismograms and will give a preview of recent developments of real-time waveform distribution on a European level. The chapter concludes with technical implementation details of the waveform database.

3.5.1 Standard for the Exchange of Earthquake Data (SEED)

SEED is an standard format for long term archiving and data exchange of digital seismological data introduced by the FDSN[31] (International Federation of Digital Seismograph Networks). The full SEED format and its two subsets: Data Only SEED (MiniSEED) and Dataless SEED formats are elaborated in full complexity in the SEED Reference Manual [IRIS 2009a]. Dataless SEED contains solely headers and meta information of time series, e.g. instrument responses. MiniSEED, as the complement of Dataless SEED, includes the actual digital waveform data enclosed by a minimal set of header information necessary to process this data. Both can essentially be combined to or split from a full SEED volume by applying the recommendations in Appendix G of the reference manual.

SEED volumes are divided into one or more contiguous blocks of fixed length, which are called logical records. Typical sizes are 512 bytes for real-time data minimizing processing latency and 4096 bytes for archiving purposes to reduce the file sizes. Each record starts with a record sequence number followed by one or multiple blockettes, data structures consisting of an identifier, size, and a sequence of data fields specific to this blockette. The actual digital time series data is distributed in a compressed form in a specific blockette (identifier 1000) complemented by other blockettes containing additional meta data like start and end time, sampling rate, quality information, etc. This basic layout allows efficient usage of SEED volumes for sequential media such as magnetic tapes, and random access media like magnetic and optical disks. It also effectively reduces the processing time for data distribution of MiniSEED volumes, as cutting and merging of data can be done at record basis without unpacking the contained data.

31 http://www.fdsn.org

MiniSEED has been widely used as the default exchange format for continuous waveform data by many major seismological organizations and Data Management Centers (DMC), e.g. IRIS[32] (Incorporated Research Institutions for Seismology), GFZ[33] (Geoforschungszentrum Potsdam), and ORFEUS[34] (Observatories and Research Facilities for European Seismology) [Ahern 2000; Shapira 2007; Eck & Sleeman 2008, p. 3; GFZ 2008]. Many analyzing tools used by seismologists are able to handle MiniSEED data, e.g. Seismic Handler[35] [Stammler 1993; Stammler & Walther 2009], SEISAN [Havskov & Ottemöller 2000], or SeisGram2K[36] [Lomax 1991]. Moreover, MiniSEED is directly used within SEEDLink, a robust TCP/IP based transfer protocol introduced in the real-time data acquisition and processing system SeisComP (Seismological Communication Processor) [IRIS 2009d; Hanka et al. 2000; Heinloo 2001]. SEEDLink is also utilized via plug-ins in other major real-time data management and processing systems as in Earthworm [Johnson et al. 1995] or Antelope [BRTT 2009]. A few commercial available data loggers are even recording directly into the MiniSEED format [Havskov et al. 2007].

SEED supports several other earthquake related data streams next to the common classical records of seismometers such as tilt meter and rotational sensors. However, SEED is also perfectly suitable for other geophysical observations based on time series, such as weather or environmental sensors, gravimetric or magnetic field data. Different data streams are distinguishable from each other by using standardized network, station, location, and channel identifier within the SEED blockettes, which will be discussed next.

Network code. The network code consists of two alphanumeric characters identifying the network operator, e.g. “BW” for the Bavarian Seismological Network (see chapter 4.2). All official seismic network codes are assigned by the FDSN archive (IRIS DMC) to ensure uniqueness to seismological data streams [IRIS 2009a, Appendix J; IRIS 2009b].

Station code. The station identifier may consist of a maximum of five letters or numbers (without spaces in between) referring to a station name in a seismic network, like “MANZ” for station Manzenberg of the BayernNetz. Station names should be unique within their network. Additionally, most seismic broadband station names are often registered with the International Registry of Seismic Stations[37] (IR) to ensure a global unique identifier of this station [Shapira 2007]. This practice became questionable with the rapid increasing number of stations within the last decade and has been faced by a new proposal from the IASPEI[38] (International Association of Seismology and Physics of Earth Interior) working group on station codes [IASPEI 2008].

32 http://www.iris.washington.edu
33 http://www.gfz-potsdam.de
34 http://www.orfeus-eu.org
35 http://www.seismic-handler.org
36 http://alomax.free.fr/seisgram/SeisGram2K.html
37 http://www.isc.ac.uk/IR/index.html
38 http://www.iaspei.org

Traditionally, station identifiers are often associated with the geographical area where the station is situated [Shapira 2007].

Location name. A location code may contain two alphanumeric values to identify a single device connected to a station. Many SEED tools known to the author accept as location code either no value at all, or only integer values starting with "00".

Channel name. Channel naming is used to identify seismic and auxiliary data streams by applying exactly three alphanumeric characters. FSDN suggests for seismic data the following conventions for each letter in Appendix A of the SEED manual.

- The first letter specifies the sampling rate and response band of an instrument, e.g. "B" for broadband seismometers (sampling rate between 10 and 80 Hz, corner frequency >= 10 seconds), or "E" for an extremely short period device (sampling rates from 80 to 250 Hz, corner frequency < 10 seconds).
- The second letter identifies the family to which an instruments belongs, like "H" for a high gain seismometer or "G" for a gravimeter.
- The last letter indicates the physical orientation of the channel, such as "Z", "N", or "E" for the vertical, north-south, or east-west components of traditional seismic devices.

Please note that FDSN usage conventions are only a subset of the actual SEED definitions and are not required for a valid SEED volume. However, it is recommended to stay within this conventions for data exchange of seismic waveform data [IRIS 2009a, p. 204].

A sequence of network, station, location, and channel identifier as introduced above, combined with the year and day of year are commonly used to label the actual SEED volumes in the file system. Archived MiniSEED data within the BayernNetz follow the SeisComP Data Structure (SDS) definition introduced with SeisComP and SEEDLink [GFZ 2009]. Other combinations can be seen in file-based MiniSEED archives directly accessible via the Internet, e.g. ORFEUS waveform archive at ftp://www.orfeus-eu.org/pub/data/continuous/.

3.5.2 Synthetic waveform data

One task of this study was to enable the joint storage and combined access of observational and synthetic waveform data within the same database. Although SEED is able to store any arbitrary 1D time series with a fixed sampling rate, there is no official FSDN convention or recommendation for synthetic seismograms. In fact, the current edition (January 2009) of the SEED reference manual explicitly states that SEED was not designed "[…] for the exchange of processed (e.g., filtered) data or synthetic data (e.g., created by computer modeling). While such use is possible, we will not support it." [IRIS 2009a, p. 8].

Extending SEED with additional fields for synthetic waveforms is no option as this will cease the level of interoperability with standard SEED tools. However, the author strongly recommends that the location field is used to solve this shortcoming. Location identifiers are generally rarely in use, but fully compatible with the existing standard and derived tools. Computationally generated seismograms can essentially be seen as data received from an additional (virtual) device which would even match the original SEED specification. Depending purely on the location identifier for synthetics would also cover the FDSN channel naming convention, basically allowing the storage of the synthetic output of a simulation run under different sampling rates and instrument orientations. Within the BayernNetz, a location identifier code of equal and more than "90" is suggested by the author, reserving nine more slots for further simulation algorithms or "virtual instruments".

It should be noted that the remarks and suggestions above only apply to the actual physical storage and distribution of computational generated seismograms in the MiniSEED format. The handling of synthetics within a database system such as SeisHub is a completely different story. Querying synthetic waveform data distinguishable from standard observational data requires either the exact knowledge of the location code or must be handled by an additional database structure. Preferably, this structure also offers further fields for additional information such as used program code and models, parameters, responsible persons, etc. For SeisHub this is essentially done by simply adding new fields into the XML station resources (see chapter 4.2). Also, please be aware that every new network, station, and location combination requires its own new XML station resource. Providing synthetics at geographical positions not actually covered by an existing station also requires the generation of a new (virtual) station name and associated station resource. There is no official naming convention for such virtual stations.

3.5.3 ArcLink

ArcLink is a distributed data request protocol usable to access archived waveform data in the MiniSEED or SEED format and associated meta information as Dataless SEED files. It has been originally founded within the German WebDC initiative of GEOFON (Geoforschungsnetz) and BGR[39] (Bundesanstalt für Geowissenschaften und Rohstoffe) [Hanka & Kind 1994; WebDC 2009]. ArcLink has been designed as a "straight consequent continuation" [Hanka & Saul 2008, p. 159] of the NetDC concept [Casey & Ahern 1999] originally developed by the IRIS DMC. Instead of requiring waveform data via E-mail or FTP requests, ArcLink offers a direct TCP/IP communication approach. A prototypic web-based request tool is available via the WebDC homepage at http://www.webdc.eu.

Recent development efforts within the NERIES[40] (Network of Excellence of Research and Infrastructures for European Seismology) project focuses on extending the ArcLink network to all major seismic data centers within Europe [WebDC 2009] in order to create an European Integrated Data Center (EIDAC). Currently (September 2009) there are four European data centers contributing to this network: ORFEUS, GFZ, INGV (Istituto Nazionale di Geofisica e Vulcanologia), and IPGP (Institut de Physique du Globe de Paris). Also, the IRIS data center has already expressed its interest to implement ArcLink in order to allow unified data access on a global scale [ORFEUS 2009]. Additionally, the Seismological Observatory Fürstenfeldbruck, Department of Earth and Environmental Sciences, Geophysics, LMU Munich has been extending this network since 2006 by a passive ArcLink node serving waveform and meta data from the BayernNetz. Please note that a working group within the NERIES project recently introduced a SOAP-based Web service "Seismolink[41]" which offers alternative access to the ArcLink infrastructure [Spinuso 2009]. Providing ArcLink via a Web service using standard HTTP as transport protocol eliminates firewall problems which might occur with ArcLink's default port (18001).

SeisHub supports the ArcLink protocol in order to handle seismic waveform requests not available within the local waveform archive (see next section). Such requests are rerouted to an ArcLink server and the results are used to update SeisHub's internal waveform database and fulfill the original client request. Interacting with the ArcLink server is done via the newly developed `obspy.arclink` module of the ObsPy library further discussed in chapter 3.6. Access to ArcLink via SeisHub is given with the waveform mapper shown in section 3.5.5.

39 http://www.szgrf.bgr.de
40 http://www.neries-eu.org
41 WSDL document http://orfeustest.knmi.nl/wsdl/seismolink/seismolink.wsdl (see chapter Fehler: Referenz nicht gefunden)

3.5.4 SEEDFileMonitor

SEEDFileMonitor is a service of SeisHub to automatically search file-based archives for SEED or MiniSEED volumes and to keep those files synchronized with the relational database back-end. The actual waveform directories among various other crawling options are specified within the `[seedfilemonitor]` section of SeisHub's configuration file (see Appendix A.2). Please note that SEEDFileMonitor does not need to know about the directory structure of the archive, as it will automatically crawl all included subdirectories of the specified paths. To increase performance, the search process can be restricted in the configuration to certain files using an Unix-style file name pattern, e.g. "`BW.EH?.*.mseed`" for all files starting with "`BW.EH`", followed by a single character and a dot, and ending with the file extension "`.mseed`". Be aware that the filename of the SEED volume is not used to extract information about network, location, station, or channel codes. Instead, those identifiers are retrieved directly from the SEED volume. However, after indexing the SEED volume the filename is linked to this data.

Technically, the file crawler may be seen as two parallel running infinite loops[42]. The first loop keeps searching for new or modified files within the archive and stores the resulting file names into a list. The latter is used by a second loop responsible for processing and indexing the given files. Indexing is done using the `obspy.seed` submodule of the ObsPy library developed within this study and introduced in detail in chapter 3.6. However, the following data are extracted for each SEED file and directly stored as a single record in the database table "default_miniseed" of SeisHub's database back-end:

- absolute path and filename of the SEED volume,
- network, station, location, and channel identifiers,
- start and end time of the contained waveform data,
- total number of gaps (missing SEED data records) and overlaps (records with reoccurring time stamps) within the volume,
- total number of all data quality flags (set for each single data record) for detection of (1) station dependent amplifier saturation, (2) digitizer clipping, (3) spikes, (4) glitches, (5) missing or padded data, (6) telemetry synchronization error, (7) possible charging digital filter, or (8) questionable time tag [IRIS 2009a, pp. 98-99], and

42 SeisHub uses an event-driven approach as briefly discussed in chapter 3.3.2, therefore it is actually one single loop for all services within SeisHub, and methods are called periodically after a certain time interval. This may sound similar but results into a completely different implementation.

