Master Thesis

Streaming Web-Services for Calculating Live Hydrological Derivatives

Christian Autermann
autermann@uni-muenster.de
Institute for Geoinformatics
University of Münster

April 10, 2014

Supervisors:

Prof. Dr. Edzer Pebesma
edzer.pebesma@uni-muenster.de
Institute for Geoinformatics
University of Münster

Jordan S. Read (PhD)
jread@usgs.gov
Center for Integrated Data Analysis
United States Geological Survey

Contents

1. Introduction **1**

2. Lake-Analyzer **3**

3. Foundations **4**

4. Matlab WPS **5**
- 4.1. Configuration . . . 7
- 4.2. Type Mapping . . . 9
- 4.3. Pooling . . . 9
- 4.4. License Issues . . . 9
- 4.5. Implementation . . . 11
- 4.6. Lake-Analyzer WPS . . . 11

5. Streaming WPS **12**
- 5.1. Protocol . . . 16
- 5.2. Messages . . . 19
- 5.3. Input Types . . . 25
  - 5.3.1. Streaming Inputs . . . 25
  - 5.3.2. Static Inputs . . . 27
  - 5.3.3. Reference Inputs . . . 29
  - 5.3.4. Polling inputs . . . 29
- 5.4. Dependencies . . . 31
- 5.5. Process Description . . . 33
- 5.6. WPS Specification Shortcomings . . . 34
- 5.7. Implementation . . . 34
- 5.8. Streaming Lake-Analyzer WPS . . . 35
- 5.9. Limitations . . . 35

6. Future Work **36**

7. Conclusion **37**

References **38**

A. Listings **I**

B. Source Code **X**

C. XML Namespaces **XI**

List of Tables

1. Type Mapping between Matlab and WPS Data 9

List of Figures

1. Sequence diagram of the Matlab WPS. 6
2. Four different types of processing data: (a) conventional processing, (b) streaming input data (c) streaming output data, (d) full input and output streaming (based on Foerster et al., 2012). 13
3. Sequence diagram of the playlist-based streaming enabled WPS (Foerster et al., 2012). 14
4. Sequence diagram of typical interaction pattern with a streaming enabled WPS algorithm using two distinct clients for sending and receiving data. 18
5. Sequence diagram of chaining two streaming processes using a generic mediator between the processes to translate output to input messages. 20
6. Sequence diagram of how to implement polling inputs for a streaming enabled WPS algorithm. 30
7. Example for a dependency graph consisting of two independent subgraphs. Arrow denoting a dependency between the nodes. 31
8. Possible execution/topological order of the dependency graph in Figure 7. Black arrows represent dependence to another vertex, colored arrows the execution order. 32

Listings

1. Matlab example function that represents a simple addition. 7
2. Matlab process configuration describing the function in Listing 1. 8
3. Process description generated from the configuration in Listing 2. 8
4. Example for a Streaming WPS input message. 21
5. Example for a Streaming WPS output message. 22
6. Example for a Streaming WPS output request message. 23
7. Example for a Streaming WPS stop message 24
8. Example for a Streaming WPS error message. 24
9. Example for a Streaming WPS describe message. 25
10. Example for a Streaming WPS description message. 26
11. Example for a Streaming WPS streaming inputs. 27
12. Example for a Streaming WPS static inputs. 28
13. Example for a Streaming WPS reference input. 29
14. Matlab process configuration describing the function in Listing 1. I
15. Matlab process description generated from the configuration in Listing describing the function in Listing 14. III

1. Introduction

Recent research has highlighted the relevance of lakes to global process such as the carbon cycle (Cole et al., 2007). Ecological studies on lakes have historically taken advantage of the "closed system" bounds to delineate a simplified ecosystem, but analyses that are formulated to answer societally relevant questions often must scale this single system science approach to hundreds, thousands, or millions of lakes (Downing et al., 2006). Therefore systems must be developed that can aggregate, analyze, and ultimately interpret hydrological data at large scales. Additionally, these analytical systems must be able to easily couple lake features with supporting data that define, for example, catchment properties, local climate, and anthropomorphic stressors. These data products are readily available as national coverages that can either be sampled and turned into model parameters, or turned into model drivers if they are time series products.

This work shall evaluate the existing tools (e.g. Lake Analyzer[1], see Read et al., 2011)), data models and the modeling frameworks used by USGS CIDA[2]. Modeling runs are based on online data brokers (such as the USGS's Geo Data Portal - GDP[3]) build upon Open Geospatial Consortium standards such as Catalogue Service for the Web[4] (CSW), Web Processing Service[5] (WPS), Web Map Service[6] (WMS) and Web Coverage Service[7] (WCS), but still rely on local algorithms, which comprise functionality for statistical quality assurance and quality control as well as the calculation of various metrics related to the physical state of the lakes (often linked with ecosystem function or disturbance). Building standardized and flexible infrastructure for analyzing foundational data used by domain scientists is an important challenge given legacy and heterogeneous architectures. Therefore building on the existing infrastructure and corresponding demands of the use case shall be considered.

One approach for a scalable system is to move the modeling to a web-based processing framework, which should rely on public and interoperable standards in the given use case. Web processing allows to chain data brokers with translators, models, and eventually post-hoc analysis of model runs. This chain provides specific information products to the user. Considering the amount of data (and future process scaling needs), such an analysis must be conducted in a streaming manner, i.e. the processing should start before the last chunk of data comes in, and

[1] https://github.com/GLEON/Lake-Analyzer
[2] http://cida.usgs.gov/
[3] http://cida.usgs.gov/gdp/
[4] http://www.opengeospatial.org/standards/cat
[5] http://www.opengeospatial.org/standards/wps
[6] http://www.opengeospatial.org/standards/wms
[7] http://www.opengeospatial.org/standards/wcs

the output should also be available in parts before the processing has completely finished to reduce the lag for domain users of the system. Existing approaches to this problem shall be critically evaluated.

This thesis work comprises the evaluation, design and prototypical implementation of a lake analysis chain for live sensor data. This includes the evaluation of existing datamodels (mainly CSV/TSV) and a standardized way to convert existing domain specific applications written in MatLab[8] into streaming web services (possibly WPS algorithms) in favor of the currently used non standardized web frontend[9].

Research Questions

- How can large scale hydrological data be processed in a service-based processing chain?
- Do available web-processing interface definitions support a live data streaming scenario, what is missing?
- Can real-time data be integrated into the processing chain for a constant (streamed) analysis?
- How does the developed architecture perform in practical test with 1000s and 10000s of lake features?
- How can continued statistical quality assurance and quality control in the application area of lake ecology be modeled in a web service chain?
- Do existing standards (data models and service interfaces for data warehousing, processing and visualization) support a streaming analysis chain? What is missing?
- How can spatial dependencies between streamed features be considered?
- How can a analysis language commonly used by domain experts (in this case MatLab) be easily deployed in a web based processing chain?

[8] http://www.mathworks.de/products/matlab/
[9] http://lakeanalyzer.gleon.org/

2. Lake-Analyzer

3. Foundations

4. Matlab WPS

- closed source, commercial software by The MathWorks, Inc.
- high-level language and interactive environment
- numerical computation, visualization and programming
- origins in matrix computations (*Mat*rix *Lab*oratory)
- weakly, dynamical typed language
- cross platform
- toolboxs
  - statistics
  - curve fitting
  - neural network
  - image processing
  - financial
  - bioinformatics
  - signal processing
- Matlab Central
  - user contributed toolboxes/files/functions/algorithms/etc.
  - mostly BSD licensed
- widespread in academics, engineering and industry
- offering a specialized lake-analyzer process not worth the effort
- generic solution would have huge benefit
- a lot of models implemented in Matlab can become available as WPS processes
- offering matlab programs as WPS processes should be easy procedure
- domain experts developing models in Matlab may not have extensive programming experience
- probably have no experience in developing web services or java code
- no switch in languages should be needed
- matlab functions with multiple return values
- create framework to easily expose a matlab function as an WPS process
- requirements
  - no web service programming/java programming experience should be needed
  - pooling of matlab instances to reduce latency
  - support for WPS complex inputs without dependency to a specific format
  - existing models and algorithms should be easy (and not intrusive) to expose as WPS algorithms

  - process description should be manually be written, but be easily generated
  -
- previous approaches: WPS4R
  - heavily format specific
    * parsing of GML/etc in the WPS and translation to R structures
    * configuration as comments in R scripts
    * focussing on scripts and not on functions

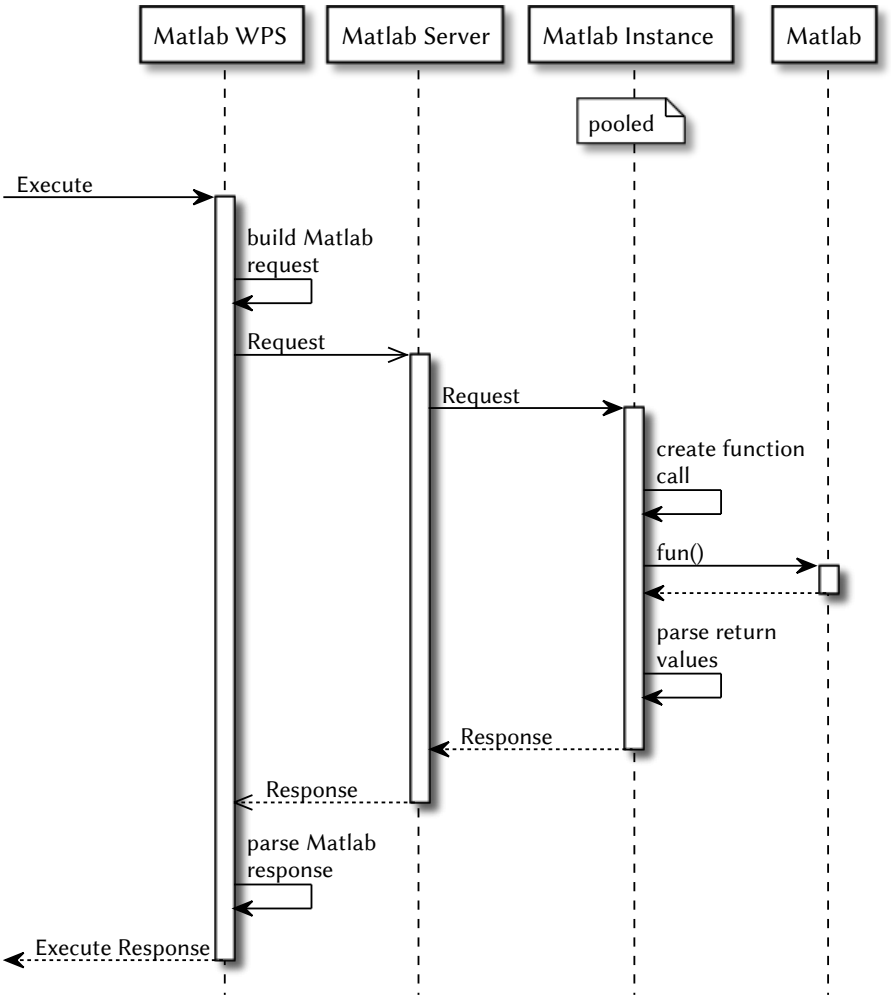

Figure 1: Sequence diagram of the Matlab WPS.