- statistics (minimum, maximum, average, median and upper and lower quantile) about data timing quality, a vendor specific value given in percentage taking clock quality and data flags into account [IRIS 2009a, p. 114]. High timing qualities mean generally good overall data quality. Timing quality is set for each single SEED data record and calculated over the whole volume if supplied by the data logger at all.

There are a few restrictions to the crawling and indexing process.

1. A single SEED volume needs to contain only one network, station, location, and channel code, otherwise the file will be ignored.
2. Each SEED file is treated in a single database entry pointing to the original file, regardless if the same network, station, location, channel, start- and end-time combination already exists. The uniqueness of waveform data within the database has to be assured within the file system. Please note that the SEEDFileMonitor is able to follow symbolic links (symlink) within the directory structure. Nevertheless, circular symbolic links within the crawling directories needs to be avoided altogether.
3. It was chosen to neglect time information for each gap and overlap into the database, as this would require a single database entry for each contiguous dataset.

The latter could be exploited by SEED volumes containing a huge number (n) of gaps by generating countless entries (n+1) in the database back-end. A typical MiniSEED volume covering a single channel for 24 hours and sampled at a frequency of 200 Hz can easily contain over 40,000 single SEED data records. This would result in a worst case scenario over 40,000 entries for just one single file.

Further, please be aware of the configuration flag "`keep_files`" (disabled by default) within the configuration section of the SEEDFileMonitor. Normally, SeisHub assumes that if a SEED volume is found as removed from the file system, it may also safely delete the associated database entry. This behavior works perfectly for small, decentralized waveform archives situated on the local hard drives of the SeisHub server. However, the capacity of hard drives is restricted. Moreover, common file archives on random access media are usually accessible via network-based data storage systems such as Network-attached Storage (NAS) or Storage Area Network (SAN) solutions. The author strongly recommends to enable the flag above for such environments. This guarantees that no database entry is automatically deleted if files are not reachable because of network problems. If really needed (for instance after manually deleting files from the archive), the flag can be set manually for a few minutes. The deletion/clean-up process will be finished within this time from the file crawler (depending on the archive size).

This leads to another point: indexing a waveform archive for the first time needs time. A single MiniSEED volume of approximately 20 MB file size uses up to 1.5 seconds for processing and indexing within the database. Depending on the size of the waveform archive, indexing the whole archive may need several hours or even days. Indexing will be resumed if SeisHub is stopped and restarted for any reason. The initial indexing could be further optimized by starting more processes focusing purely on the indexing task (see chapter 3.3). However, this is only recommended for computers with multiple cores.

At the current state, SeisHub only handles waveform files in the SEED or MiniSEED formats. From the technical point of view there is no reasons not to integrate other waveform formats like SAC [Goldstein 1998] or GSE2 [GSE 1997] (both are actually already supported within the ObsPy library). However, the following considerations must be taken into account for such extension.

- Streaming support for many other waveform formats is limited, which would result in significant longer processing times for cutting or merging of data of such formats. Here, the full digital waveform data has to be extracted and processed – a time consuming procedure compared to the handling of streams working on record basis.
- Both formats SAC and GSE2 in their current versions do not support network and location identifier which are required within SeisHub. Workarounds could be either defaulting to some preset values or extracting network and location names from the corresponding file name or from an extra configuration file.
- Typical SEED features like timing and data quality information would not be available within the database.

Generally it is a good idea to keep all files within the waveform archive in a single data format in order to reduce the common issues of format conversions. SEED seems the most obvious choice amongst all available waveform formats, partly because it is the standard format for archiving data at the Seismological Observatory Fürstenfeldbruck at the Department of Earth and Environmental Sciences, Geophysics, LMU Munich. However, as elaborated within the previous pages, it is also the default exchange and archiving format of many major international data centers and it is used in standard exchange protocols and services like SEEDLink and ArcLink. Staying with one format in SeisHub also further reduces the complexity of the already quite large (but powerful) software package. Nevertheless, the SeisHub client delivered with the ObsPy library offers the possibility of to store the requested waveform data into SAC or GSE2 format.

3.5.5 Waveform Mapper

The waveform mapper within the “seismology” plug-in of SeisHub unifies the request interface for locally archived seismograms and the retrieval of globally distributed waveform data via the ArcLink protocol. As mentioned in chapter 3.4.4, mappers are plug-ins for SeisHub extending the RESTful Web service. The waveform mapper at the current state offers only a HTTP GET method to the client, therefore seismograms can only be requested from SeisHub. Upload, modification or deletion of waveform data via a Web service is questionable and so far not supported.

The waveform mapper is bound to http://localhost:8080/seismology/waveform/getWaveform. After a HTTP GET request approaches the mapper, SeisHub checks the internal database if any data files are available for the required network, station, location, and channel combination during the requested time span. If files are found, they will be merged into a single MiniSEED volume and sent back to the requesting client. However, if no data could be found at all, the mapper will try to fetch the seismograms directly from the ArcLink network. Any waveform data resulting from such a request will be forwarded to the requesting client, but additionally stored by SeisHub in the `trunk/data/arclink` directory. Repeated requests to the ArcLink network can be avoided by allowing the waveform crawler service to index already downloaded data from the `arclink` directory.

The functionality of SeisHub's waveform mapper will be demonstrated using for now the purely URL-based Web service interface reflecting its web-based nature. As an example: retrieving waveform data for all available channels (“*”) of the seismic station Manzenberg (“MANZ”) of the BayernNetz network (“BW”) for the first 62.505 seconds of year 2008 would resolve into the following URL:

> http://localhost:8080/seismology/waveform/getWaveform?start_datetime=2008-01-01T00:00:00.000000Z&network_id=BW&station_id=MANZ&channel_id=*&end_datetime=2008-01-01T00:01:02.505000Z.

The server's response after calling this URL is a single MiniSEED file containing three seismic channels (“EHZ”, “EHE”, “EHN”), each with a sampling rate from 200 Hz and 12502 data samples. Essentially one can now play with the parameters “start_datetime”, “end_datetime”, “network_id”, “station_id”, “location_id”, and “channel_id” to retrieve locally available or remotely accessible waveform data. Please refer to the WebDC web front-end at http://www.webdc.eu/arclink/query for the complete list of possible network, station, and channel combinations within the ArcLink network.

Please note that in the URL string above, option “location_id” is not set. Missing parameters are usually automatically replaced with default values within the mapper, in this case all streams with an arbitrary location code are retrieved.

An alternative method requesting the same waveform data as above is shown in listing 3-13. It essentially shows a simple Python program using the `obspy.seishub` module of the ObsPy library. This module offers a basic Application programming interface (API) directly communicating with the waveform mapper. Please refer to chapter 3.6 for more information about this very convenient request method.

```
1  from obspy.core import UTCDateTime
2  from obspy.seishub.client import Client
3  c = Client("http://localhost:8080")
4  start = UTCDateTime(2008, 1, 1, 0, 0, 0, 0)
5  end = start + 62.505
6  st = c.waveform.getWaveform("BW", "MANZ", "", "*", start, end)
7  st.write("output.mseed", format="MSEED")
8  print st
```

Listing 3-13: *Retrieving waveform data using the ObsPy module* `obspy.seishub`.

Processing of seismograms is time and memory intensive, therefore the waveform mapper had to be restricted to ensure an acceptable overall performance of the SeisHub server. Please be aware of the following restrictions if working with the waveform mapper.

1. The maximal time span is limited to six hours – a larger time span request will automatically be truncated.
2. The mapper assumes that a SeisHub server is either responsible for a single station, or, if not, it fetches the related data via the ArcLink network. Routing of a request to an ArcLink server is therefore only started if no data at all can be found within the local waveform archive. This approach is problematic for queries using wildcards in any parameter. Theoretically, any such request must be redirected to an ArcLink server searching for possible new seismograms. However, this is no option because wildcards are often used in standard requests, and an additional ArcLink request would significantly increase the processing time (especially if for some reason the ArcLink server is not reachable). The author recommends not to use wildcards for retrieving seismograms which are not available via the local waveform archive.
3. Wildcards are not allowed for network and station codes.

3.6 ObsPy

ObsPy[43] is an open source, platform independent, modular software package for seismological observatories initiated by Moritz Beyreuther, Lion Krischer, and the author in 2008 at the Department of Earth and Environmental Sciences, Geophysics, LMU Munich. ObsPy is written in the Python programming language and offers a wide spectrum of sophisticated tools for working with seismic data such as waveform, inventory or event based data. A main feature is a consistent interface for reading, writing, processing, and plotting seismograms of different standards, such as SAC [Goldstein 1998], GSE2 [GSE 1997], MiniSEED and full SEED [IRIS 2009a].

One of its design goals was to keep the number of external dependencies as low as possible. For performance reasons, however, ObsPy directly depends on NumPy[44], a convenient, space-saving, and very fast array manipulation package for Python. Although most functionalities are implemented in Python, a few time critical subroutines, e.g. unpacking binary waveform data, are handled via external shared C libraries, e.g. in Linux/Unix named shared objects (*.so) and in Windows known as Dynamic Linked Libraries (DLL). It is possible to access these libraries via Python's native foreign function module ctypes[45] without writing additional external header files. As pre-compiled shared libraries can be distributed within Python packages, this approach even eliminates the need for an installed compiler and other build tools on the target system. This is especially important for the Windows operating system, as compiler tools are usually not part of a standard installation, but instead must be downloaded and installed manually.

Many submodules were directly influenced by the development of SeisHub and most of the ObsPy's core classes are now standard dependencies of SeisHub. The following modules are mainly created for or are required dependencies of SeisHub: `obspy.core`, `obspy.mseed`, `obspy.imaging`, `obspy.arclink`, `obspy.xseed`, and `obspy.seishub`. Further modules of ObsPy are `obspy.gse2`, `obspy.signal`, `obspy.sac` and `obspy.wav`, which will not be covered in detail within this study.

Module `obspy.core`. The core module is required by all other modules of ObsPy. It provides a convenient class for handling and manipulating UTC-based dates and times by extending Python's DateTime object. The module also offers standardized interfaces and methods for waveform (streams and traces), events, and inventory data.

43 http://www.obspy.org
44 http://numpy.scipy.org
45 http://docs.python.org/library/ctypes.html

Module `obspy.mseed`. This module is primary a Python wrapper around libmseed[46], a fast C library framework for manipulating and managing SEED volumes (see chapter 3.5.1) written and maintained by Chat Trabant (IRIS). Additionally, the module offers several sophisticated functions to retrieve quality information from SEED files such as the number of gaps and overlaps or statistics about data quality and timing issues provided.

The `obspy.mseed` module is a required dependency for the file crawler of the SEED waveform database extension of SeisHub. The performance of reading and processing of waveform data is of utmost importance as the file crawler constantly synchronizes incoming real time waveform data with the SeisHub database.

The following basic Python script reads a MiniSEED file named "`test.mseed`" from hard disk into a stream object containing all available traces including header information and the unpacked data samples. The function `obspy.read()` is provided by the `obspy.core` module introduced above and may be used to import any waveform file supported by ObsPy. The actual file format is either given as an additional parameter or will be auto-detected by checking the first few bytes of the file. It is also possible to import multiple files into a single stream by providing a Unix style pathname pattern as first parameter.

```
1  import obspy
2  st = obspy.read("test.mseed")
3  print st
```

Listing 3-14: *Reading a MiniSEED file from disk using the ObsPy module `obspy.mseed`.*

The script simply returns information about all contained traces within the MiniSEED volume, particularly network, station, location, and channel identifier, start and end date, number of samples and sampling rate.

Module `obspy.imaging`. The `obspy.imaging` module is a collection of standard graphic routines such as plotting waveform data as spectrograms or seismograms, or imaging fault-plane solutions as beach ball plots [Fowler 2005, pp. 130-140]. All plotting functions directly rely on pylab/matplotlib[47], a powerful, open source 2D plotting library for Python capable of producing high quality figures with a syntax almost identical to MATLAB.

Most routines were initially created for the Exupéry project and are required by SeisHub's "seismology" plug-in.

46 http://www.iris.washington.edu/pub/programs/libmseed-2.3.tar.gz
47 http://matplotlib.sourceforge.net

The following example creates two beach ball plots using the moment tensor and the best double couple solutions calculated by the USGS for the 2004 Sumatra earthquake [USGS 2004].