- matlab function <-> wps process
- not format specific
- no conversion of complex inputs/outputs

    - single output formats
- matlab program has to parse inputs
- easy to publish existing scripts and functions as WPS processes
- multi-tier implementation
    - Matlab WPS
        * Translates WPS Execute requests to Matlab client requests
        * Translates Matlab client responses to WPS Execute responses
        * configuration with YAML file to create description and translate inputs/outputs
    - Matlab Client
        * WebSocket client to access the Matlab server.
        * offers simple request building API
    - Matlab Server
        * WebSocket server that pools multiple Matlab Instances
        * delegates requests to free instances
    - Matlab Instance
        * a Java wrapper around a Matlab instance
    - Matlab
        * A headless instance of the Matlab software

4.1. Configuration

Listing 1: Matlab example function that represents a simple addition.

```
function result = add(a, b)
  result = a + b
end
```

- Can not be used to offer any function as process
- would not conform to Mathworks license
- configuring of a single function as a process
- configuration YAML file

Listing 2: Matlab process configuration describing the function in Listing 1.

```yaml
---
function: add
connection: local
identifier: matlab.add
version: 1.0.0
inputs:
  - identifier: a
    type: double
  - identifier: b
    type: double
outputs:
  - identifier: result
    type: double
...
```

Listing 3: Process description generated from the configuration in Listing 2 (see Appendix C for omitted XML namespaces).

```xml
<ProcessDescription wps:processVersion="1.0.0">
  <ows:Identifier>matlab.add</ows:Identifier>
  <ows:Title>matlab.add</ows:Title>
  <DataInputs>
    <Input minOccurs="1" maxOccurs="1">
      <ows:Identifier>a</ows:Identifier>
      <ows:Title>a</ows:Title>
      <LiteralData>
        <ows:DataType ows:reference="xs:double"/>
        <ows:AnyValue/>
      </LiteralData>
    </Input>
    <Input minOccurs="1" maxOccurs="1">
      <ows:Identifier>b</ows:Identifier>
      <ows:Title>b</ows:Title>
      <LiteralData>
        <ows:DataType ows:reference="xs:double"/>
        <ows:AnyValue/>
      </LiteralData>
    </Input>
  </DataInputs>
  <ProcessOutputs>
    <Output>
      <ows:Identifier>result</ows:Identifier>
      <ows:Title>result</ows:Title>
      <LiteralOutput>
        <ows:DataType ows:reference="xs:double"/>
      </LiteralOutput>
    </Output>
  </ProcessOutputs>
</ProcessDescription>
```

Table 1: Type Mapping between Matlab and WPS Data

		Matlab Type	
	Data	**For single inputs**	**For multiple inputs**
Complex	*any*	String	Cell
Bounding Box	-	-	-
Literal	xs:int	Numeric	Array
	xs:boolean	Numeric	Array
	xs:dateTime	Numeric	Array
	xs:double	Numeric	Array
	xs:float	Numeric	Array
	xs:byte	Numeric	Array
	xs:short	Numeric	Array
	xs:int	Numeric	Array
	xs:long	Numeric	Array
	xs:string	String	Cell
	xs:anyURI	String	Cell

4.2. Type Mapping

4.3. Pooling

- matlab instances are pooled
- reduced starting time of instances
- limitation of instances

4.4. License Issues

MATLAB usage is, as any software, restricted by the softwares license. MATLAB is a proprietary and commercial product and a such the software and its usage is more restricted than e.g. a open source software such as GNU R. Relevant for the MATLAB WPS is section 4.8 of *The MathWorks, Inc. Software License Agreement* (The MathWorks, Inc., 2013):

> *4. LICENSE RESTRICTIONS. The License is subject to the express restrictions set forth below. Licensee shall not, and shall not permit any Affiliate or any Third Party to: [...] 4.8. provide access (directly or indirectly) to the Programs via a web or network Application, except as permitted in Article 8 of the Deployment Addendum;*

As the MATLAB WPS offers MATLAB functionalities through a web service interface, the usage is highly restricted, as the referenced *Deployment Addendum* (The MathWorks, Inc., 2013) states:

> *8. WEB APPLICATIONS. Licensee may not provide access to an entire Program or a substantial portion of a Program by means of a web interface.*
>
> *For the Network Concurrent User Activation Type. Programs licensed under the Network Concurrent User Activation Type may be called via a web application, provided the web application does not provide access to the MATLAB command line, or any of the licensed Programs with code generation capabilities. In addition, Licensed Users may not provide access to an entire Program or a substantial portion of a Program. Such operation of an application via a web interface may be provided to an unlimited number of web browser clients, at no additional cost, for Licensee's own use for its Internal Operations, and for use by Third Parties.*
>
> *For the Network Named User and Standalone Named User Activation Types. Programs licensed under the Network Named User and Standalone Named User Activation Types may be called via a web application, provided the web application does not provide access to the MATLAB command line, or any of the licensed Programs with code generation capabilities, and such application is only accessed by designated Network Named User or Standalone Named User licensees of such Programs.*
>
> *Programs licensed under any other Activation Type may not be called via a web interface.*

Only the *Network Concurrent User Activation Type* is allowed to offer MATLAB scripts and functions as long it does not offer access to the MATLAB command line interface. *Network and Standalone Named User* license types require additional authentication mechanism in place in order to restrict the access to the web application. As the MATLAB WPS does not offer the possibility to access the MATLAB command line interface or substantial portion of MATLAB, but restricts access to configured MATLAB function calls, customers owning a license of the first type are allowed to deploy a WPS offering MATLAB processes to a open network, while users of the second class of licenses are still allowed to deploy them with an additional authentication mechanism. Using a pool of MATLAB instances on a remote server on the other hand introduce additional problems in regard of the license. In theory these MATLAB can be used to perform about any function call, and thus provide access to the MATLAB command line interface. Even though the access is restricted to simple function calls and does not allow variable declaration,

nested function calls or function definition, it may be considered a license violation the deploy this infrastructure in a public environment.

A conclusive analysis of the legal implications of the system is out of the scope of this thesis, but certainly should be done before a system facilitating the MATLAB WPS or any of its components is deployed in a public or productive environment.

4.5. Implementation

4.6. Lake-Analyzer WPS

5. Streaming WPS

In contrast to conventional data processing, such as the method used in the WPS, streaming processing approaches show considerable benefits. Regarding to time efficiency and with reference to the already mentioned problems of processing substantial large data sets or live data, the development of a streaming enabled WPS seems to be of great value.

Streaming describes the sequential processing, analysis or transfer of chunks of a datasets in contrast to the random access processing of the dataset as a whole.

- definition von stream
- definition von chunk
- definition von streaming
- definition von streaming processing
- processing takes place on small chunks instead of the complete dataset
- reduced processing resources needed to process smaller chunks
- reduced latency to see the output
- enables processing of indefinite large datasets (e.g. live analysis)
- widely known:
  - media streaming (live/on-demand) video/audio streaming
    - RTP and RTCP (Schulzrinne et al., 2003), RTSP (Schulzrinne et al., 1998), SIP (Rosenberg et al., 2002)
  - inter process communication
    - pipes/sockets (local or network) (Buschmann et al., 1996)
- Many processes accept inputs that are aggregates of smaller inputs (such as rasters and tiles, feature collections and features, etc.). Often these inputs are processed separately

Streaming processing can be divided into three categories, that differ from conventional processing (see Figure 2 (a)). Characteristic for input streaming (b) is the parallel occurrence of input and processing with a subsequent output after processing finished. On the other hand, output streaming processing describes the isolated input supply and parallel processing and output (c). Combining these two approaches results in the third category, full input and output streaming, in which input, processing and output take place concurrently. Despite their respective concurrency, all three categories have the very same advantage. By parallelizing processing and input and/or output, the overall execution and initial response time is appreciably shorter. Full input and output streaming enabled processes have the additional advantage to be able to process indefinite large datasets by processing each input data chunk separately and outputting a output data chunk for each of them. Through this the analysis of live sensor data

can be accomplished. Each of these categories of processing demands different requirements from the process or algorithm. To create a stream the dataset needs to divisible into smaller chunks, input streaming enabled algorithms need to be able to operate on each of these chunks separately and output streaming enabled processes need to be able to produce intermediate results. Input streaming would result in no benefits for algorithms requiring global knowledge of the dataset, because they can not start processing prior to all data chunks have arrived. Processes that result in a single output value, for which the processing has to be completed offer no advantage, when they are output streaming enabled.

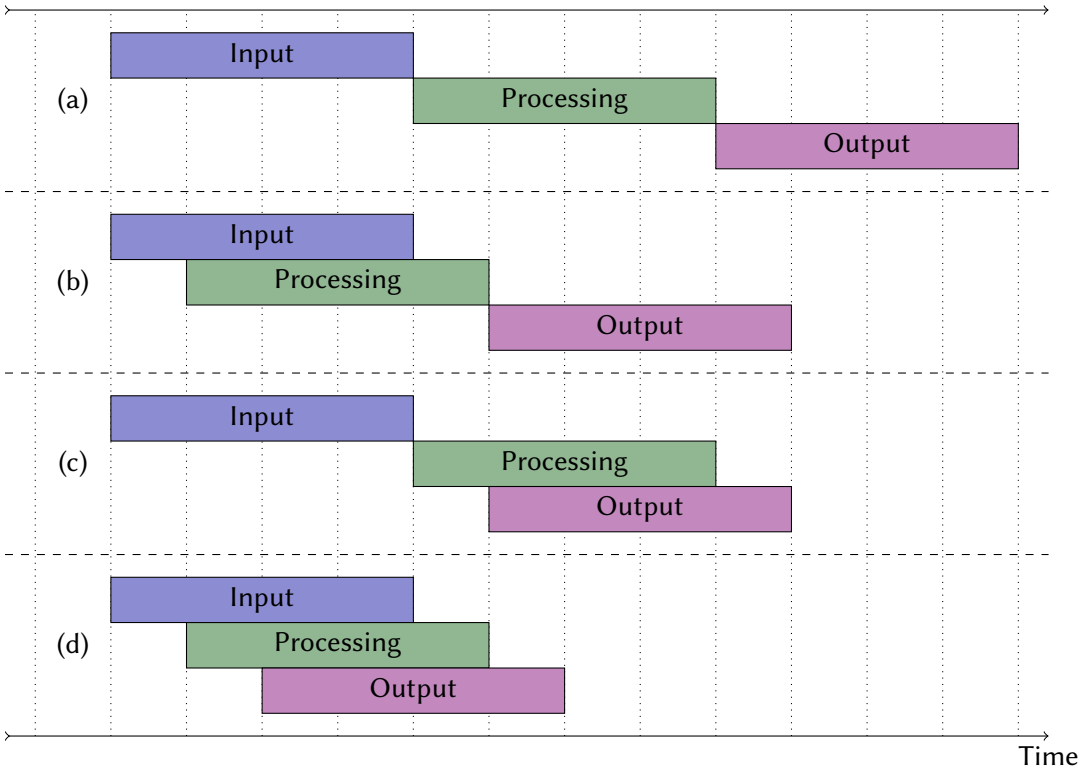

Figure 2: Four different types of processing data: (a) conventional processing, (b) streaming input data (c) streaming output data, (d) full input and output streaming (based on Foerster et al., 2012).

Previous approaches to combine the concept of streaming and web-based processing of spatio-temporal data using the WPS are drafted in strong correlation to media streaming (Foerster et al., 2012) by using playlist files (Pantos and May, 2013) as inputs and outputs of a WPS process. The process is executed asynchronously and the output playlist location is published using the <wps:ProcessStarted> element of the process status response (see Figure 3). As the WPS specification is not designed to be extensible, the elements content is restricted to a simple

string and can not contain complex Extensible Markup Language (XML) structures. Furthermore the elements definition states, that it is a “a human-readable text string whose contents are left open to definition by each WPS server, but is expected to include any messages the server may wish to let the clients know. Such information could include how much longer the process may take to execute, or any warning conditions that may have been encountered to date. The client may display this text to a human user” (Open Geospatial Consortium, 2007). Despite the goal of maintaining compatibility to WPS specification and existing software components, this represents a misappropriation of the element and will result in incompatibilities with existing WPS client solutions. Besides that, this solution is only able to transport a single playlist location to the client and thus, a WPS process may only have a single streaming output.

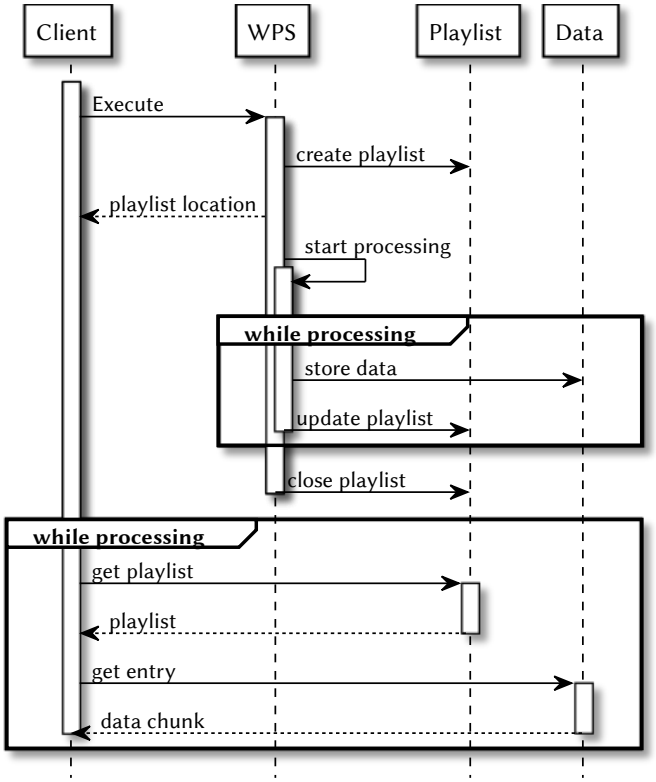

Figure 3: Sequence diagram of the playlist-based streaming enabled WPS (Foerster et al., 2012).

Input parameters may also be supplied using a playlist file. The coordination of several streaming inputs is either not possible or heavily dependent on the streaming enabled process. A process accepting two streamed datasets, that are combined during processing, has to decide which data chunks it has to combine and .... Even in the simplest case of combining chunks of

both streams with the same index can have serious implications in the use case of live analysis. If a data chunk gets lost, the process, either due to hardware or network failure, the process will combine chunks, that are not related. In continuous process, this error can not be detected, as to indefinite streams of data will always have matching indexes. Use cases in which the rate of incoming data between streams differ or data chunks depend on other chunks are very hard to model and will result in highly specialized processes, that depend not only on the structure and format of input data, but also on the data source, and thus it's incoming rate. By this, generic solutions, that convert existing WPS processes into streaming enabled processes are hard to develop, and most streaming enabled processes may not be used in contexts other that it was developed for.

Moreover, realizing streaming by continuous polling of playlists is highly inefficient. Neither can the client know the rate output data is produced nor can the WPS process know at which rate input data becomes available. By polling at a too slow rate the arrival of data chunks may be missed, which results in a slower process execution and by polling at a too high rate, network and computation resources are wasted. Adaptive polling rates may be a solution for this problem, but are useless in cases, where the rate of incoming data changes across the process execution. The usage of playlist to transport data from the client to the server, in contrast to transporting data from the server to the client, for which the origin in media streaming playlist was developed, is additionally questionable. Clients need the capability to publish files as resources, that a accessible using a URL (e.g. on a FTP or HTTP server). In a web browser environment, a JavaScript client is only able to do this using an external service, that has to store the data and maintains the playlist. A pure JavaScript browser client is not able to use streaming inputs in this playlist-based streaming WPS approach. The implementation of this approach is additionally limited. Input parameter data streams are not implemented and process implementations have to split inputs to create output streams (see Figure 2 (c)). Splitting spatio-temporal data into smaller chunks is not as trivial as e.g. splitting a audio or video stream in single frames and so the process implementations become heavily format dependent and dependencies between data chunk can only expressed as part of the data, in a format, that the process is able to understand and to handle. Also this approach requires a reimplementation of already existing processes to achieve streaming outputs.

A streaming enabled WPS should extend the traditional processing paradigm (see Figure 2 (a)) to enable input only streaming (Figure 2 (b)), output only streaming (Figure 2 (c)), and full input/output streaming (Figure 2 (d)), for which input parameters are supplied subsequently and output data chunks is published as it becomes available. To accomplish this, it should not rely on inefficient polling techniques, in which the server or client is requesting a resource

continuously over time, but should rely on true streaming technologies, that offer a full-duplex communication channel between client and server. Streaming enabled process should be accessible from the same environments as conventional WPS processes. This especially includes web browser environments, that are particular restricted in their possibilities. A streaming enabled WPS process should rely on existing widely known and standardized technologies, it should be especially as interoperable as possible to the WPS specification, but should not compromise streaming functionality by enforcing incompatible standards. As spatio-temporal and it's processing and analysis often can not be treated independent to surrounding data, dependencies between streamed data chunks have to be considered. This will require the streaming enabled process to be able not only to operate on sequential data but also be able to allow, to some degree, random access to the data. Despite handling of dependencies between spatio-temporal features should be considered, processes and algorithms, that require global knowledge of the dataset may not profit from a streaming enabled WPS and should not be considered relevant for a streaming enabled WPS. The system should be as generic as the existing WPS specification, so it should not rely on specific data formats and allow easy chaining of streaming processes. As possible use cases include not only live analysis of data, but also the processing of large dataset, data chunks should be processed in parallel if possible. As this may result in a undefined order of outputted data chunks, client need to be able to correlate output data chunks with the input parameter chunks. Existing WPS processes should be easily converted to streaming enabled processes, without the need to develop them from scratch.

The following sections should introduce a approach for a Streaming WPS, that will fulfill the above requirements. As seen in previous approaches the constraints imposed by the WPS specification are too strict to implement a streaming enabled WPS fulfilling the requirements, that is compatible to the standard. Previous solutions compromised functionality for sake of (incomplete) compatibility with the inflexible standard. In order to enable true, browser compatible, streaming this approach will break out of the constraining WPS standard and develop a message based architecture using WebSockets to accomplish true full-duplex streaming of data while reusing terminology and technology specified by the WPS specification.

5.1. Protocol

As the the WPS specification is not flexible enough to model a full streaming scenario, the WPS has to be bypassed. For this a more flexible interaction model was developed, that extends the conventional processing approach. This protocol is message based and enables full-duplex stream processing of spatio-temporal data. A *streaming enabled algorithm* is a WPS algorithm

that supports the here defined protocol while a *streaming process* is the identifiable instance of an algorithm, created by executing the streaming enabled algorithm using the WPS Execute operation. The streaming process is the core of the Streaming WPS and receives subsequent inputs and will emit intermediate results. While the execution of the streaming enabled algorithm is fully supported by the WPS specification, all interaction with the streaming process is not part of the standard. To communicate with the streaming process, the client needs information on how to connect to the process. As the WPS specification does not allow subsequent outputs, the call of the Execute operation will return immediately to transport this information to the client, and can not persist over the lifetime of the streaming process.

To enable a full duplex communication with the streaming process WebSockets will be used to transport messages. This is needed to *push* messages to clients instead of letting the clients constantly request updates.

The detailed interaction protocol is depicted in Figure 4. A client (*Sender*) issues a Execute to a streaming enabled WPS algorithm (step 1). The algorithm will instantiate a delegate (step 2), that is responsible for processing data chunks, and a streaming process (step 3), that is responsible for client interactions and task scheduling. The Execute response will contain the necessary details to connect to the streaming processes, such as the the identifier of the streaming process and the WebSocket endpoint URL (step 4).

With these details a client can connect directly to the streaming process bypassing the WPS interface. In step 5 another client[10] (*Receiver*) connects to the streaming process and subscribes to the future outputs of the process. By this the client does not need to constantly issue requests to the streaming process to check for new outputs, but will receive outputs automatically as long as the receiving client stays connected using the WebSocket. After this one or multiple clients start sending chunks of data as input parameters to the streaming process (step 6). The clients may open a new connection for every input or use the same connection over the lifetime of the streaming process. The streaming process will check the inputs for validity (step 7) and will queue them for processing (step 8). Processing takes places asynchronously in parallel manner and there is no guarantee of order (besides restrictions imposed by dependencies, see sections 5.3.3 and 5.4). When the are free capacities to process the data and all other requirements are met, the delegate will tasked to process the data (step 9). The delegate implementation can return a intermediate result in step 10, which will be forwarded to all registered receivers in step 11. Steps 6 to 11 may be repeated indefinitely (e.g. live analysis of data) or until the sending client has no more inputs to feed. As the streaming process would wait in this case for ever (or

[10]Even though sender and receiver are two different entities in this diagram, there are no restrictions imposed to the amount of clients, either senders or receivers, or their nature (senders may also be receivers).