- Moment Tensor:
  Mrr = 0.91, Mtt =-0.89, Mff = -0.02, Mrt = 1.78 , Mrf = -1.55, Mtf = 0.47;
- Double Couple (only using one nodal plane):
  Strike = 274°, Dip = 13°, Slip = 55°.

```
1  from obspy.imaging.beachball import Beachball
2  mt = [0.91, -0.89, -0.02, 1.78, -1.55, 0.47]
3  Beachball(mt, size=300, linewidth=2, color='#777777',
4            file='obspy.imaging.beachball.1.png')
5  np1 = [274, 13, 55]
6  Beachball(np1, size=300, linewidth=2, color='#333333',
7            file='obspy.imaging.beachball.2.png')
```

Listing 3-15: *Generating two beach ball plots using the ObsPy module* `obspy.imaging`.

The Python program in listing 3-15 results into two PNG graphic files, similar to the two images in figure 3-7.

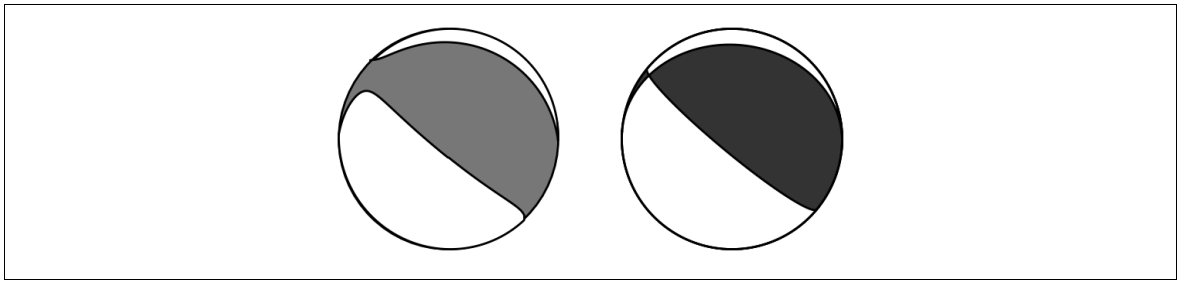

Figure 3-7: *Beach ball plots for a Moment Tensor solution (l.) and the best Double Couple (r.) for the 2004 Sumatra earthquake [USGS 2004] using the ObsPy module* `obspy.imaging`.

Module `obspy.arclink`. This module simplifies requesting waveform, response, and inventory data from any ArcLink server (see chapter 3.5.3). At the current state, ArcLink serves continuous waveform either as full SEED or MiniSEED. Please note that the waveform request is not restricted to event based data. Inventory information may be requested as Dataless SEED or in an ArcLink specific XML format.

SeisHub uses ObsPy's ArcLink module within the waveform mapper to fulfill seismogram data requests not available via the local waveform archive.

The next example illustrates how to request and plot 30 minutes of all three broadband channels ("`BH*`") of station Fürstenfeldbruck ("`FUR`") of the German Regional network ("`GR`") for an seismic event around 2009-08-20 06:35:00 (UTC).

```
from obspy.core import UTCDateTime
from obspy.arclink.client import Client
c = Client()
start = UTCDateTime(2009, 8, 20, 6, 35, 0, 0)
st = c.getWaveform("GR", "FUR", "", "BH*", start, start + 60*30)
st.plot()
```

Listing 3-16: *Retrieving and plotting seismograms using the ObsPy module* `obspy.arclink`.

Waveform data fetched from an ArcLink node is converted into an ObsPy stream object. The seismogram is truncated by the client to the actual requested time span, as ArcLink internally cuts SEED files for performance reasons on record base in order to avoid uncompressing the waveform data (see also chapter 3.5.1). The output of example 3-16 is shown in figure 3-8.

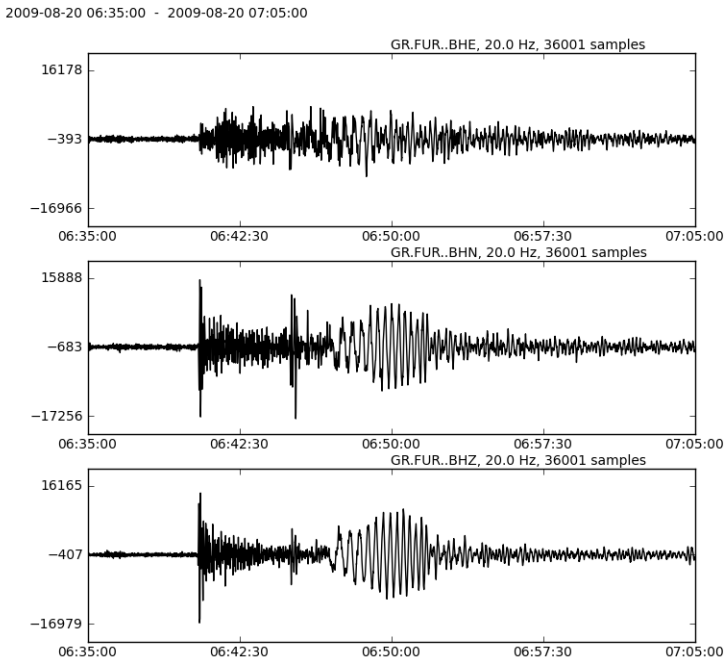

Figure 3-8: *Plotted waveform data using the ObsPy module* `obspy.imaging`.

Module `obspy.xseed`. This module is a converter from Dataless SEED volumes into the XML-SEED format, and vice versa. XML-SEED is a proposed XML markup standard for SEED volumes [Tsuboi et al. 2004]. Additionally, a modified XML-SEED version has been created (labeled as version "1.1") to overcome minor shortcomings of the original standard like typos in tag names, missing blocks of the specification, and inconsistent handling of some data fields. Both versions are not directly compatible, but can be transformed into each other applying the parser to read one format and export into another format.

The module `obspy.xseed` is used within the "seismology" plug-in of SeisHub to generate XML resources for seismic stations. The next example shows how to generate such XML-SEED document from a given Dataless SEED file "`dataless.seed.BW_MANZ`" containing meta data of of station Manzenberg. Listings 2-1 and 2-2 in the XML Essentials chapter already showed excerpts of the input file and the resulting XML document.

```
from obspy.xseed import Parser
p = Parser()
p.parseSEEDFile("dataless.seed.BW_MANZ")
xml_doc = p.getXSEED()
fp = open("dataless.seed.BW_MANZ.xseed", 'w')
fp.write(xml_doc)
fp.close()
```

Listing 3-17: *Generating a XML-SEED document for station Manzenberg using module* `obspy.xseed`.

Module `obspy.seishub`. The SeisHub module is a client implementation for easy interaction with a SeisHub Web service instance. It provides classes and methods for retrieving seismic events, stations, and waveform data using the mappers provided within the "seismology" plug-in of SeisHub.

The next example retrieves exactly 10.5 seconds of all available seismograms of the BayernNetz ("`BW`") at the beginning of the year 2008 from a remote SeisHub server instance running at http://teide.geophysik.uni-muenchen.de:8080. Afterwards, every single trace of each available station, location and channel combination will be stored on the hard disk as a separated seismogram file in the GSE2 standard format. It should be noted that the actual conversion to the GSE2 format is done on client side.

```
from obspy.core import UTCDateTime
from obspy.seishub.client import Client
c = Client("http://teide.geophysik.uni-muenchen.de:8080")
start = UTCDateTime(2008, 1, 1, 0, 0, 0, 0)
st = c.waveform.getWaveform("BW", "*", "*", "*",
                            start, start + 10.5)
for tr in st:
    filename = '%s.%s.%s.%s.gse2' % (tr.stats.network,
                                     tr.stats.station,
                                     tr.stats.location,
                                     tr.stats.channel)
    tr.write(filename, format="GSE2")
```

Listing 3-18: *Retrieving multiple seismograms from SeisHub using the ObsPy module* `obspy.seishub`.

A second example 3-19 fetches waveform, poles, zeros, and gain for a certain seismic station (`BW.RJOB..EHZ`) on a given time span from the same SeisHub server. Poles and zeros are then used to simulate the seismogram with a Wood-Anderson seismometer [NMSOP 2002, c. 11, p. 43]:

- poles: `-6.2832 - 4.7124i`, `-6.2832 + 4.7124i`,
- zeros: `0.0 + 0.0i`, `0.0 + 0.0i`, and
- gain: `1 / 2.25`.

```
from obspy.core import UTCDateTime
from obspy.seishub.client import Client
from obspy.signal import seisSim
import obspy, pylab as pl
# PAZ of Wood-Anderson seismometer
PAZ_WA = {
    'poles': [-6.2832 - 4.7124j, -6.2832 + 4.7124j],
    'zeros': [0.0 + 0.0j, 0.0 + 0.0j],
    'gain': 1. / 2.25
}
# fetching PAZ and waveform via SeisHub
c = Client("http://teide.geophysik.uni-muenchen.de:8080")
start = UTCDateTime(2008, 10, 28, 8, 30, 0)
st = c.waveform.getWaveform("BW", "RJOB", "", "EHZ", start,
                            start + 180)
paz = c.station.getPAZ("BW", "RJOB", start)
df = st[0].stats.sampling_rate
# simulating instrument
data = seisSim(st[0].data, df, paz,
               inst_sim=PAZ_WA, water_level=600.0)
# plotting using matplotlib
T = pl.arange(0, len(st[0])/df, 1/df)
pl.subplot(211)
pl.plot(T, st[0].data, 'k')
pl.title(st[0].getId() + '  ' + str(st[0].stats.starttime))
pl.ylabel(paz['name'])
pl.subplot(212)
pl.plot(T, data, 'k')
pl.ylabel('Wood Anderson')
pl.xlabel('Time [s]')
pl.savefig('obspy.seishub.paz.png', dpi=300)
```

Listing 3-19: *Retrieving zeros, poles, and the gain from SeisHub and simulating a Wood Anderson seismometer using the ObsPy modules* `obspy.seishub` *and* `obspy.signal`.

For in-depth theoretical background for inverse and simulation filtering of digital seismograms, please see Scherbaum [2007, pp. 137-164]. The simulation method `seisSim` itself and the graphical output via pylab/matplotlib was written by Moritz Beyreuther (lines 18-31). However, the example program tries to emphasize on data retrieval (lines 11-17) using the SeisHub server. The program results into graph 3-9 containing two seismograms showing the original waveform recorded with an Streckeisen STS-2 seismometer and the Wood-Anderson simulated seismogram.

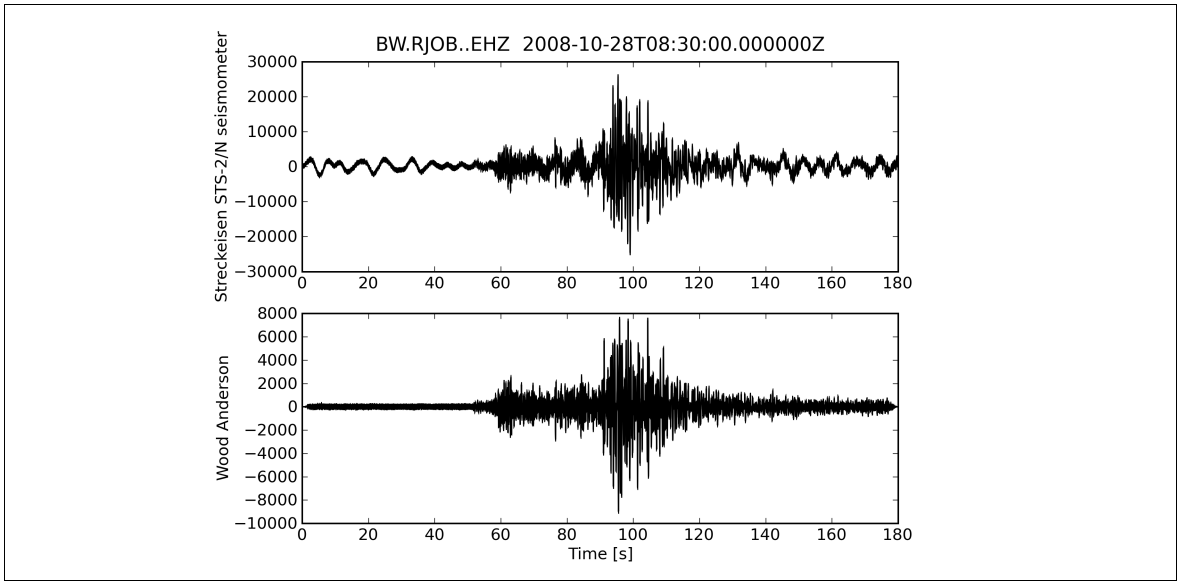

Figure 3-9: *Plots of original seismogram (Streckeisen STS-2) and simulated waveform (Wood-Anderson) using* `obspy.signal` *and matplotlib.*

All modules of the ObsPy package are supporting setuptools[48]. Moreover, ObsPy modules considered stable are additionally registered with the Python Package Index[49] (PyPI).