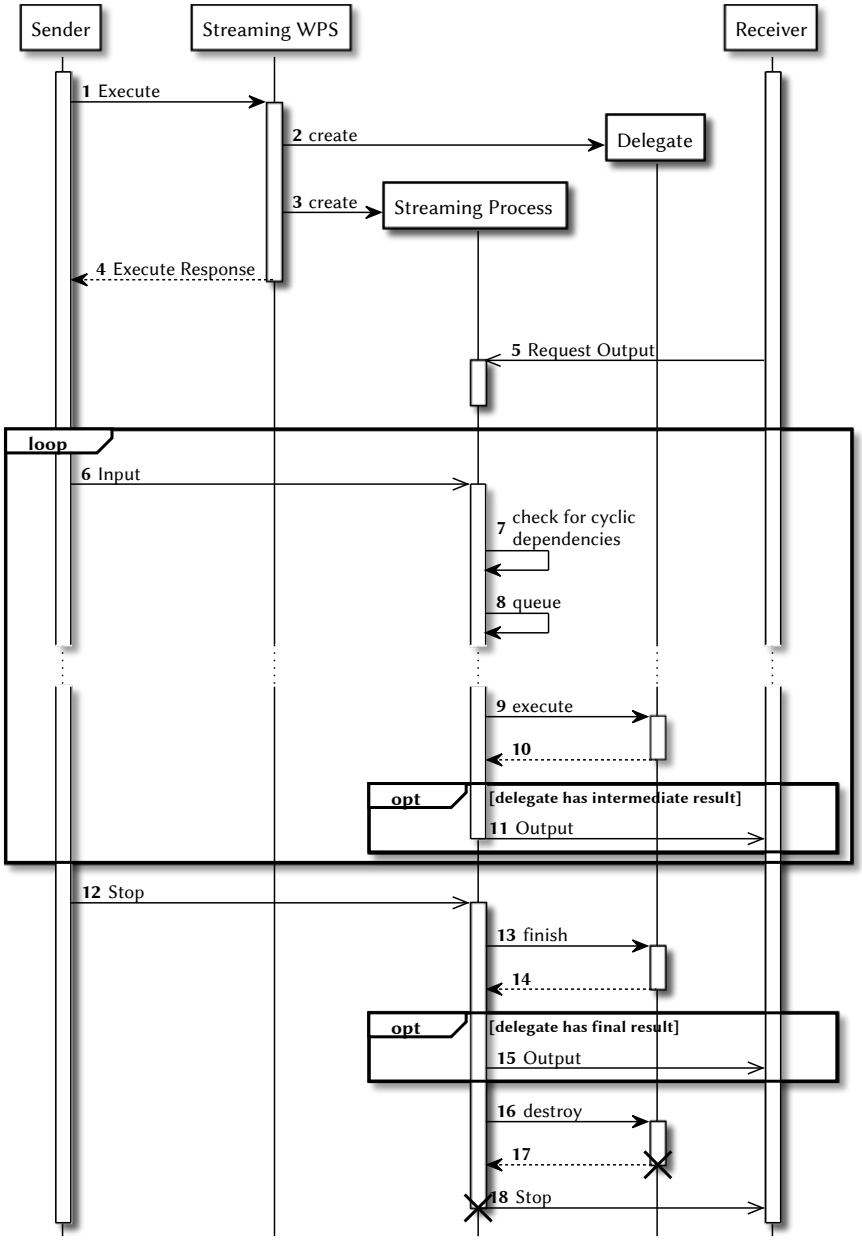

Figure 4: Sequence diagram of typical interaction pattern with a streaming enabled WPS algorithm using two distinct clients for sending and receiving data.

at least until some timeout interferes), the client has to stop the streaming process explicitly (step 12). This will cause the streaming process to stop accepting inputs, to process all not yet processed inputs and to request a last potential output from the delegate (step 13 & 14), which will be forwarded to all listening clients (step 15). After this it will destruct the delegate (steps

16 & 17) and will notify all registered listeners, that there will be no further outputs become available by publish forwarding the stop message (step 18). The streaming process will destroy itself after this.

A detailed description of the various messages of this protocol can be found in section 5.2.

The protocol permits various streaming usage scenarios. A delegate, that produces a output for every input message creates a full input/output streaming process (see Figure 2 (d)), a delegate that produces only a final output results in a input only streaming process (see Figure 2 (b)). By suppling a single input message and repeating step 11, a suitable delegate may create a output streaming process (see Figure 2 (c)) and, although not reasonable, even the traditional processing approach depicted in Figure 2 (a) can be simulated by passing all inputs in a single input message and producing a single output message.

Using message provoked streaming iterations (the combination of a input message, its processing and (optional) output message) allows the use of multiple streaming inputs and outputs. In contrast to previous approaches it is possible for the streaming process to relate these to a single processing iteration without any knowledge of their semantics, because the client encapsulates them in a single message.

The protocol also enables the chaining of processing steps. This can be realized in two ways: a delegate itself may represent a WPS process chain and thus chain every processing step or several streaming process are chained itself. A simple mediator translating input messages to output messages (see Figure 5). This mediator can be realized using a dedicated streaming enabled algorithm accepting a input/output mapping and the connection parameters of the streaming processes to connect. After requesting the outputs of the source streaming process it can translate every output message to an input message and forward the stop message. A receiving client will simply connect to the second streaming process and will received the data process by the chain. By requesting the outputs first streaming process even intermediate result of the chain are accessible.

5.2. Messages

To fulfill the above defined protocol several messages have to be exchanged between sender, streaming process and receiver. In order to correlate input and outputs or to show the source of an error, the message format has to have a concept of message references. WebSockets do not have such a concept as it is only a thin layer on top of TCP, that introduces handshake and addressing mechanism to be compatible with HTTP and a minimal framing of messages.

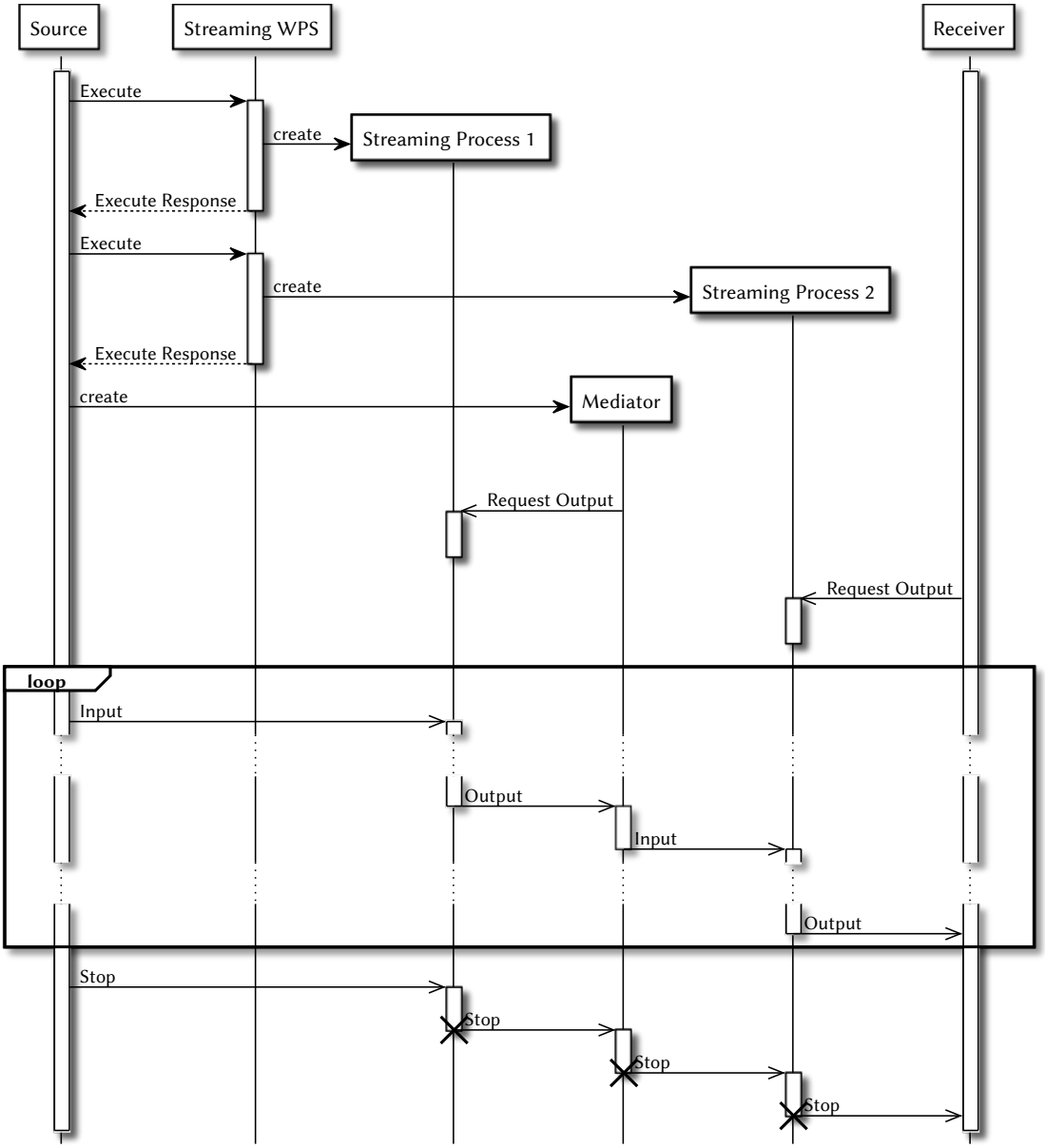

Figure 5: Sequence diagram of chaining two streaming processes using a generic mediator between the processes to translate output to input messages.

This framing is merely needed to establish a message-based instead of a stream-based protocol, as the latter would make it hard to differentiate between individual messages (Fette and Melnikov, 2011). To enable referencing of messages, and by this a asynchronous reply mechanism, another layer is needed. As the WPS is mostly based on XML, the message format should also be XML based. This enables the usage of large parts of the WPS schema and allows the reuse

of many components written to interact with the WPS.

The widely known SOAP protocol (Lafon et al., 2007), which may also be used as an optional binding of the WPS and thus can be easily adopted, is a ideal candidate for this. In combination with Web Services Addressing (WSA) (Rogers et al., 2006) it creates a XML based message framework, that allows asynchronous requests and responses over a arbitrary protocol. Besides introducing a concept of addressing and routing of messages (that will not be used in the Streaming WPS), one can assign a globally unique identifier to any message using WSA, that can be referenced with arbitrary semantics (e.g. reply).

The Streaming WPS defines seven SOAP messages.

Input Message Input messages are used by clients to supply subsequent inputs to a streaming iteration of a streaming process. They loosely resemble a WPS Execute request by consisting of any number of inputs and a identifier, which references the streaming process to which the inputs should be supplied. An example can be seen in Listing 4, possible inputs can be seen in section 5.3.

Listing 4: Example for a Streaming WPS input message (see Appendix C for omitted XML namespaces).

```
<soap:Envelope>
  <soap:Header>
    <wsa:RelatesTo RelationshipType="https://github.com/autermann/streaming-wps/needs">uuid:f31da315-
        bce3-4e26-8112-3ccf0ecf1ab5</wsa:RelatesTo>
    <wsa:MessageID>uuid:6a0e50c7-85c4-448c-962d-894c41c441bf</wsa:MessageID>
    <wsa:Action>https://github.com/autermann/streaming-wps/input</wsa:Action>
  </soap:Header>
  <soap:Body>
    <stream:InputMessage>
      <stream:ProcessID>uuid:c99e6f21-f0a0-4770-9615-db3501490f0a</stream:ProcessID>
      <stream:Inputs>
        [...]
      </stream:Inputs>
    </stream:InputMessage>
  </soap:Body>
</soap:Envelope>
```

Output Messages Output messages are used by the streaming process to transport intermediate results at the end of a streaming iteration or a final result at the end of the streaming process to listening clients. They loosely resemble a WPS Execute response by containing a arbitrary number of outputs and the identifier of the process, that produced the outputs. Output

messages containing intermediate result are replies to their corresponding input message and reference them using WSA. If the processing used the output of any other streaming iteration (see sections 5.3.3 and 5.4) the corresponding output messages are also referenced. An example can be seen in Listing 5.

Listing 5: Example for a Streaming WPS output message (see Appendix C for omitted XML namespaces).

```
<soap:Envelope>
  <soap:Header>
    <wsa:MessageID>uuid:ef9676f0-13b1-473b-a783-8fed8cbd6513</wsa:MessageID>
    <wsa:RelatesTo>uuid:6a0e50c7-85c4-448c-962d-894c41c441bf</wsa:RelatesTo>
    <wsa:RelatesTo RelationshipType="https://github.com/autermann/streaming-wps/used">uuid:cf19d698-
        f288-477b-a4ff-39611b46920e</wsa:RelatesTo>
    <wsa:Action>https://github.com/autermann/streaming-wps/output</wsa:Action>
  </soap:Header>
  <soap:Body>
    <stream:OutputMessage>
      <stream:ProcessID>uuid:c99e6f21-f0a0-4770-9615-db3501490f0a</stream:ProcessID>
      <stream:Outputs>
        <stream:Output>
          <ows:Identifier>output1</ows:Identifier>
          <wps:Data>
            <wps:LiteralData dataType="xs:string">input1</wps:LiteralData>
          </wps:Data>
        </stream:Output>
        <stream:Output>
          <ows:Identifier>output2</ows:Identifier>
          <wps:Data>
            <wps:ComplexData mimeType="application/xml" encoding="UTF-8">
              <hello>world</hello>
            </wps:ComplexData>
          </wps:Data>
        </stream:Output>
        <stream:Output>
          <ows:Identifier>output3</ows:Identifier>
          <wps:Data>
            <wps:BoundingBoxData crs="EPSG:4326" dimensions="2">
              <ows:LowerCorner>52.2 7.0</ows:LowerCorner>
              <ows:UpperCorner>55.2 15.0</ows:UpperCorner>
            </wps:BoundingBoxData>
          </wps:Data>
        </stream:Output>
      </stream:Outputs>
    </stream:OutputMessage>
  </soap:Body>
</soap:Envelope>
```

Output Request Message A output request message is used by client to let a streaming process know, that it would like to receive outputs from the process. There is no direct counter part in the WPS specification but the concept is similar to the continuous request of the WPS

response during a asynchronous process execution. As WebSockets offer a full-duplex messaging channel a continuous polling of outputs is not needed, but the streaming process can push outputs directly to listening clients. To initialize this listening the client register to one or more streaming processes using their corresponding identifiers. An example can be seen in Listing 6.

Listing 6: Example for a Streaming WPS output request message (see Appendix C for omitted XML namespaces).

```
<soap:Envelope>
  <soap:Header>
    <wsa:MessageID>uuid:950a3380-1de4-4634-ba2d-ffdf324157d7</wsa:MessageID>
    <wsa:Action>https://github.com/autermann/streaming-wps/request-output</wsa:Action>
  </soap:Header>
  <soap:Body>
    <stream:OutputRequestMessage>
      <stream:ProcessID>uuid:c99e6f21-f0a0-4770-9615-db3501490f0a</stream:ProcessID>
    </stream:OutputRequestMessage>
  </soap:Body>
</soap:Envelope>
```

Stop Message As streaming process can run indefinitely long, input supplying clients need to be able to let the streaming process know, that there will be no further inputs become available. To achieve this a stop message (see Listing 7) is send to the streaming process. The process will propagate the stop message to all listening clients to let them know there will be no further outputs. Before the stop message is propagated all streaming iterations, that are not yet processed will be finished but the process will not accept any further inputs. If there are still unresolved dependencies (see sections 5.3.3 and 5.4) the streaming process will fail with an error message.

Error Message Errors are transported, as in the WPS specification, using OGC Web Services Common (OWS) exception reports (Open Geospatial Consortium, 2007). If the delegate of a process fails or a supplied input message can not be processed due to whatever conditions, the error is propagated to listening clients. The error is always send to the client that send the message causing the error (if the client is still connected) and in case the error is caused during the execution of a streaming iteration also to all listening clients, that registered through a output request message. In contrast to failures during input validation, due to constraints imposed by dependencies (see sections 5.3.3 and 5.4), errors raised during the execution of a streaming iteration can not be compensated, but will stop the streaming process. The causing

Listing 7: Example for a Streaming WPS stop message (see Appendix C for omitted XML namespaces).

```
<soap:Envelope>
  <soap:Header>
    <wsa:MessageID>uuid:01ea8dab-5da9-46eb-81b4-06dcea32ca01</wsa:MessageID>
    <wsa:Action>https://github.com/autermann/streaming-wps/stop</wsa:Action>
  </soap:Header>
  <soap:Body>
    <stream:StopMessage>
      <stream:ProcessID>uuid:c99e6f21-f0a0-4770-9615-db3501490f0a</stream:ProcessID>
    </stream:StopMessage>
  </soap:Body>
</soap:Envelope>
```

message of a failure may obtained from the reply relation encoded using WSA. An example of an error message can be found in Listing 8.

Listing 8: Example for a Streaming WPS error message (see Appendix C for omitted XML namespaces).

```
<soap:Envelope>
    <soap:Header>
        <wsa:RelatesTo>uuid:6a0e50c7-85c4-448c-962d-894c41c441bf</wsa:RelatesTo>
        <wsa:MessageID>uuid:dc640a0a-d505-4591-baea-2a556412237e</wsa:MessageID>
        <wsa:Action>https://github.com/autermann/streaming-wps/error</wsa:Action>
    </soap:Header>
    <soap:Body>
        <stream:ErrorMessage>
            <stream:ProcessID>uuid:c99e6f21-f0a0-4770-9615-db3501490f0a</stream:ProcessID>
            <ows:Exception exceptionCode="RemoteComputationError">
                <ows:ExceptionText>Remote computation failed</ows:ExceptionText>
            </ows:Exception>
        </stream:ErrorMessage>
    </soap:Body>
</soap:Envelope>
```

Describe & Description Message Describe messages are directly adopted from the WPS Describe Process operation. Due to conditions described in section 5.5 a client needs to able to retrieve a description from a running streaming process. The message simply contains the identifier of the process the clients wants to have the description from (an example can be seen in Listing 9). The reply resembles a Describe Process response and is encoded in a description message referencing the describe message and containing the streaming process description and (see Listing 10).

Listing 9: Example for a Streaming WPS describe message (see Appendix C for omitted XML namespaces).

```
<soap:Envelope>
  <soap:Header>
    <wsa:MessageID>uuid:9ca0ed4a-0e24-4843-bb81-da2af3e23d8c</wsa:MessageID>
    <wsa:Action>https://github.com/autermann/streaming-wps/describe</wsa:Action>
  </soap:Header>
  <soap:Body>
    <stream:DescribeMessage>
      <stream:ProcessID>uuid:c99e6f21-f0a0-4770-9615-db3501490f0a</stream:ProcessID>
    </stream:DescribeMessage>
  </soap:Body>
</soap:Envelope>
```

5.3. Input Types

The aforementioned requirements imply three different types of input for a Streaming Process. They differ in the aspect of time (*When are they supplied?*) and scope (*Where are they used?*). Besides that all of them are based on the very same input types the WPS standard defines:

Complex Input Complex data structures that can be described by a mime type, an encoding and a schema. They can represent raster data, XML structures such as GML feature collections, CSV or any type of data. This data can be supplied inline or as reference to an external HTTP resource.

Literal Input Data that can be represented by a single string value and can be described by data type and a unit of measurement.

Bounding Box Input Data that represents a multi dimensional bounding box with a associated coordinate reference system.

5.3.1. Streaming Inputs

The first and most obvious type of input are streaming inputs. They are provided for a single streaming iteration and will only be used in that iteration representing the core of streaming enabled processing (see Listing 11).

Listing 10: Example for a Streaming WPS description message (see Appendix C for omitted XML namespaces).

```
<soap:Envelope>
  <soap:Header>
    <wsa:RelatesTo>uuid:9ca0ed4a-0e24-4843-bb81-da2af3e23d8c</wsa:RelatesTo>
    <wsa:MessageID>uuid:5ba3d87b-85d0-47eb-9dac-57cf193abd06</wsa:MessageID>
    <wsa:Action>https://github.com/autermann/streaming-wps/description</wsa:Action>
  </soap:Header>
  <soap:Body>
    <stream:DescriptionMessage>
      <stream:ProcessID>uuid:f7683417-ab11-4317-a833-d73aa443443d</stream:ProcessID>
      <stream:StreamingProcessDescription wps:processVersion="1.0.0"
          finalResult="false" intermediateResults="false"
          statusSupported="false" storeSupported="true">
        <ows:Identifier>uuid:c99e6f21-f0a0-4770-9615-db3501490f0a</ows:Identifier>
        <ows:Title>com.github.autermann.wps.streaming.example.AddAlgorithm</ows:Title>
        <DataInputs>
          <Input maxOccurs="1" minOccurs="1">
            <ows:Identifier>a</ows:Identifier>
            <ows:Title>a</ows:Title>
            <LiteralData>
              <ows:DataType ows:reference="xs:long"/>
              <ows:AnyValue/>
            </LiteralData>
          </Input>
          <Input maxOccurs="1" minOccurs="1">
            <ows:Identifier>b</ows:Identifier>
            <ows:Title>b</ows:Title>
            <LiteralData>
              <ows:DataType ows:reference="xs:long"/>
              <ows:AnyValue/>
            </LiteralData>
          </Input>
        </DataInputs>
        <ProcessOutputs>
          <Output>
            <ows:Identifier>result</ows:Identifier>
            <ows:Title>result</ows:Title>
            <LiteralOutput>
              <ows:DataType ows:reference="xs:long"/>
            </LiteralOutput>
          </Output>
        </ProcessOutputs>
      </stream:StreamingProcessDescription>
    </stream:DescriptionMessage>
  </soap:Body>
</soap:Envelope>
```

A conventional algorithm to compute the histogram of a raster (e.g. a satellite image) needs the complete raster as a single complex input for processing. A streaming enabled variant would split the raster in several smaller tiles and supply each of them in a single input message to the streaming process. The algorithm can process each tile on it's own and update the global histogram. Besides that the process does not have to store the complete raster, it is also able to

Listing 11: Example for a Streaming WPS streaming inputs (see Appendix C for omitted XML namespaces).

```
<stream:Inputs>
  <stream:StreamingInput>
    <ows:Identifier>intput1</ows:Identifier>
    <wps:Data>
      <wps:LiteralData dataType="xs:string">input1</wps:LiteralData>
    </wps:Data>
  </stream:StreamingInput>
  <stream:StreamingInput>
    <ows:Identifier>intput2</ows:Identifier>
    <wps:Data>
      <wps:ComplexData mimeType="application/xml" encoding="UTF-8">
        <hello>world</hello>
      </wps:ComplexData>
    </wps:Data>
  </stream:StreamingInput>
  <stream:StreamingInput>
    <ows:Identifier>intput3</ows:Identifier>
    <wps:Data>
      <wps:BoundingBoxData>
        <wps:BoundingBoxData crs="EPSG:4326" dimensions="2">
          <ows:LowerCorner>52.2 7.0</ows:LowerCorner>
          <ows:UpperCorner>55.2 15.0</ows:UpperCorner>
        </wps:BoundingBoxData>
      </wps:BoundingBoxData>
    </wps:Data>
  </stream:StreamingInput>
  <stream:StreamingInput>
    <ows:Identifier>input4</ows:Identifier>
    <wps:Reference mimeType="application/xml" encoding="UTF-8" schema="http://schemas.opengis.net/gml
        /3.1.1/base/gml.xsd" xlink:href="http://geoprocessing.demo.52north.org:8080/geoserver/wfs?
        service=WFS&amp;version=1.0.0&amp;request=GetFeature&amp;typeName=topp:tasmania_roads&amp;srs=
        EPSG:4326&amp;outputFormat=GML3"/>
  </stream:StreamingInput>
</stream:Inputs>
```

output intermediate histograms to the client.