The actual source code, an auto-generated API documentation, basic installation instructions and a comprehensive tutorial may be found at the ObsPy homepage (http://www.obspy.org). The library is open source software and contributions enhancing its functionality are more than welcome.

48 http://pypi.python.org/pypi/setuptools/
49 http://pypi.python.org/pypi/

3.7 Extreme Programming (XP)

The development process of SeisHub and ObsPy were both heavily driven by the principles of Extreme Programming. The XP approach is a relatively new software development methodology introduced by Kent Beck in 1999. Beck originally described the goals of XP by four basic concepts[50]: simplicity, communication, feedback, and courage [Beck 1999, pp. 32-36].

Simplicity. Traditional theories state that "software becomes increasingly expensive to change over the lifetime of a project" [Burke & Coyner 2003, p. 11]. XP encourages simplicity in software design and coding in order to minimize the cost of change. This means that programmers should use the simplest possible design that fulfills the desired requirements. Extra functionality can be added later, if there is an actual need for it. Simplicity is viewed as the heart of XP because it affects all other concepts significantly.

Communication. This concept can be split between code documentation and communication between developers and/or end users.

Code communicates best to programmers when the concept of simplicity is applied. Supporting techniques are source code comments and self-documenting[51] code. A critical part of code documentation are unit tests which show the design of a individual class, method or function effectively by exposing functionality via concrete examples. XP requires that the major part of the source code is covered by unit tests.

Communication between developers and end users should be done on a regular basis sharing and validating ideas and further project requirements.

Feedback. XP relies on small working steps and short release cycles in order to receive immediate feedback on the current project status. End users may evaluate each newly developed feature and can easily interfere or influence further development.

Direct feedback on the quality of code is retrieved via unit tests. Programmers write unit tests for all programming logic that could possibly break. Also, existing software flaws are best communicated by unit tests that proves the failure of a certain piece of code.

Concrete feedback works together with communication and simplicity. The more feedback you have, the easier it is to communicate. Simple code means more and faster feedback in form of functional tests by a tester or even the end user.

50 Beck refers to them as "values", similar to long term social goals of human societies, which often conflict to short term individual goals.

51 Also known as self-describing source code. Basically a loose set of rules to ensure readable programs, such as human understandable variable names or clean structured source code.

Courage. Basically, developers and end users inexperienced with XP need courage to go for some of the involved XP practices, as e.g. pair programming or test-driven development seem to slow down the initial development progress. Programmers also need courage to take time to try out different approaches and as well be able throw code away for the sake of simplicity. Overall it will improve the cost of change.

Beck derives more general principles and finally concrete practices from the four concepts. None of the principles and practices are completely new – as a matter of fact most are as old as programming itself and its techniques have been proven over decades, especially for the management strategies even for centuries [Beck 1999, p. 9]. The innovation of XP is picking the best practices of other methodologies and combining them.

The following XP practices introduced by Beck have been successfully applied during the SeisHub project:

- **The Planning Game:** The scope of the next release is constantly changed to the current need. A ticketing system has been used to reorder and prioritize identified short term tasks in order to reach long term goals.
- **Simple design:** Following the concept of simplicity, the system has to be designed as simple and elegant as possible. This is harder than it sounds, as elegant simple code needs experience and effort. Also, SeisHub is "only" a prototype produced by a very small team of programmers in a limited time frame. A lot of potential for simplification is probably left and code optimization is a continuing task.
- **Comments:** Source code comments and self-documenting code are key factors for the readability of code. Frankly, software without it is plain useless.
- **Unit tests:** SeisHub and ObsPy have a huge test coverage[52], currently containing over 380 unique test cases mostly applying multiple tests per case. Most software bugs detected have been documented as an own test case in order to prevent recurrence of the same specific issue.
- **Test-driven development (TDD):** This technique takes the practices of unit tests to a more extreme level. It specifies that test cases defining the desired functionality must be written before actually implementing the source code. Most core routines of SeisHub have been designed with TDD.
- **Refactoring:** The source code has been constantly restructured and its design improved in order to fit the concept of simplicity or add further flexibility.

52 Compared with other geophysical software projects known to the author.

- **Pair programming:** Many parts of SeisHub and ObsPy have been written by two or more programmers together directly reviewing each others work. Weekly sprints[53] ensure face-to-face communication between developers which increases the overall productivity and motivation.
- **Collective ownership:** Any developer can change any code at any time. Additional, all source code of SeisHub (http://www.seishub.org) and ObsPy (http://www.obspy.org) have been publicly available in the Internet from the very beginning of each project.
- **Coding standards:** Techniques like pair programming and collective ownership require a standardized coding convention between all parties involved. Like most Python projects, SeisHub and ObsPy adhere to the PEP 8: Style Guide for Python Code [Rossum & Warsaw 2001], PEP 257: Docstring Conventions [Goodger & Rossum 2001], and Epytext Markup Language [Loper 2008], an extension of Python's Docstring conventions.

In order to support the communication and feedback concepts and related practices of XP, in 2006 the author introduced the web-based project managing software Trac[54] and the version control system Subversion[55] at the Department of Earth and Environmental Sciences, Geophysics, LMU Munich. Both software packages are nowadays standard development tools used in many major software projects at the geophysics section of the department. Trac offers a sophisticated issue tracking system, a documentation wiki and a source code browser in one unified web front-end. Trac itself is written in Python, has a very modular layout and is therefore highly customizable. Many plug-ins have been written by the author to improve Trac for a multi-user, multi-project environment[56]. The plug-in architecture of Trac highly inspired the component system of SeisHub.

Extreme Programming proved as an extremely valuable and effective software development technique, once every developer involved adapted the practices stated above. Especially test-driven development increased the overall readability and validness of source code. The author strongly recommends to implement XP practices in all future scientific software projects, even for projects with only a single developer.

53 A brief, intense software development gathering focusing mostly on a single issue.
54 http://trac.edgewall.org
55 http://www.subversion.org
56 http://svn.geophysik.uni-muenchen.de/trac/tracmods/

3.8 Future Extensions

SeisHub has undergone a long developing process due to the rather complex project topic and the very limited manpower actually committed to the project. As already implied in the first chapter of this work: SeisHub is primary a software prototype exploring and demonstrating web-based technologies to handle seismological data. Unfortunately not every idea having emerged over the last years could be implemented so far. This section will cover features which are still missing or need major improvement in future additions.

Insufficient user management. SeisHub offers only a rudimentary user management restricting access to all protected resources through a simple user name and password combination. Although multiple users (and passwords) may be created, all get the same access rights granted within SeisHub. A sophisticated user management is not a trivial task, it has to be elaborated on real use cases. The latter are missing for now, but are needed to solve all questions of scope, such as: On what level should SeisHub be restricted - service, components or even each single resource? Do granted rights inherit from parent resources underlying objects? Is a plain URL-based restriction of resources feasible? Is a group or role based middle layer necessary? There is no way to implement a solution which covers all possible scenarios mentioned above. For now, the implemented user management fulfills the requirement for a basic protection of resources. However, this is definitively a weak point of SeisHub, which should be improved once there are specific needs for it.

Data sharing. A very unique feature of SeisHub would be the possibility to share automatically or on request data amongst different SeisHub servers. This idea arose in a very early development stage of SeisHub and some technical details related to this topic are already solved and implemented. Examples are the (for now completely disabled) heartbeat service sharing a list of running SeisHub instances, or the (yet unused) origin field in the meta data section of every XML resource to reflect the real source of a document. SeisHub's architecture certainly offers the possibility to explore this feature.

Limited database support. One of SeisHub's early design goals to support every SQL database able to interface with the SQLAlchemy library has not been reached, because a few dialect specific features had to be used within this project. One example is the PostGIS[57] spatial database extension for PostgreSQL required by the Exupéry project (see chapter 4.1). However, this is negligible for GIS independent projects. More restricting is the missing support for SQL Views in SQLAlchemy, which must be added manually for each SQL dialect used. There are also other minor issues related

57 http://postgis.refractions.net

to the automated creation of unique primary keys. Finally it was decided for now to support only the databases SQLite and PostgreSQL in order not to downgrade development progress. Supporting more databases later on is possible assuming there is an actual need for it within the seismological community.

Limits on the database back-end. At the start of the project in 2006 native XML databases were rather rare and mostly not open source software. The situation has eased as many new open source projects focusing on this topic appeared over the last few years. A significant amount of brainstorming and development time has been used for the correct and fast implementation of XML to SQL mapping within SeisHub. Still there are some unsolved issues with the current implementation, such as the limited XML namespace and grouping support for indexes as mentioned in chapter 3.4.2. Developers of future versions of SeisHub should try to improve these issues. Additionally, they should carefully watch the XML database market and even consider changing the database back-end at one point. This certainly invalidates part of this study, on the other hand the benefit of using a community supported open source native XML database tested and improved by many people may outweigh the drawbacks. However, such a decision would result in numerous changes of SeisHub's core components connected to data handling. It also conflicts with the goal of having a plain SQL database back-end working behind the scene.

Fast waveform previews. The waveform indexing process could also be used to generate an additional file of each daily SEED volume containing minimal and maximal values at a very low frequency, e.g. every 60 seconds. Those files could be used in a GUI or web-based front-end for a fast preview of seismic events over multiple channels and large time spans.

4 Scientific Application of SeisHub

Introduction. SeisHub's ability to process, store, and distribute collocated multi-component, multi-disciplinary data has already been proven by the two geophysical projects: Exupéry, a decision support system for a mobile volcano monitoring system, and BayernNetz, the seismological network of the Bavarian Seismological Service (Erdbebendienst Bayern). Both projects and relevant plug-ins developed for SeisHub will be introduced throughout the following two chapters.

4.1 Exupéry - Volcano Fast Response System

SeisHub's first real application as database is Exupéry[58], a mobile Volcano Fast Response System (VFRS) that can quickly be deployed for volcanic monitoring in case of a volcanic crisis or volcanic unrest anywhere in the world [Hort & Zakšek 2008]. The primary goals of the Exupéry project are providing a stable communication basis for stations in the field and an expert system collecting data from various sources into a centralized database that allows scientists and local authorities to assess the data through one unified web-based GIS interface [Bernsdorf et al. 2009]. Field data is hereby collected via a relatively new distributed wireless network approach based on mesh nodes creating a self-organized multi-point to multi-point network. This technology is used for communication with established ground-based standard volcanic monitoring techniques, such as:

- seismology using multiple three-component broadband instruments,
- ground deformation via GPS sensors and a IBIS-L[59] Ground Based Synthetic Aperture Radar (GBSAR) device,
- remote sensing of areal SO_2 distribution accompanied by measurements of SO_2 and BrO flux, and
- plume speed using the dual-beam Miniature Differential Optical Absorption Spectrometer (Mini-DOAS).

A novelty of the Exupéry project is the direct integration of satellite-based observations like (satellite names in brackets):

- areal mapping of ground deformation using DInSAR (Differential Interferometric Synthetic Aperture Radar) and PInSAR (Polarimetric Interferometric Synthetic Aperture Radar) methods (Envisat, ERS1,2, ALOS, TerraSAR),
- wide area degassing signatures, especially SO_2 (GOME-2), and

58 http://www.exupery-vfrs.de
59 http://www.idscompany.it/page.php?f=179&id_div=4

- detection of thermal activity via infrared imaging (AVHRR, GOES, MODIS, ASTER).

The system also includes an automatic alert level estimation in order to characterize the activity state of the volcano. The alert level is directly incorporated into the GIS, supporting scientists and local authorities allowing to improve the decision making process in the case of a volcanic unrest.

Exupéry involves researchers of nine different research institutions allover Germany and is funded by the German Ministry for Education and Research (BMBF) within the Geotechnologien[60] project [Zakšek 2008]. The project is divided into five work packages among the project partners. One of the work packages, supervised by Joachim Wassermann (Department of Earth and Environmental Sciences, Geophysics, LMU Munich) and Klaus Stammler (BGR), focuses on the expert system containing the central geophysical database (SeisHub), the GIS interface (developed by Jena-Optronik GmbH[61]), and the alert level estimation (Moritz Beyreuther, Department of Earth and Environmental Sciences, Geophysics, LMU Munich).

Although the development of SeisHub was not directly initiated by Exupéry, many architectural decisions were strongly influenced by its database requirements. The project offered an ideal and realistic test scenario as the demanded centralized geophysical database had to handle all kinds of data categories:

- event-based and continuous data,
- stationary, ground-based measurements and satellite data,
- time series, images (2D), and models (3D).