5.3.2. Static Inputs

Algorithms that operate on a streaming input often need inputs that are common to every iteration. It would be redundant and inefficient to transfer inputs like configuration parameters in every input message for every streaming iteration. For this, the concept of static inputs needs to be introduced. Static inputs are parameters that are supplied when a streaming process is created and apply to every streaming iteration (see Listing 12). While the streaming process handles a streaming iteration, the static inputs are merged with the inputs of the causing input message and transparently supplied to the process's delegate. This way a conventional process

can be easily converted into a streaming enabled process.

Listing 12: Example for a Streaming WPS static inputs (see Appendix C for omitted XML namespaces).

```
<stream:StaticInputs>
  <wps:Input>
    <ows:Identifier>input1</ows:Identifier>
    <wps:Data>
      <wps:LiteralData dataType="xs:string">input1</wps:LiteralData>
    </wps:Data>
  </wps:Input>
  <wps:Input>
    <ows:Identifier>input2</ows:Identifier>
    <wps:Data>
      <wps:ComplexData mimeType="application/xml" encoding="UTF-8">
        <hello>world</hello>
      </wps:ComplexData>
    </wps:Data>
  </wps:Input>
  <wps:Input>
    <ows:Identifier>input3</ows:Identifier>
    <wps:Data>
      <wps:BoundingBoxData>
        <wps:BoundingBoxData crs="EPSG:4326" dimensions="2">
          <ows:LowerCorner>52.2 7.0</ows:LowerCorner>
          <ows:UpperCorner>55.2 15.0</ows:UpperCorner>
        </wps:BoundingBoxData>
      </wps:BoundingBoxData>
    </wps:Data>
  </wps:Input>
  <wps:Input>
    <ows:Identifier>input4</ows:Identifier>
    <wps:Reference mimeType="application/xml" encoding="UTF-8" schema="http://schemas.opengis.net/gml
        /3.1.1/base/gml.xsd" xlink:href="http://geoprocessing.demo.52north.org:8080/geoserver/wfs?
        service=WFS&amp;version=1.0.0&amp;request=GetFeature&amp;typeName=topp:tasmania_roads&amp;srs=
        EPSG:4326&amp;outputFormat=GML3"/>
  </wps:Input>
</stream:StaticInputs>
```

For example, a traditional process implementation of the Douglas–Peucker algorithm (Douglas and Peucker, 1973) would require a feature collection and a ϵ value as inputs. In a streaming environment, one would model the ϵ input as a static input supplied at process creation and stream the feature collection as single features in streaming inputs. Other examples are a coordinate transformation process, that accepts a feature collection and a target coordinate reference system (CRS) or a buffer algorithm that accepts a feature collection and a buffer size. Buffer size and CRS would be supplied as static inputs and the feature collection would be split into several streaming inputs and supplied in independent streaming iterations.

5.3.3. Reference Inputs

While streaming offers no real benefit to algorithms that require global knowledge of the data set, there are often cases where algorithms only require knowledge about few other chunks of the dataset or even only about the result of their processing. To model these dependencies between streaming iterations, reference inputs can be used (see Listing 13). These reference the output of another, previous or upcoming, iteration as an input parameter. Reference inputs break out of the conventional non-random access paradigm of streaming and allow a semi-random access processing of a data set. Inputs are described by referencing the corresponding output identifier and the input message that has or will produce the output data. The order of incoming input messages is irrelevant to the use of reference inputs, as input messages referencing not yet available outputs will be delayed until they can processed (see section 5.4).

Listing 13: Example for a Streaming WPS reference input (see Appendix C for omitted XML namespaces).

```
<stream:Inputs>
  <stream:ReferenceInput>
    <ows:Identifier>input3</ows:Identifier>
    <stream:Reference>
      <wsa:MessageID>uuid:f31da315-bce3-4e26-8112-3ccf0ecf1ab5</wsa:MessageID>
      <stream:Output>output1</stream:Output>
    </stream:Reference>
  </stream:ReferenceInput>
</stream:Inputs>
```

A conventional algorithm to analyze a river system, in which each processing of a river depends on the processing results of the rivers flowing into it, the complete river system data set would be supplied as a single input parameter. In a streaming enabled process, each river would be supplied as a streaming input. The output of the rivers a river depends on would be supplied as additional reference inputs.

5.3.4. Polling inputs

The last category of possible input types for a streaming WPS are polling inputs. These inputs are continuously polled from an external resource and a new streaming iteration would be started, when new inputs become available. Polling inputs would be supplied at process creation time and would contain a reference to an external resource, that is requested continuously. To not miss inputs, when they become available a playlist file, as described in previous

approaches (Foerster et al., 2012) would be need. The implementation of polling inputs as part of this streaming WPS specification would present the very same issues, that were criticized in previous approaches. How one can define the polling frequency used to retrieve the playlist, how can multiple polling inputs be declared, and how would they be combined by the streaming WPS? For this reason the Streaming WPS will not implement polling inputs. These input types are by far better handled on client side, as the client typical knows of the rate data becomes available and so can choose a appropriate polling frequency and also is able to coordinate multiple polling inputs by having a deeper understanding of their affiliation. Polling inputs could be implemented as shown in Figure 6: the client polls a data provider (e.g. a Sensor Observation Service[11] (SOS)) to check if new data is available and convert this data into a streaming input for the Streaming WPS.

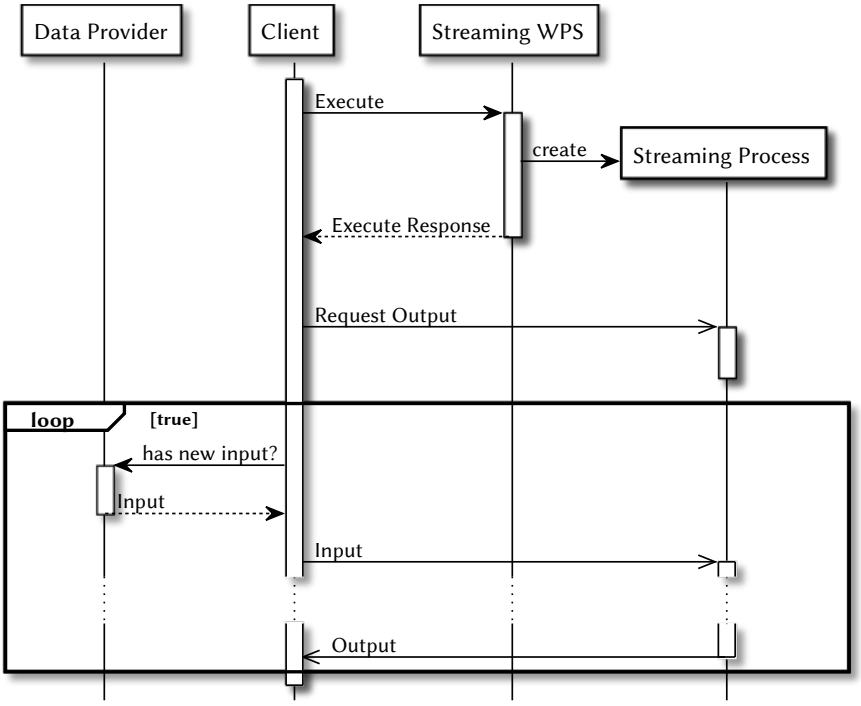

Figure 6: Sequence diagram of how to implement polling inputs for a streaming enabled WPS algorithm.

[11]http://www.opengeospatial.org/standards/sos

5.4. Dependencies

The definition of Reference Inputs in Section 5.3.3 implies a mechanism to resolve dependencies and to order the execution of streaming iterations. These are considered as tasks and can declare dependencies to other streaming iterations either by mapping an input to the output of another streaming iteration or by declaring a explicit dependency on another streaming iteration.

Dependencies can be best modeled using a Directed Acyclic Graph (DAG). A DAG is a structure $D = (V, E)$ consisting of a set of vertices (or nodes) V and edges (or arcs) E where every edge $e \in E$ is a ordered pair $v_1 \rightarrow v_2$ with $v_1, v_2 \in V$. The distinct vertices $v_1, \ldots, v_n \in V$ are called a path if for all successive vertices v_i, v_{i+1} exists a edge $v_i \rightarrow v_{i+1} \in E$. A directed graph is called acyclic if there exists no path in G with $v_1 = v_n$. A subgraph of a graph is the graph $G' = (V', E')$ with $V' \subseteq V$ and $E' = \{v_1 \rightarrow v_2 \in E | v_1, v_2 \in V'\}$. Two subgraphs $G_1 = (V_1, E_1), G_2 = (V_2, E_2)$ are independent if $V_1 \cap V_2 = \emptyset$ and there exists no edge $v_1 \rightarrow v_2 \in E$ with $v_1 \in V_1 \wedge v_2 \in V_2$ or $v_2 \in V_1 \wedge v_1 \in V_2$.

In a dependency graph, vertices represent a task, package or other entity that has dependencies and edges represent these dependencies (v_1 depends on v_2). Dependency graphs have to be acyclic as a cycle would introduce a cyclic dependency, that can not be resolved.

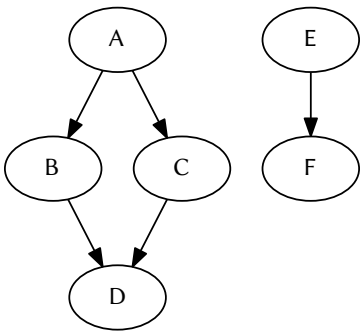

Figure 7: Example for a dependency graph consisting of two independent subgraphs. Arrow denoting a dependency between the nodes.

A system containing the tasks A, B, C, D, E, F and the dependencies $A \rightarrow B, A \rightarrow C, B \rightarrow D, C \rightarrow D$ and $E \rightarrow F$ will result in a DAG consisting of two independent subgraphs (see Figure 7).

The execution order of a dependency graph can be derived from the topological ordering of the graph: a "topological ordering, ord_D, of a directed acyclic graph $D = (V, E)$ maps each vertex to a priority value such that $ord_D(x) < ord_D(y)$ holds for all edges $x \to y \in E$" (Pearce and Kelly, 2007), a possible execution order is the list of all vertices sorted by descending ord_D. The topological order of a DAG can be computed using e.g. Breadth-first search (BFS) in linear time (Cormen et al., 2001). In most cases the topological ordering is not unique, Figure 8 shows one possible execution order for the before mentioned graph.