These very specific demands within the Exupéry project opened up new insights and ideas which improved the overall outcome of SeisHub. The following components and extensions have been created by the author for the Exupéry project:

- plug-in `trunk/plugins/seishub.plugins.exupery` including package “exupery“ and thirteen different resource types reflecting all possible data sources covered within this project,
- various XSLT documents for format conversion of resource types (all GIS related),
- several mapper classes offering services for the GIS system,
- imaging routines for plotting fault-plane solutions (beach balls) and seismograms (see chapter 3.6), and

60 http://www.geotechnologien.de
61 http://www.jena-optronik.com

- new base component interface class allowing to create user-defined SQL Views, primary used for the non-standard, spatial database extension PostGIS[62] for PostgreSQL needed by the GIS system.

All resource type specific validation schemas were delivered by the BGR. Further many seismic functionalities of the "seismology" package are also directly used and therefore tested within Exupéry.

The close cooperation with the developers of the GIS system, as first real Web service client for SeisHub's RESTful Web service, greatly influenced and enhanced the implementation design and handling of server-side mapper classes. The front-end of the GIS system is a Flash[63] application embedded into a website. The Flash plug-in is available free of charge for all major browsers. The GIS system is backed by the Java application server Apache Tomcat[64] deploying the mature open source software GeoServer [Garnett et al. 2009] able to handle standardized georeferenced datasets. GeoServer has further been adapted to interface with the SeisHub database. The GIS system also incorporates maps from the Google Maps Web service as default cartographic material.

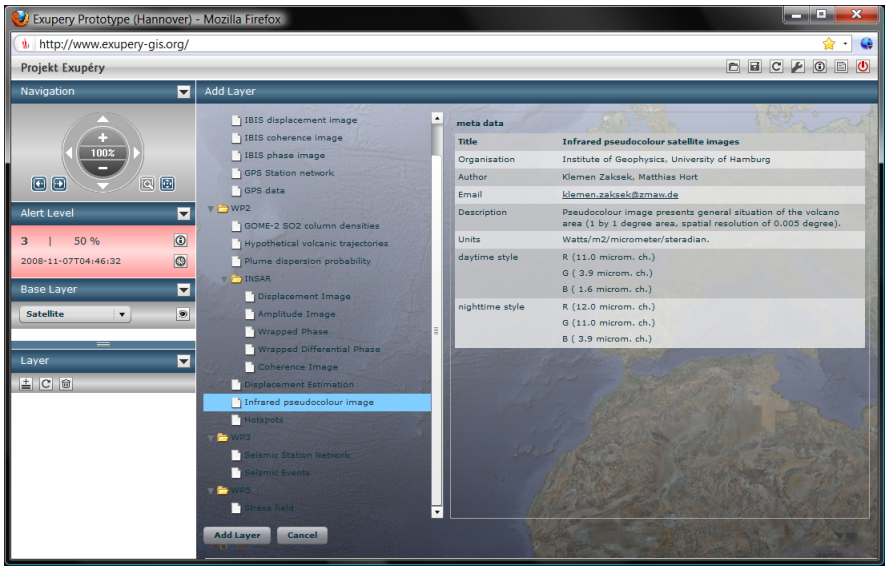

Figure 4-1: *Data layer selection panel "Add layer" of Exupéry's web-based GIS front-end (right) and the estimated "Alter level" panel for the selected volcano and timespan (middle left).*

Figures 4-1, 4-2, and 4-3 give a small impression of the layout and functionality of the web-based GIS front-end. The combination of GeoServer and the Flash application are responsible for preprocessing and rendering of all seen data. However, GeoServer frequently queries the SeisHub database fetching data and/or meta-data of available resources, partly converting them into

62 http://postgis.refractions.net
63 http://www.adobe.com/de/products/flashplayer/
64 http://tomcat.apache.org

GeoServer compatible formats, or accessing primary data directly via SQL statements. Those three application fields will be examined briefly in the next paragraphs.

The first data layer in picture 4-2 displaying volcanic SO_2 column densities is essentially given from the data provider as image in the georeferenced Tagged Image File Format (GeoTIFF). A GeoTIFF image may contain all kinds of information for exact spatial reference, however it cannot cover additional details like measurement methods and devices, or secondary, derived information. Such files are therefore always associated with an XML resource within SeisHub, providing the required (meta-)data and pointing to the original GeoTIFF image on the local file system. Besides fetching resources, GeoServer also uses that meta information to provide meaningful layer-based setup dialogs for proper filtering of data, e.g. restricting date chooser to minimum and maximum data values of all entries.

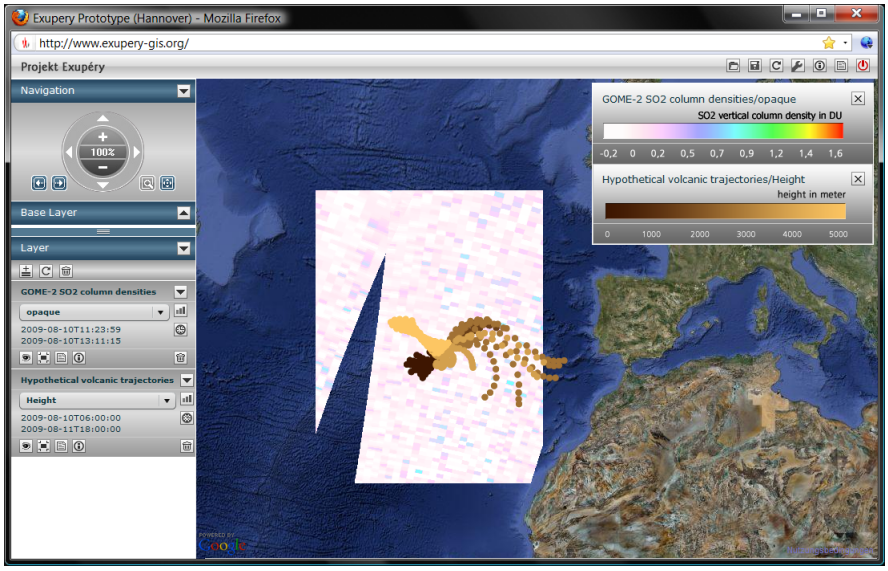

Figure 4-2: *Two activated georeferenced satellite-based data layers: (1) volcanic SO_2 column densities, and (2) hypothetical volcanic trajectories of plumes.*

The direct data conversion from stored XML resources into GeoServer compatible formats is demonstrated in the second data layer of figure 4-2. The visualized hypothetical volcanic trajectories are saved on daily basis as a XML resource in SeisHub basically containing a point cloud of latitude, longitude, and height values complemented by additional meta information. The point cloud of such resources can be transformed on request with a XSLT document into the Keyhole Markup Language (KML) standard [OGC 2008]. KML is used directly within GeoServer, but also may be viewed by any other products able to process such geospatial data.

GeoServer's direct access via SQL statements is shown for the seismic station layer in picture 4-3. Data of each station is stored as an unique XML resource within SeisHub. In order to generate a geospatical object layer which can be understood and processed by the GeoServer requires an

iteration over all XML station resources (or their indexes) which can be time consuming for large datasets. Using the PostGIS extension combined with SQL Views reduces the processing time significantly, as PostGIS can be handled from GeoServer very efficiently.

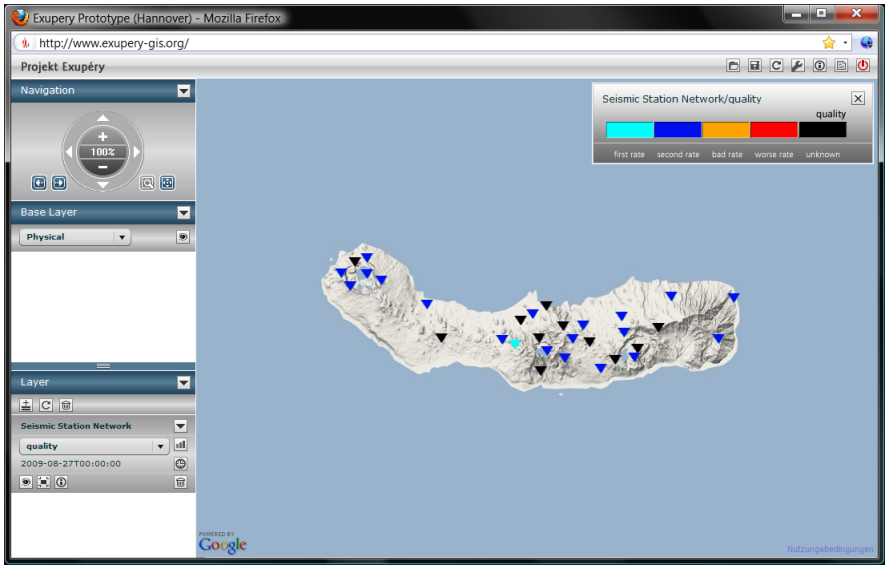

Figure 4-3: *Seismic station network quality layer during the Azores field test (April – September 2009).*

The Exupéry prototype including all necessary software components has been running at the BGR in Hannover since spring 2009. Satellite data is automatically fetched via scripts from the respective data providers and included into the database. Ground-based data storage and retrieval has been tested only with sample data provided by the responsible data providers, due to the lack of real data at this point. Additionally, a similar setup of the Exupéry prototype has been tested under field conditions at the Azores since April 2009. This setup includes over 20 seismic stations, the ground measurements of deformation and gas flux. Merely ground-based data could be collected due to limited Internet bandwidth. All data retrieved at the Azores was synchronized at daily base with the BGR setup.

SeisHub proved for the Exupéry project as a very stable geophysical database server and Web service, as no major issues or bugs have occurred yet during the whole runtime. This is by far the most important requirement for any server software. The concept of easy and fast extensibility by adding new indexes, schemas, style sheets, mappers, and resource types has been applied by the author and others many times. The setup process of SeisHub has been tested by various people within the project several times and was reported as feasible. The GIS system and other client programs made excessive use of SeisHub's Web service interfaces proofing its reliability. Last but not least, SeisHub already seems to have an impact in the volcano community, as scientists with similar projects show great interest in keeping things compatible or reducing the work and cost of creating their own volcanological database.

4.2 Bavarian Seismological Network (BayernNetz)

BayernNetz is a modern seismological network of the Bavarian Seismological Service[65] (Erdbebendienst Bayern), a cooperation between the Bavarian Environment Agency[66] (Bayerisches Landesamt für Umwelt) and the Department of Earth and Environmental Sciences, Geophysics, LMU Munich. BayernNetz has been designed for monitoring local seismic activity in Bavaria and bordering regions for events of a local magnitude (Richter magnitude) M_L of 2.0 and above [Kraft 2006]. It currently (September 2009) consist of five broadband and eighteen short period seismometer stations, which data is collected, archived and analyzed at the Geophysical Observatory in Fürstenfeldbruck near Munich. Waveform data received from broadband stations are additionally transmitted in real time to the European data center ORFEUS and BGR.

The hardware and software setup of BayernNetz for automated and continuous recording, transferring and archiving of waveform data is state of the art and therefore well equipped for the future. However, there is a growing deficit of software for analyzing and processing this data. The current seismic database and processing GUI used on a daily base by the analyzing seismologists at the observatory is the software package GIANT [Rietbrock 1996; Rietbrock & Scherbaum 1998]. The Graphical Interactive Aftershock Network Toolbox (GIANT) was written by Rietbrock during his doctoral thesis in 1996 and is closely connected with Scherbaum's Programmable Interactive Toolbox for Seismological Analysis (PITSA). GIANT uses the proprietary, embedded database system “db_Vista” from Raima Corporation[67] as a data back-end. The package also grants access to various standard analysis programs such as HYPO71 [Lee & Lahr 1972; Lee & Lahr 1975] or FOCMEC [Snoke et al. 1984].

GIANT proved over the last decade as a reliable database system for event-based waveform processing, but it is rather ill-prepared for working with continuous multi-parameter datasets. A major goal of this study was to create a flexible database able to serve event-related data structures as well as continuous data streams. The SEEDFileMonitor service (see chapters 3.3.2 and 3.5) has been developed in order to fulfill these requirements with SeisHub. Additionally, all needed parametric data types and mappers are defined in the plug-in “seismology”. The latter is currently able to handle automatically detected or manually reviewed seismic events, station information, and event-based and continuous waveform data. All seismic data types and related mappers will briefly be introduced within the next paragraphs.

65 http://www.erdbeben-in-bayern.de
66 http://www.lfu.bayern.de
67 http://www.raima.com

Seismic stations: Seismic network operators usually deliver station specific meta data either via Dataless SEED files or via so-called response files (RESP) [IRIS 2009c], a derived ASCII format of Dataless SEED. Both formats are not XML-based and therefore not directly deployable within SeisHub. The XML-SEED converter introduced with the ObsPy library (chapter 3.6) allows for generating standard XML-SEED documents from Dataless SEED files. These XML documents are used in SeisHub as unique station resources stored within the resource type "station" in the package "seismology". Please be aware that the term seismic station is actually incorrect, as each XML-SEED resource is restricted to describing a single seismic device using a unique combination of network, station, and location codes (see chapter 3.5.1) during a certain time span, but may have multiple channel codes declared. Also, any modification of the station setup such as replacing an instrument, must be announced by a new XML-SEED document.

The complete overview of all available station resources within the database can be found at http://localhost:8080/xml/seismology/station/[68]. XML station resources are child nodes of this folder. The format option "map" may be used to retrieve a simple XHTML representation of a station resource (see figure 3-6 for station Manzenberg, BayernNetz).

There are two station-specific mappers delivered in the "seismology" plug-in.

1. Mapper http://localhost:8080/seismology/station/getList returns a list of all indexed values of each single seismic station within SeisHub used by the `obspy.seishub` client.
2. Mapper http://localhost:8080/seismology/station/dataless/XXX can be used to convert a single station resource on the fly from the XML-SEED into the Dataless SEED format using the `obspy.xseed` module.

The final part "XXX" has to be replaced by the actual resource name of a station XML resource, e.g. "dataless.seed.BW_MANZ.xml" for station Manzenberg of the BayernNetz. All available resource names can be seen either by the station overview or by using the first mapper.

Seismic events: Seismic events are currently stored in a custom, simplified XML dialect derived from QuakeML. It is not 100% compatible with the currently available version of QuakeML, as the latter is still under heavy development closely connected to the NERIES project (see chapter 3.5.3). However, the transformation process from the internally used format into a finalized, officially accepted QuakeML standard should not be difficult, as they are closely related.

Seismic event resources are generated either by the real-time processing system Earthworm (see chapter 3.5.1) or are stored after a manual review of a seismic event using GIANT. Both applications have been extended with a small upload program, which creates and transmits XML resources to SeisHub. Unique identifiers used within Earthworm and GIANT are also applied to

68 For a user friendly output in a browser use the additional parameter "?format=xhtml".

generate unique resource names for the XML database. All stored XML-based event resources can be browsed via http://localhost:8080/xml/seismology/event/. Similar to seismic station resources, the option "map" can be used to retrieve a simple XHTML representation of a seismic event.

There are currently two event-specific mappers delivered in the "seismology" plug-in.

1. Mapper http://localhost:8080/seismology/event/getList returns a list of all indexed values of each seismic event within SeisHub used by the `obspy.seishub` client.
2. Mapper http://localhost:8080/seismology/event/plotBeachball creates a graphical fault-plane solution as beach ball plot by using the `obspy.imaging` module.

Waveform data: All seismic waveform data recorded within the BayernNetz are automatically fetched, processed, and archived by the real-time data acquisition and processing system SeisComP. Time series are hereby stored into a central file archive of daily, channel-specific MiniSEED files (see also SDS, chapter 3.5.1). These waveforms are automatically indexed by SeisHub using the SEEDFileMonitor service, which is elaborated in chapter 3.5.4. The mapper unifying the request for archived seismograms and the retrieval of externally waveform data via the ArcLink protocol is discussed in full detail in chapter 3.5.5.

A SeisHub instance has been running at the Geophysical Observatory Fürstenfeldbruck since spring 2009. This setup is currently used parallel to the GIANT database back-end to evaluate its features and eventually smooth the transition process to the new technology. The author currently focuses on finding a suitable graphical processing tool (preferably based on Python) capable of interconnecting with SeisHub. Please be aware that the latter was not part of this study. However, first steps to solve this issue have been taken with the Exupéry GIS. Moreover, the author is in close contact with the developers of the next generation of the seismic processing tool Seismic Handler [Stammler & Walther 2009], who expressed great interest in incorporating the ObsPy library and interfacing with SeisHub.

5 Conclusions

The object of this study was to develop a database prototype demonstrating and exploring web-based technologies in the field of seismology. Designing such a novel database system from scratch allows a developer to immediate many of the deficiencies of existing seismic databases outlined in the general introduction of this thesis. SeisHub, the database approach introduced by the author, offers a suitable, modern solution for the following known problems of currently deployed database systems. In more detail, SeisHub

1. serves event-related data structures as well as continuous seismic data streams,
2. stores and distributes collocated multi-component, multi-disciplinary data,
3. handles quality information delivered within seismic data streams,
4. can import waveform data from neighboring seismological networks via the ArcLink/WebDC approach,
5. can store data and models of computed synthetic waveforms for specific events or region of interest,
6. has a modern, modular, platform independent software architecture, and
7. is completely based on standards and open source software.

Another achievement of this study is the foundation of the seismological library ObsPy for the Python programming language. ObsPy offers a wide spectrum of sophisticated tools for working with seismic data such as waveform, inventory or event-related data. Many submodules were directly influenced by the development of SeisHub and most of the ObsPy's core classes are now standard dependencies of SeisHub. Noteworthy modules developed by the author are:

1. `obspy.arclink`, a comprehensive ArcLink/WebDC client;
2. `obspy.xseed`, a format conversion tool between Dataless SEED and the XML-SEED standard; and finally
3. `obspy.seishub`, a RESTful Web service client implementation able to interface with a SeisHub server.

Certain features of the SeisHub database system and the ObsPy library are unique within the seismological community and therefore emphasized again in the following paragraph.

1. SeisHub is able to process and distribute event-based and continuous seismic waveform data as well as collocated multi-component, multi-disciplinary datasets within one single database system. This is achieved by its flexible, modular database design and by using XML resources as fundamental unit of storage for parametric datasets.

2. Waveform data requested by a client and not available within the local waveform archive can automatically be fetched by SeisHub from remote seismic data centers using the ArcLink protocol. Results are directly incorporated into the local waveform database.
3. The `obspy.xseed` module of the ObsPy library offers the first open source, platform independent conversion tool for Dataless SEED volumes into the XML-SEED standard format, and vise versa.

The database structure designed within this study forms the nucleus for future developments. Further steps will focus on the adaptation of a suitable graphical processing tool using SeisHub as a data back-end. SeisHub's ability to manage observational as well as synthetic seismograms offers entirely new ways of analyzing data, extracting (differential) travel times, identifying systematic misfits in data, etc. Once such a processing tool is available, it will have a strong impact on the practice of analyzing, processing, and interpreting of waveform data within seismology.

References

[Ahern 2000] Ahern, T., Automated Data Handling within the IRIS DMS, *IRIS DMS Electronic Newsletter*, vol. 2, no. 4.

[Beck 1999] Beck, K., Extreme Programming Explained: Embrace Change, Addison-Wesley Longman, Amsterdam, ISBN 0-321-27865-8.

[Berners-Lee et al. 2005] Berners-Lee, T., R. Fielding, and L. Masinter, Uniform Resource Identifier (URI): Generic Syntax, STD 66, RFC 3986, http://tools.ietf.org/html/rfc3986 (last accessed 15 September 2009).

[Bernsdorf et al. 2009] Bernsdorf S., R. Barsch, K. Zakšek, M. Beyreuther, M. Hort, and J. Wassermann, Decision support system for the mobile volcano fast response system, *Int. Journal of Digital Earth (IJDE)*, submitted.

[Bos 2003] Bos, B., XML in 10 points, http://www.w3.org/XML/1999/XML-in-10-points.html (last accessed 15 September 2009).

[Bourret 2009] Bourret R., XML Database Products:XML-Enabled Databases, http://www.rpbourret.com/xml/ProdsXMLEnabled.htm (last accessed 15 September 2009).

[BRTT 2009] BRTT: Boulder Real Time Technologies, Boulder Real Time Technologies, http://www.brtt.com (last accessed 15 September 2009).

[Burke & Coyner 2003] Burke, E. M., and B. M. Coyner, Java Extreme Programming Cookbook, O'Reilly Media, Inc., ISBN 978-0-596-00387-6.

[Buxmann et al. 2003] Buxmann P., E. Wüstner, R. Barsch, C. Rödel, and S. Schade, <x: act> -Ein Webservice für die Konvertierung von XML-Dokumenten, *Wirtschaftsinformatik*, vol. 2, pp. 143-160.

[Casey & Ahern 1999] Casey R., and T. Ahern, Technical Manual for Networked Data Centers (NetDC) Protocol, http://www.iris.edu/manuals/netdc/NETDC.htm (last accessed 15 September 2009).

[Codd 1970] Codd, E.F., A Relational Model of Data for Large Shared Data Banks, *Communications of the ACM* , vol. 13, no. 6, pp. 377–387.

[Costello 2009] Costello, R. L., Building Web Services the REST Way, http://www.xfront.com/REST-Web-Services.html (last accessed 15 September 2009).

[Daconta et al. 2003] Daconta, M. C., L. J. Obrst, and Smith K. T., The Semantic Web: A guide to the future of XML, Web Services and Knowledge Management, Wiley Publishing, Inc., ISBN 0-471-43257-1.

[Davis 2008] Davis, S., Mastering Grails: RESTful Grails, http://www.ibm.com/developerworks/library/j-grails09168/ (last accessed 15 September 2009).

[Eck & Sleeman 2008] Eck, T. van, and R. Sleeman, ORFEUS Activity Report 2007, http://www.orfeus-eu.org/Documents/ORFEUS_report_2007.pdf (last accessed 15 September 2009).

[ECMA 1999] European Computer Manufacturers Association, ECMAScript Language Specification, http://www.ecma-international.org/publications/files/ECMA-ST/ECMA-262.pdf (last accessed 15 September 2009).

[Evjen et al. 2007] Evjen B., K. Sharkey, T. Thangarathinam, M. Kay, A. Vernet, and S. Ferguson, Professional XML (Programmer to Programmer), John Wiley and Sons Ltd, ISBN 0-471-77777-3.

[Faassen 2005] Faassen, M., What is Pythonic?, http://faassen.n--tree.net/blog/view/weblog/2005/08/06/0 (last accessed 15 September 2009).

[Fettig 2005] Fettig, A., Twisted Network Programming Essentials, O'Reilly Media, Inc., ISBN 0-596-10032-9.

[Fielding 2000] Fielding, R., Architectural Styles andthe Design of Network-based Software Architectures, Dissertation, University of California, Irvine.

[Fowler 2005] Fowler, C. M. R., The Solid Earth: An Introduction to Global Geophysics, Cambridge University Press, Cambridge, ISBN 0-521-89307-0.

[Garnett et al. 2009] Garnett, J. et al., GeoServer Webpage, http://geoserver.org (last accessed 15 September 2009).

[GFZ 2008] GFZ, GEOFON Data Archive Information, http://geofon.gfz-potsdam.de/geofon/new/arc_inf.html (last accessed 15 September 2009).

[GFZ 2009] GFZ, SeisComP Data Structure (SDS) 1.0, http://geofon.gfz-potsdam.de/geofon//seiscomp/SDS.html (last accessed 15 September 2009).

[Goldstein 1998] Goldstein, P. et al., SAC2000 User's Manual, http://www.iris.edu/manuals/sac/manual.html (last accessed 15 September 2009).

[Goodger & Rossum 2001] Goodger, D., and G. van Rossum, Docstring Conventions, http://www.python.org/dev/peps/pep-0257/ (last accessed 15 September 2009).

[Graham et al. 2004] Graham, S., D. Davis, S. Simeonov, G. Daniels, P. Brittenham, Y. Nakamura, P. Fremantle, D. Koenig, and C. Zentner, Building Web Services with Java: Making Sense of XML, SOAP, WSDL, and UDDI (2nd Edition), , ISBN 978-0672326417.

[GSE 1997] Group of Scientific Experts Third Technical Test, GSETT-3, Provisional GSE2.1, Message Formats & Protocols, Operations Annex 3, May 1997,

http://www.seismo.ethz.ch/autodrm/downloads/provisional_GSE2.1.pdf (last accessed 15 September 2009).

[Hadley 2009] Hadley M. J., Web Application Description Language (WADL), https://wadl.dev.java.net/wadl20090202.pdf (last accessed 15 September 2009).

[Hanka & Kind 1994] Hanka, W., and R. Kind, The GEOFON Program, *Annali di Geofisica*, vol. 37, no. 5, pp. 1060-1065.