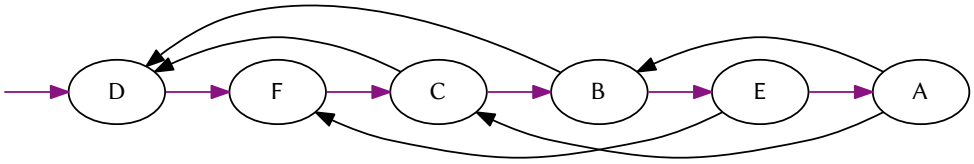

Figure 8: Possible execution/topological order of the dependency graph in Figure 7. Black arrows represent dependence to another vertex, colored arrows the execution order.

In contrast to conventional dependency systems like package managers the Streaming WPS can not operate on a static graph of dependencies but on a graph to which vertices and edges are added constantly. Conventional topological sorting algorithms have to recompute the ordering for every insertion from scratch which will have a big performance impact for the scenario of a great number of small streaming iterations. There exist few dynamic topological sort algorithms that will maintain the topological order across edge and node insertions and will only recompute the ordering if necessary.

Most dependency graphs generated using the Streaming WPS will probably consist of multiple independent subgraphs, no dependencies at all would be the most extreme example, or quite sparse graphs. For this the algorithm described by Pearce and Kelly (2007) seems to be appropriate. Even it is theoretically it is inferior to other algorithms for dynamic topological sorting, it especially performs better on sparse graphs and on dense graphs only a constant factor slower than other algorithms (Pearce and Kelly, 2007).

Dependencies are of particular importance in case of execution failures. If the computation of a streaming iteration fails for whatever reason, all iterations, that directly or indirectly depend on this iteration can not complete. As this also holds true for iterations, that are supplied at a later time in the streaming process, the process can not proceed ignoring the error. Due to this every error that occurs during the execution of a streaming iteration result in the termination

of the streaming process. Dependencies also have a special meaning at the end of a streaming process, when a stop message is sent to notify the streaming process to accept no further inputs and finish pending streaming iterations. At this point all dependencies need to be able to be satisfied, which implies that all referenced input messages have been sent to the streaming process. In case a referenced input message the service is not able to complete gracefully and fail. As references to future streaming iterations are allowed, prior to this point, it is not possible for the Streaming WPS to determine if reference may not be fulfilled. As the service is not able to fail fast for incorrect references, clients using dependencies between streaming iterations have to pay careful attention to references.

It should also be noted, that the smallest referenceable unit for a streaming process is the output of a streaming iteration. Format specific references, e.g. to a particular feature inside a feature collection, are not possible using this protocol and streaming process implementations need to be designed to not need smaller components or have to deploy a own referencing strategy (e.g. by additionally supplying an additional input to identify the feature of the referenced collection). But, as this results in superfluous transfer of data, such solutions should be avoided. One may point out, that there is now way to reference input parameters of other streaming iterations, but this use case should be already covered by the WPS' own input reference parameters (see section 5.3).

5.5. Process Description

The conventional process description mechanism of the WPS is not sufficient to describe streaming processes.

It consists of a `DescribeProcess` request issued to the WPS and the retrieval of one or more process descriptions of the specified process. These descriptions contain detailed descriptions of input and output parameters of the process and information about the supported formats, units of measurement or coordinate reference systems of each parameter. They also include details about allowed values, default value and multiplicity of input parameters (Open Geospatial Consortium, 2007).

Because the Streaming WPS uses the WPS interface only to start a Streaming Process and the WPS interface does not provide any extension points for process descriptions, the `DescribeProcess` operation can only be used to describe the starting process, but not the input or output parameters of a streaming process.

In case of generic processes, e.g. processes that delegate to other WPS processes, information about input and output parameters is not even available prior to the execution of the streaming process. Furthermore input parameter cardinalities may change due to the use of static inputs. By this a valid input parameter for a delegate process may not be used in subsequent inputs because the maximal occurrence of the parameter is already exhausted using static input parameters. By this a process description for a streaming process will always be instance specific and can not be generated by the associated WPS process.

With knowledge of the delegate process a client may has enough information to facilitate the streaming process but for other streaming process there is no way for a generic client to know the input parameters of the process.

To compensate this shortcoming a method is needed to describe a Streaming Process instance at runtime.

- other process description formats
- differentiation between intermediate results and final result

5.6. WPS Specification Shortcomings

- different procedure description format, like in the sensorweb
- process instance need to be identifiable
- WSDL like description language of wps processes
- differentiation between continous outputs and final results
- allow different transport layers (like websockets)
- allowing subsequent input parameters

5.7. Implementation

- Server:
  - based on the 52°North WPS
  - includeable module
  - default implementation uses another WPS process as delegate
- Client
  - small JavaScript library
  - abstracts the message generation and WebSocket interaction
  - may be used to start generic delegation processes

5.8. Streaming Lake-Analyzer WPS

- simple application of the Streaming WPS and MATLAB WPS
- LakeAnalyzer may need further adjustments to allow live analysis
- remove down sampling code
- operate on single point in time
- etc

5.9. Limitations

- No input/output conversion
- Only default format is requested from delegate
- process will not fail fast in under every condition
  - inputs first are checked at execution time
- receivers are only provided with upcoming
  - no replay queue

6. Future Work

7. Conclusion

References

F. Buschmann, R. Meunier, H. Rohnert, P. Sommerlad, and M. Stal. Pattern-oriented software architecture volume 1: A system of patterns. 1996.

J. J. Cole, Y. T. Prairie, N. F. Caraco, W. H. McDowell, L. J. Tranvik, R. G. Striegl, C. M. Duarte, P. Kortelainen, J. A. Downing, J. J. Middelburg, and J. Melack. Plumbing the global carbon cycle: Integrating inland waters into the terrestrial carbon budget. *Ecosystems*, 10(1):172–185, 2007. ISSN 1432-9840.

T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to algorithms*, volume 2. MIT press Cambridge, 2001.

D. H. Douglas and T. K. Peucker. Algorithms for the reduction of the number of points required to represent a digitized line or its caricature. *Cartographica: The International Journal for Geographic Information and Geovisualization*, 10(2):112–122, 1973.

J. A. Downing, Y. T. Prairie, J. J. Cole, C. M. Duarte, L. J. Tranvik, R. G. Striegl, W. H. McDowell, P. Kortelainen, N. F. Caraco, J. M. Melack, and J. J. Middelbiurg. The global abundance and size distribution of lakes, ponds, and impoundments. *Limnology and Oceanography*, 51(5): 2388–2397, 2006. ISSN 1939-5590.

I. Fette and A. Melnikov. The Websocket Protocol. RFC 6455 (Proposed Standard), December 2011. URL http://www.ietf.org/rfc/rfc6455.txt.

T. Foerster, B. Baranski, and H. Borsutzky. Live geoinformation with standardized geoprocessing services. In *Bridging the Geographic Information Sciences*, pages 99–118. Springer, 2012.

Y. Lafon, M. Gudgin, M. Hadley, M. Moreau, N. Mendelsohn, A. Karmarkar, and H. F. Nielsen. SOAP version 1.2 part 1: Messaging framework (second edition). W3C recommendation, W3C, April 2007. URL http://www.w3.org/TR/2007/REC-soap12-part1-20070427/.

Open Geospatial Consortium. OpenGIS Web Processing Service. Implementation Specification, OGC, June 2007. URL http://portal.opengeospatial.org/files/?artifact_id=24151.

R. Pantos and W. May. HTTP Live Streaming. Internet Draft (Informational), October 2013. URL http://tools.ietf.org/id/draft-pantos-http-live-streaming-12.txt.

D. J. Pearce and P. H. J. Kelly. A dynamic topological sort algorithm for directed acyclic graphs. *Journal of Experimental Algorithmics*, 11:1–7, 2007. ISSN 1084-6654.

J. S. Read, D. P. Hamilton, I. D. Jones, K. Muraoka, L. A. Winslow, R. Kroiss, C. H. Wu, and E. Gaiser. Derivation of lake mixing and stratification indices from high-resolution lake buoy data. *Environmental Modelling & Software*, 26(11):1325–1336, 2011. ISSN 1364-8152.

T. Rogers, M. Hadley, and M. Gudgin. Web services addressing 1.0 - core. W3C recommendation, W3C, May 2006. URL http://www.w3.org/TR/2006/REC-ws-addr-core-20060509.

J. Rosenberg, H. Schulzrinne, G. Camarillo, A. Johnston, J. Peterson, R. Sparks, M. Handley, and E. Schooler. SIP: Session Initiation Protocol. RFC 3261 (Proposed Standard), June 2002. URL http://www.ietf.org/rfc/rfc3261.txt.

H. Schulzrinne, A. Rao, and R. Lanphier. Real Time Streaming Protocol (RTSP). RFC 2326 (Proposed Standard), April 1998. URL http://www.ietf.org/rfc/rfc2326.txt.

H. Schulzrinne, S. Casner, R. Frederick, and V. Jacobson. RTP: A Transport Protocol for Real-Time Applications. RFC 3550 (INTERNET STANDARD), July 2003. URL http://www.ietf.org/rfc/rfc3550.txt.

The MathWorks, Inc. The MathWorks, Inc. Software License Agreement, September 2013.