[Hanka & Saul 2008] Hanka, W., and J. Saul, GEOFON and its Role in Earthquake Monitoring and Tsunami Warning - In: Husebye, E. S. (Eds.), Earthquake Monitoring and Seismic Hazard Mitigation in Balkan Countries, 151-162.

[Hanka et al. 2000] Hanka, W., A. Heinloo, and K.-H. Jaeckel, Networked Seismographs: GEOFON Real-Time Data Distribution, *ORFEUS Newsletter*, vol. 2, no. 3.

[Havskov & Ottemöller 2000] Havskov, J., and L. Ottemöller, SEISAN earthquake analysis software, *Seismological Research Letters (SRL)*, 70, pp. 532-534.

[Havskov et al. 2007] Havskov J., L. Ottemöller, and R. L. P. Canabrava, SEISAN: Multiplatform implementation of MINISEED/SEED, *ORFEUS Newsletter*, vol. 7, no. 2.

[Heinloo 2001] Heinloo, A., Seedlink: The Missing Link for Real-time Earthquake Monitoring, M.Sc. Thesis, Tartu University, Estonia.

[Hort & Zakšek 2008] Hort, M., and K. Zakšek, Managing volcanic unrest: The mobile volcano fast response system, Use of Remote Sensing Techniques for Monitoring Volcanoes and Seismogenic Areas 2008, pp. 1-6.

[IASPEI 2008] IASPEI Working Group on Station Codes, Seismic Station Codes – New Coding Standards, http://www.iaspei.org/commissions/CSOI/SNSC_draft_document.pdf (last accessed 15 September 2009).

[IRIS 2009a] IRIS Consortium, SEED Reference Manual - Standard for the Exchange of Earthquake Data, SEED Format Version 2.4, http://www.iris.washington.edu/manuals/SEEDManual_V2.4.pdf (last accessed 15 September 2009).

[IRIS 2009b] IRIS Consortium, Network codes, http://www.iris.edu/stations/networks.txt (last accessed 15 September 2009).

[IRIS 2009c] IRIS Consortium, EVALRESP Manual (V3.3.0) , http://www.iris.washington.edu/manuals/evalresp.htm (last accessed 15 September 2009).

[IRIS 2009d] IRIS Consortium, SeedLink, http://www.iris.edu/data/dmc-seedlink.htm (last accessed 15 September 2009).

[ISO/IEC 19757-2] ISO/IEC, Information technology - Document Schema Definition Language (DSDL) - Part 2: Regular-grammar-based validation - RELAX NG, http://standards.iso.org/ittf/PubliclyAvailableStandards/c037605_ISO_IEC_19757-2_2003%28E%29.zip (last accessed 15 September 2009).

[ISO/IEC 19757-3] ISO/IEC, Information technology - Document Schema Definition Language (DSDL) - Part 3: Rule-based validation - Schematron, http://standards.iso.org/ittf/PubliclyAvailableStandards/c040833_ISO_IEC_19757-3_2006(E).zip (last accessed 15 September 2009).

[ISO/IEC 9075-1] ISO/IEC, Information technology - Database languages - SQL - Part 1: Framework (SQL/Framework), http://standards.iso.org/ittf/PubliclyAvailableStandards/c045498_ISO_IEC_9075-1_2008.zip (last accessed 15 September 2009).

[Johnson et al. 1995] Johnson, C. E., A. Bittenbinder, B. Bogaert, L. Dietz, and W. Kohler, Earthworm: A flexible approach to seismic network processing, *IRIS Newsletter*, vol. 14, no. 2, pp. 1-4.

[Jones & Drake 2002] Jones, C. A., and F. L. Drake, Jr., Python & XML, O'Reilly Media, Inc., ISBN 0-596-00128-2.

[Korpela 2006] Korpela, J. K., Unicode Explained, O'Reilly Media, Inc., ISBN 0-596-10121-X.

[Kraft 2006] Kraft, T., A New Seismological Network for Bavaria and its Application to the Study of Meteorologically Triggered Earthquake Swarms, Dissertation, Department of Earth and Environmental Sciences, LMU Munich.

[Kulchenko & Ray 2002] Kulchenko P., and R. J. Ray, Programming Web Services with Perl, O'Reilly & Associates, Inc., ISBN 0-596-00206-8.

[Laux & Martin 2000] Laux A., and L. Martin , XML:DB Initiative: XUpdate - XML Update Language, http://xmldb-org.sourceforge.net/xupdate/xupdate-wd.html (last accessed 15 September 2009).

[Lee & Lahr 1972] Lee, W. H. K., and J. C. Lahr, HYP071: A computer program for determining hypocenter, magnitude, and first motion pattern of local earthquakes, *Open File Report, U. S. Geological Survey*, 100 pp.

[Lee & Lahr 1975] Lee, W. H. K. and J. C. Lahr, HYP071 (Revised): A computer program for determining hypocenter, magnitude, and first motion pattern of local earthquakes, *U. S. Geological Survey Open File Report 75-311*, 113 pp.

[Lomax 1991] Lomax, A. J., User Manual for SeisGram. In Digital Seismogram Analysis and Waveform Inversion, http://alomax.free.fr/seisgram/SeisGram2K.html (last accessed 15 September 2009).

[Loper 2008] Loper, E., The Epytext Markup Language, http://epydoc.sourceforge.net/manual-epytext.html (last accessed 15 September 2009).

[Lutz 2006] Lutz, M., Programming Python, O'Reilly Media, Inc., ISBN 0-596-00925-9.

[Maier et al. 2009] Maier, R., T. Hädrich, and R. Peinl, Enterprise Knowledge Infrastructures, Springer, Berlin, ISBN 978-3-540-89767-5.

[Martelli et al. 2005] Martelli, A., A. M. Ravenscroft, and D. Ascher, Python Cookbook, O'Reilly Media Inc., ISBN 0-596-00797-3.

[Melton & Buxton 2006] Melton, J., and S. Buxton, Querying XML - XQuery, XPath, and SQL/XML in Context, Morgan Kaufmann, ISBN 1-55860-711-0.

[Møller & Schwartzbach 2006] Møller A., and M. I. Schwartzbach, An Introduction to XML And Web Technologies, Addison-Wesley, ISBN 0-321-26966-7.

[NMSOP 2002] Bormann, P. (Ed.), IASPEI New Manual of Seismological Observatory Practice (NMSOP), Volume 1, Geoforschungszentrum Potsdam, ISBN 3-9808780-0-7.

[OASIS 2004] OASIS Committee, UDDI Version 3.0.2, http://uddi.org/pubs/uddi_v3.htm (last accessed 15 September 2009).

[OGC 2008] Open Geospatial Consortium Inc., OGC KML, http://www.opengeospatial.org/standards/kml/ (last accessed 15 September 2009).

[ORFEUS 2009] ORFEUS, ORFEUS Annual Report 2008, http://www.orfeus-eu.org/Documents/ORFEUS_Annual_Report_2008.pdf (last accessed 15 September 2009).

[Pautasso et al. 2008] Pautasso, C., O. Zimmermann, and F. Leymann, RESTful Web Services vs. Big Web Services: Making the Right Architectural Decision, 2008, 805-814.

[Ray 2001] Ray, E.T., Learning XML, O'Reilly Media, Inc., ISBN 978-0-596-00046-2.

[RFC 2616] Fielding, R., J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, T. Berners-Lee, Hypertext Transfer Protocol - HTTP/1.1, http://www.ietf.org/rfc/rfc2616.txt (last accessed 15 September 2009).

[RFC 2617] Franks J., P. Hallam-Baker, J. Hostetler, S. Lawrence, P. Leach, A. Luotonen, and L. Stewart, HTTP Authentication: Basic and Digest Access Authentication, http://www.ietf.org/rfc/rfc2617.txt (last accessed 15 September 2009).

[RFC 4252] Ylonen, T., and C. Lonvick, The Secure Shell (SSH) Authentication Protocol, http://www.ietf.org/rfc/rfc4252.txt (last accessed 15 September 2009).

[RFC 5023] Gregorio J., and B. de hÓra, The Atom Publishing Protocol, http://tools.ietf.org/rfc/rfc5023.txt (last accessed 15 September 2009).

[Richardson & Ruby 2007] Richardson L., and S. Ruby, RESTful Web Services, , ISBN 0-596-52926-0.

[Rietbrock & Scherbaum 1998] Rietbrock, A., and F. Scherbaum, The GIANT Analysis System (Graphical Interactive Aftershock Network Toolbox), *Seismological Research Letters (SRL)*, vol. 69, no. 1, pp. 40-45.

[Rietbrock 1996] Rietbrock, A., Entwicklung eines Programmsystems zur konsistenten Auswertung großer seismologischer Datensätze mit Anwendung auf die Untersuchung der Absorptionsstruktur der Loma-Prieta-Region, Kalifornien, Dissertation, LMU Munich.

[Rossum & Warsaw 2001] Rossum, G. van, and B. Warsaw, Style Guide for Python Code, http://www.python.org/dev/peps/pep-0008/ (last accessed 15 September 2009).

[Scherbaum 2007] Scherbaum, F., Of poles and zeros. Fundamentals of digital seismology, Springer Netherlands, ISBN 978-0-7923-6835-9.

[Schorlemmer et al. 2004] Schorlemmer D., A. Wyss, S. Maraini, S. Wiemer, and M. Baer, QuakeML - An XML schema for seismology, *ORFEUS Newsletter*, vol. 6, no. 2.

[Shapira 2007] Shapira, A., New station code names, http://www.isc.ac.uk/stationcode/doc/avi0706.html (last accessed 15 September 2009).

[Snoke et al. 1984] Snoke, J. A., J. W. Munsey, A. G. Teague, and G. A. Bollinger, A program for focal mechanism determination by combined use of polarity and SV-P amplitude ratio data, *Earthquake Notes*, vol. 55, no. 3, p. 15.

[Spinuso 2009] Spinuso, A., Webservices, http://neriesdataportalblog.freeflux.net/webservices/ (last accessed 15 September 2009).

[Staken 2001] Staken, K., XML Database API Draft, http://xmldb-org.sourceforge.net/xapi/xapi-draft.html (last accessed 15 September 2009).

[Stammler & Walther 2009] Stammler K., and M. Walther, Seismic Handler, http://www.seismic-handler.org (last accessed 15 September 2009).

[Stammler 1993] Stammler, K., SeismicHandler - Programmable multichannel data handler for interactice and automatic processing of seismological analyses, *Comp. Geosciences*, vol. 19, no. 2, pp. 135-140.

[Stenberg et al. 2009] Stenberg D. et al., cURL and libcurl, http://curl.haxx.se/ (last accessed 15 September 2009).

[Taylor & Harrison 2008] Taylor, I. J., and A. Harrison, From P2P to Grids to Service on the Web: Evolving Distributed Communities, Springer, ISBN 978-1-84800-122-0.

[Tsuboi et al. 2004] Tsuboi, S., J. Tromp, and D. Komatitsch, An XML-SEED Format for the Exchange of Synthetic Seismograms, *EOS Transactions of American Geophysical Union*, suppl., SF31B-03.

[Twisted 2008] The Twisted Development Team, The Twisted Documentation, http://twistedmatrix.com/projects/core/documentation/howto/book.pdf (last accessed 15 September 2009).

[USGS 2004] NEIC Web Team, USGS Fast Moment Tensor Solution, http://neic.usgs.gov/neis/eq_depot/2004/eq_041226/neic_slav_q.html (last accessed 15 September 2009).

[Van Rossum et al. 1990] Van Rossum, G. et al., Python Language Website, http://www.python.org (last accessed 15 September 2009).

[W3C 1999a] World Wide Web Consortium, XML Path Language (XPath), http://www.w3.org/TR/xpath (last accessed 15 September 2009).

[W3C 1999b] World Wide Web Consortium, XSL Transformations (XSLT), http://www.w3.org/TR/xslt (last accessed 15 September 2009).

[W3C 2001a] World Wide Web Consortium, XML Linking Language (XLink) Version 1.0, http://www.w3.org/TR/xlink/ (last accessed 15 September 2009).

[W3C 2001b] World Wide Web Consortium, Web Services Description Language (WSDL) 1.1, http://www.w3.org/TR/wsdl (last accessed 15 September 2009).

[W3C 2004a] World Wide Web Consortium, Web Services Glossary, http://www.w3.org/TR/ws-gloss/ (last accessed 15 September 2009).

[W3C 2004b] World Wide Web Consortium, XML Schema Part 0: Primer Second Edition, http://www.w3.org/TR/xmlschema-0/ (last accessed 15 September 2009).

[W3C 2004c] World Wide Web Consortium, XML Schema Part 1: Structures Second Edition, http://www.w3.org/TR/xmlschema-1/ (last accessed 15 September 2009).

[W3C 2004d] World Wide Web Consortium, XML Schema Part 2: Datatypes Second Edition, http://www.w3.org/TR/xmlschema-2/ (last accessed 15 September 2009).

[W3C 2006] World Wide Web Consortium, Namespaces in XML 1.0 (Second Edition), http://www.w3.org/TR/xml-names/ (last accessed 15 September 2009).

[W3C 2007a] World Wide Web Consortium, SOAP Version 1.2 Part 0: Primer (Second Edition), http://www.w3.org/TR/soap12-part0/ (last accessed 15 September 2009).

[W3C 2007b] World Wide Web Consortium, SOAP Version 1.2 Part 1: Messaging Framework (Second Edition), http://www.w3.org/TR/soap12-part1/ (last accessed 15 September 2009).

[W3C 2007c] World Wide Web Consortium, SOAP Version 1.2 Part 2: Adjuncts (Second Edition), http://www.w3.org/TR/soap12-part2/ (last accessed 15 September 2009).

[W3C 2007d] World Wide Web Consortium, XQuery 1.0: An XML Query Language, http://www.w3.org/TR/xquery/ (last accessed 15 September 2009).

[W3C 2008] World Wide Web Consortium, Extensible Markup Language (XML) 1.0 (Fifth Edition), http://www.w3.org/TR/xml/ (last accessed 15 September 2009).

[W3C 2009] World Wide Web Consortium, Cascading Style Sheets Level 2 Revision 1 (CSS 2.1) Specification, http://www.w3.org/TR/CSS2/ (last accessed 15 September 2009).

[WebDC 2009] WebDC Initiative, European Integrated Data Archives - EIDA, http://www.webdc.eu/webdc_arc_eu.html (last accessed 15 September 2009).

[Wei 2005] Wei, C. K., AJAX: Asynchronous Java + XML?, http://www.developer.com/design/article.php/3526681 (last accessed 15 September 2009).

[Wikipedia 2009] Wikipedia, The Free Encyclopedia, List of Web service specifications, http://en.wikipedia.org/wiki/List_of_Web_service_specifications (last accessed 15 September 2009).

[Zakšek 2008] Zakšek, K., Exupery-VFRS: Introduction, http://www.exupery-vfrs.de (last accessed 15 September 2009).

Appendix

A.1 SeisHub Installation Guide

The following section shows how to install SeisHub and associated components. It will not cover the installation of a relational database back-end, like PostgreSQL. Please refer to the manual of the preferred database.

For Linux and UNIX systems the author suggests to install SeisHub as a non-administrative user applying a new, separate, local Python 2.6.x instance.

Installing Python on a Windows operating system is more complicated because development tools like a C compiler are not part of a standard Windows distribution. Therefore many Python modules using C extensions have to be delivered as binary package with an executable installer. The fastest, most unproblematic way is to install Python and all extensions as the administrative system user.

Python

1. Download and uncompress the latest stable Python 2.6.x package for the used operating system from http://www.python.org/download/. Windows user may just use the executable installer and skip to the next subsection.
2. Run

   ```
   ./configure –prefix=$HOME
   make
   make install
   ```

3. Add `$HOME/bin` to the `PATH` environmental variable, e.g. in bash:

   ```
   export PATH="$HOME/bin:$PATH"
   ```

4. Call `python` in command line. It should show the correct version number.

Easy Install

“Easy Install” is a powerful command-line based package management tool for Python. Like CPAN for Perl, it automates the download, build, installation and update process of Python packages.

1. Download http://python-distribute.org/distribute_setup.py.
2. Run

   ```
   python distribute_setup.py
   ```

Required Python extensions

1. Run separately and check for errors.

```
easy_install SQLAlchemy
easy_install Cheetah
easy_install pycrypto
easy_install Twisted
easy_install lxml
easy_install pyOpenSSL
easy_install numpy
easy_install obspy
easy_install obspy.xseed
easy_install obspy.arclink
```

2. Windows users need to install pywin32 (Python for Windows extension). Download and install from http://sourceforge.net/projects/pywin32/.

Additional database bindings (optional)

SeisHub uses as default data back-end SQLite, which comes with Python 2.6.x. For PostgreSQL are additional database bindings required.

- For a PostgreSQL database run

```
easy_install psycopg2[69]
```

SeisHub

1. Either use the version delivered within this study or checkout the latest SeisHub version from the subversion directory via

```
svn co https://svn.geophysik.lmu.de/svn/seishub/trunk
```

2. Go into `trunk/bin` and correct the directory paths in the start and stop shell scripts.
3. Start SeisHub.

69 Requires developer packages for PostgreSQL. Binary packages for Windows OS can be found at http://www.stickpeople.com/projects/python/win-psycopg/.

A.2 Configuration File seishub.ini

Section `[seishub]`

Option	Default value	Description
`host`	`localhost`	Host IP of the server.
`log_level`	`DEBUG`	Logging level.
`min_password_length`	`8`	Minimal password length.

Section `[db]`

Option	Default value	Description
`uri`	`sqlite:///db/seishub.db`	Database URI.
`verbose`	`False`	Database verbosity.
`pool_size`	`5`	Number of connections to allow in connection pool.
`max_overflow`	`20`	Number of connections to keep open in the connection pool.

Section `[web]`

Option	Default value	Description
`autostart`	`True`	Enables HTTP/HTTPS service on start-up.
`http_port`	`8080`	HTTP port number.
`https_port`	`8443`	HTTPS port number.
`http_log_file`	`10`	HTTP access log file.
`https_log_file`	`5`	HTTPS access log file.
`https_pkey_file`	`conf/https.pkey.pem`[70]	Private key file.
`https_cert_file`	`conf/https.cert.pem`[70]	Certificate file.
`admin_theme`	`magic`	Default administration theme.
`default_pages`	`index, index.htm, index.html`	Default index pages.

70 The path delimiter within the file string depends on the operating system.

Section [sftp]

Option	Default value	Description
autostart	False	Enables SFTP service on start-up.
port	5021	SFTP port number.
public_key_file	conf/sftp.public.key[70]	Public RSA key.
private_key_file	conf/sftp.private.key[70]	Private RSA key.

Section [ssh]

Option	Default value	Description
autostart	False	Enables SSH service on start-up.
port	5001	SSH port number.
public_key_file	conf/ssh.public.key[70]	Public RSA key.
private_key_file	conf/ssh.private.key[70]	Private RSA key.

Section [manhole]

Option	Default value	Description
autostart	False	Enables Manhole service on start-up.
port	5001	Manhole port number.
public_key_file	conf/ manhole.public.key[70]	Public RSA key.
private_key_file	conf/ manhole.private.key[70]	Private RSA key.

Section [fs]

Every entry in this section will result into a own virtual directory mapped into the resource tree and therefore available via the HTTP and SFTP service. As example one would write the line "`test = /temp/test`" in order to map the physical directory "`/temp/test`" into the virtual directory "`/test`". Additional lines may be used to add further directories.

Section `[seedfilemonitor]`

Option	Default value	Description
`autostart`	`False`	Enables SEEDFileMonitor service on start-up.
`paths`	`data`	Comma separated list of file paths to scan for SEED files.
`pattern`	`*.*.*.*.*.*.*`	SEED file name pattern.
`crawler_period`	`10`	Path check interval in seconds.
`scanner_period`	`5`	File check interval in seconds.
`crawler_file_cap`	`1000`	Maximum files in watch list.
`focus_on_recent_files`	`True`	Scanner focuses on recent files.
`keep_files`	`False`	If set database entries won't be removed if file is missing.

A.3 SOAP Example

WSDL Document

Downloaded via http://almighty.pri.univie.ac.at/~mangler/helloService.wsdl.

```
<?xml version="1.0"?>
<definitions
    name="helloService"
    targetNamespace="urn:helloService"
    xmlns:typens="urn:helloService"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
    xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
    xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
    xmlns="http://schemas.xmlsoap.org/wsdl/">
  <types>
    <xsd:schema xmlns="http://www.w3.org/2001/XMLSchema"
                targetNamespace="urn:helloService"/>
  </types>
  <message name="sayHelloToRequest">
    <part name="name" type="xsd:string"/>
  </message>
  <message name="sayHelloToResponse">
    <part name="greeting" type="xsd:string"/>
  </message>
  <portType name="helloServicePort">
    <operation name="sayHelloTo">
      <input message="typens:sayHelloToRequest"/>
      <output message="typens:sayHelloToResponse"/>
    </operation>
  </portType>
  <binding name="helloServiceBinding" type="typens:helloServicePort">
    <soap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http"/>
    <operation name="sayHelloTo">
      <soap:operation soapAction="urn:helloService/sayHelloTo"/>
      <input>
        <soap:body use="encoded" namespace="urn:helloService"
                   encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
      </input>
      <output>
        <soap:body use="encoded" namespace="urn:helloService"
                   encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
      </output>
    </operation>
  </binding>
  <service name="helloServiceService">
    <port name="helloServicePort" binding="typens:helloServiceBinding">
      <soap:address
          location="http://almighty.pri.univie.ac.at/~mangler/helloService.php"/>
    </port>
  </service>
</definitions>
```

SOAP Request

Example SOAP message for RPC method “sayHelloTo” containing the string “TEST” for the Web service situated at http://almighty.pri.univie.ac.at/~mangler/helloService.php.

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<SOAP-ENV:Envelope
    xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
    xmlns:typens="urn:helloService"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
    xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
    xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <SOAP-ENV:Body>
    <mns:sayHelloTo xmlns:mns="urn:helloService"
        SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
      <name xsi:type="xsd:string">TEST</name>
    </mns:sayHelloTo>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

SOAP Response

Example SOAP response of the RPC method “sayHelloTo” retrieved from the Web service situated at http://almighty.pri.univie.ac.at/~mangler/helloService.php using the request above.

```
<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope
    xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
    xmlns:ns1="urn:helloService"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
    SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
  <SOAP-ENV:Body>
    <ns1:sayHelloToResponse>
      <greeting xsi:type="xsd:string">Hello TEST</greeting>
    </ns1:sayHelloToResponse>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

A.4 Supplementary CD-ROM

All source code developed within this thesis are included on the supplementary CD-ROM discussed below and are also publicly available at the home pages of SeisHub http://www.seishub.org and ObsPy http://www.obspy.org. The CD-ROM contains the following files and folders.

File or directory	Description
`thesis.pdf`	This thesis in Portable Document Format (PDF).
`thesis/`	All images, Python programs, documents used throughout this study, structured in subdirectories that correspond to the chapters of this thesis.
`software/obspy/`	Recent version of the ObsPy library.
`software/seishub/`	Recent version of SeisHub.

Curriculum Vitae

Curriculum Vitae

Persönliche Daten:

Name:	Robert Georg Barsch
geboren:	02. Februar 1977 in Dresden
Staatsangehörigkeit:	deutsch
Familienstand:	ledig

Schulausbildung:

1983 – 1991	Klement-Gottwald-Oberschule Dresden Reick
1991 – 1993	Fritz-Löffler-Gymnasium Dresden-Südvorstadt
1993 – 1994	Austauschschüler Highschool Ladysmith/Wisconsin, USA
Juli 1994	Amerikanisches Highschool Diploma
1994 – 1996	Abitur Fritz-Löffler-Gymnasium Dresden Südvorstadt
Juni 1996	Allgemeine Hochschulreife

Zivildienst:

1996 – 1997	Zivildienst, Diakonie Hainichen (bei Dresden)

Studium:

1997 – 2004	Studium Geophysik an der TU Bergakademie Freiberg (TUBAF)
2000 – 2004	Parallelstudium Network-Computing an der TUBAF
August 2004	Diplom im Fach Geophysik Thema: „Verifizierung der seismologischen Datenauswertung im Vergleich von alternativen Bearbeitungssystemen“

Berufstätigkeit:

seit 1994	EDV Service Ingenieurbüro Barsch & Bergmann GmbH Dresden
1999 – 2000	Hilfswissenschaftler am Institut für Geophysik, TUBAF

2000 – 2003	Hilfswissenschaftler am Institut für Informatik, TUBAF
2001 – 2004	Inhaber und Betreiber der Webhosting Firma DOYOUWEB.DE
2002 – 2003	Hilfswissenschaftler am Institut für Wirtschaftsinformatik, TUBAF
2004 - 2006	Wissenschaftlicher Mitarbeiter am Department für Geo- und Umweltwissenschaften, Geophysik, LMU München
seit Juni 2006	Doktorand am Department für Geo- und Umweltwissenschaften, Geophysik, LMU München