A. Listings

Listing 14: Matlab process configuration describing the function in Listing 1.

```yaml
---
connection:
  host: localhost
  port: 7000
identifier: org.gleon.LakeAnalyzer
version: 1.0.0
title: Lake Analyzer
abstract: Lake Analyzer
function: Run_LA_WPS
inputs:
  # input files
  - identifier: bathymetry
    title: Bathymetry
    abstract: >
      A bathymetry file is a comma delimited (after ver. 3.5,
      tab delimited) text file with extension of [.bth].
      The file starts from one line header and followed
      by the hypsographic data at each depth (Example
      2.1). Depths must start from zero (i.e. surface)
      with a unit of meters, and hypsographic curve data
      with area as square meters is followed by comma
      delimiter. If the hypsographic curve is not
      concluded with zero at the bottom, LakeAnalyzer
      program automatically assigns zero to the bottom
      depth which was defined during the configuration
      process (see section 3). LakeAnalyzer linearly
      interpolates the given hypsographic curve. Change
      to the hypsographic curve due to surface elevation
      change is not supported by the current version of
      the LakeAnalyzer.
    type: { mimeType: text/csv }
  - identifier: waterLevel
    title: Water Level
    abstract: >
      The Water Level file is a tab delimited text file with
      the file extension of [.lvl]. Water level input is
      optional for all the outputs. It is useful for
      estuaries and lake with significant level changes
      which affect hypsographic curve of the water body.
      If the program locates the water level file in
      the correct directory with correct file name, the
      effect of water level fluctuation to the
      bathymetry area are calculated when calculating
      stabilities. The water level file contains one
      header [DateTime level(positive Z down)]. From the
      second line, date/time information with the
      format of [yyyy-mm-dd HH:MM], and water level from
      the highest elevation area measurement available
      (original depth is the surface level stated in the
      *.bth file) should be described. Level depths
      must be equal or greater than 0.
    type: { mimeType: text/csv }
  - identifier: windSpeed
    title: Wind Speed
    abstract: >
      The wind speed file is a tab delimitated text file with
      extension of [.wnd]. Wind speed data are used for
      uStar, Lake Number, and Wedderburn Number
      calculations. Time scale and resolution of the
      wind speed must match the water temperature inputs
      . The file starts from one line header [dateTime
      windSpeed]. From the second line, date/time
      information with the format of [yyyy-mm-dd HH:MM],
      and wind speed data in m/s should be described.
    type: { mimeType: text/csv }
  - identifier: waterTemperature
    title: Water Temperature
    abstract: >
      The water temperature file is a tab delimitated text file
      with a file extension of [.wtr]. The file should
      contain one header which starts from DateTime,
      followed by individual thermister depths in meters
      with format of [temp5] (see Example 2.2).
      LakeAnalyzer uses header information to acquire
      thermister depth. Temperature data should be
      inserted from the following line. The data starts
      from the date/time inputs, which should be
      formatted as [yyyy-mm-dd HH:MM].
    type: { mimeType: text/csv }
  - identifier: salinity
    title: Salinity
    abstract: >
      The salinity file is a tab delimitated text file with the
      file extension of [.sal]. Salinity input is
      optional for all the outputs. If the program
      locates the salinity file in the correct directory
      , the effect of salinity on the density is
      calculated during the process. Salinity time can
      be independent to the other input files. The
      salinity file contains one header line starting
      from DateTime, and followed by depths of
      measurements in format of [salinity2.0]. The
      second line is the beginning of the actual data
      inputs, starting from date/time in format [yyyy-mm
      -dd HH:MM]. After tab separation, salinity should
      be indicated Practical Salinity Scale (PSS) units.
    type: { mimeType: text/csv }
    minOccurs: 0
  # .lke file contents
  - identifier: outputResolution
    title: Output Resolution
    abstract: >
      Output resolution specifies the time-step (s) of the
      calculations made for Lake Analyzer. If the
      temporal resolution of the input data is coarser
      than the entry for this input, calculations will
      be made according to input data resolution.
    type: int
    unit: s
  - identifier: totalDepth
    title: Total Depth
    abstract: >
      Total depth (m) must be greater or equal to than the
      maximum depth given in the .bth file. If the total
      depth is not included in the .bth file, it is
      assumed that the area at total depth is 0 (m2) and
      the depth area curve is linearly interpolated
      from this depth to the values in the .bth file.
    type: double
    unit: m
  - identifier: windHeight
    title: Wind Height
    abstract: >
      Height from surface for wind measurement (m). Height of
      wind measurement is used for the wind speed
      correction factor in Eqn 11.
    type: double
    unit: m
  - identifier: windAveraging
    title: Wind Averaging
    abstract: >
      Wind averaging (s) is the backwards-looking smoothing
      window used for the calculation of uSt and SuSt.
      This calculation allows for the relevant wind
      duration to influence the calculation of wind-
      derived parameters.
    type: int
    unit: s
  - identifier: layerAveraging
    title: Layer Averaging
    abstract: >
      Thermal averaging (s) is the smoothing window used for
      metaT, metaB, thermD, SmetaT, SmetaB, and SthermD.
      Temporal smoothing for thermal layers is intended
      to minimize the effects of internal waves on
      these parameters.
    type: int
    unit: s
  - identifier: outlierWindow
    title: Outlier Window
```

```
    abstract: >
      Outlier window (s) is the window size (seconds) for
            outlier removal, where measurements outside of the
             bounds ( μ ± 2.5 · σ ) based on the standard
            deviation and the mean inside the outlier window
            are removed. Outlier removal is performed on .wtr
            and .wnd files prior to down-sampling (if
            applicable).
    type: int
    unit: s
  - identifier: maxWaterTemp
    title: Maximum Water Temperature
    abstract: >
      Maximum allowed water temperature (°C), where all values
            of .wtr file not fitting this criteria are removed
             before outlier checking.
    type: double
    unit: °C
    minOccurs: 0
  - identifier: minWaterTemp
    title: Minimum Water Temperature
    abstract: >
      Minimum allowed water temperature (°C), where all values
            of .wtr file not fitting this criteria are removed
             before outlier checking.
    type: double
    unit: °C
    minOccurs: 0
  - identifier: maxWindSpeed
    title: Maximum Wind Speed
    abstract: >
      Maximum allowed wind speed (m/s), where all values of .
            wnd file not fitting this criteria are removed
            before outlier checking.
    type: double
    unit: m/s
    minOccurs: 0
  - identifier: minWindSpeed
    title: Minimum Wind Speed
    abstract: >
      Minimum allowed wind speed (m/s), where all values of .
            wnd file not fitting this criteria are removed
            before outlier checking.
    type: double
    unit: m/s
    minOccurs: 0
  - identifier: metaMinSlope
    title: Minimum Metalimnion slope
    abstract: >
      Minimum slope for the range of the metalimnion (kg m-3
            per meter), which is used to calculated values of
            metaT, metaB, SmetaT, and SmetaB according to Eqn
            2.
    type: double
    unit: (kg/m^(-3))/m
  - identifier: mixedTempDifferential
    title: Mixed Temperature Differential
    abstract: >
      Minimum surface to bottom thermistor temperature
            differential (°C) before the case of 'mixed' is
            applied. When 'mixed' is true, all thermal layer
            calculations are no longer applicable, and values
            are given as the depth of the bottom thermistor.
    type: double
    unit: °C
  # .plt file contents
  - identifier: figRes
    abstract: Resolution of the figure in dots per inch
    title: Plot Resolution
    type: int
    values: [ 50, 100, 200, 300, 400, 500 ]
    unit: dpi
  - identifier: figUnits
    title: Figure Units
    abstract: Units of measure for figure size
    type: string
    values: [ inches, centimeters, points ]
  - identifier: figWidth
    title: Figure Width
    abstract: Width of figure (relative to figUnits)
    type: double
  - identifier: figHeight
    title: Figure Height
    abstract: Height of figure (relative to figUnits)
    type: double
  - identifier: leftMargin
    title: Left Margin
    abstract: Space between left edge of figure and y-axis (
          relative to figUnits)
    type: double
  - identifier: rightMargin
    title: Right Margin
    abstract: Space between right edge of figure and right axis
    type: double
  - identifier: topMargin
    title: Top Margin
    abstract: Space between the top edge of the figure and the
          top of the plot axis
    type: double
  - identifier: botMargin
    title: Bottom Margin
    abstract: Space between the bottom edge of the figure and
          the bottom of the plot x-axis
    type: double
  - identifier: fontName
    title: Font Name
    abstract: Font name for plot text
    type: string
    values: [ Arial, Times New Roman, Helvetica ]
  - identifier: fontSize
    title: Font Size
    abstract: Font sive for plot text
    type: int
    values: [ 8, 9, 10, 11, 12, 14 ]
  - identifier: heatMapMin
    title: Minimum Heat Map Value
    abstract: Value that represents the minimum heatmap color
    type: double
  - identifier: heatMapMax
    title: Maximum Heat Map Value
    abstract: Value that represents the maximum heatmap color
    type: double
outputs:
  - identifier: results
    title: Raw Results
    type: { mimeType: text/csv }
  - identifier: results_wtr
    title: Raw Results
    type: { mimeType: text/csv }
  - identifier: N2
    title: Buoyancy frequency
    type:
      mimeType: image/png
      encoding: Base64
  - identifier: SN2
    title: Parent buoyancy frequency
    type:
      mimeType: image/png
      encoding: Base64
  - identifier: Ln
    title: Lake number
    type:
      mimeType: image/png
      encoding: Base64
  - identifier: SLn
    title: Parent lake number
    type:
      mimeType: image/png
      encoding: Base64
  - identifier: metaB
    title: Metalimnion bottom depth
    type:
      mimeType: image/png
      encoding: Base64
  - identifier: SmetaB
    title: Parent metalimnion bottom depth
    type:
      mimeType: image/png
      encoding: Base64
  - identifier: metaT
    title: Metalimnion top depth
    type:
      mimeType: image/png
      encoding: Base64
  - identifier: SmetaT
    title: Parent metalimnion top depth
    type:
      mimeType: image/png
```

```
      encoding: Base64
  - identifier: T1
    title: Mode one vertical seiche period
    type:
      mimeType: image/png
      encoding: Base64
  - identifier: ST1
    title: Parent mode one vertical seiche period
    type:
      mimeType: image/png
      encoding: Base64
  - identifier: St
    title: Schmidt stability
    type:
      mimeType: image/png
      encoding: Base64
  - identifier: thermD
    title: Thermocline depth
    type:
      mimeType: image/png
      encoding: Base64
  - identifier: SthermD
    title: Parent thermocline depth
    type:
      mimeType: image/png
      encoding: Base64
  - identifier: uSt
    title: u star (turblent velocity scale from wind)
    type:
      mimeType: image/png
      encoding: Base64
  - identifier: SuSt
    title: Parent u star (turblent velocity scale from wind)
    type:
      mimeType: image/png
      encoding: Base64
  - identifier: wTemp
    title: Water temperature
    type:
      mimeType: image/png
      encoding: Base64
  - identifier: W
    title: Wedderburn number
    type:
      mimeType: image/png
      encoding: Base64
  - identifier: SW
    title: Parent Wedderburn number
    type:
      mimeType: image/png
      encoding: Base64
  - identifier: wndSpd
    title: Wind speed
    type:
      mimeType: image/png
      encoding: Base64
...
```

Listing 15: Matlab process description generated from the configuration in Listing describing the function in Listing 14.

```
<ProcessDescription statusSupported="false" storeSupported="
      true" wps:processVersion="1.0.0">
  <ows:Identifier>org.gleon.LakeAnalyzer</ows:Identifier>
  <ows:Title>Lake Analyzer</ows:Title>
  <ows:Abstract>Lake Analyzer</ows:Abstract>
  <DataInputs>
    <Input minOccurs="1" maxOccurs="1">
      <ows:Identifier>bathymetry</ows:Identifier>
      <ows:Title>Bathymetry</ows:Title>
      <ows:Abstract>A bathymetry file is a comma delimited (
            after ver. 3.5, tab delimited) text file with
            extension of [.bth]. The file starts from one line
             header and followed by the hypsographic data at
            each depth (Example 2.1). Depths must start from
            zero (i.e. surface) with a unit of meters, and
            hypsographic curve data with area as square meters
             is followed by comma delimiter. If the
            hypsographic curve is not concluded with zero at
            the bottom, LakeAnalyzer program automatically
            assigns zero to the bottom depth which was defined
             during the configuration process (see section 3).
             LakeAnalyzer linearly interpolates the given
            hypsographic curve. Change to the hypsographic
            curve due to surface elevation change is not
            supported by the current version of the
            LakeAnalyzer.</ows:Abstract>
      <ComplexData>
        <Default>
          <Format>
            <MimeType>text/csv</MimeType>
          </Format>
        </Default>
        <Supported>
          <Format>
            <MimeType>text/csv</MimeType>
          </Format>
        </Supported>
      </ComplexData>
    </Input>
    <Input minOccurs="1" maxOccurs="1">
      <ows:Identifier>waterLevel</ows:Identifier>
      <ows:Title>Water Level</ows:Title>
      <ows:Abstract>The Water Level file is a tab delimited
            text file with the file extension of [.lvl]. Water
             level input is optional for all the outputs. It
            is useful for estuaries and lake with significant
            level changes which affect hypsographic curve of
            the water body. If the program locates the water
            level file in the correct directory with correct
            file name, the effect of water level fluctuation
            to the bathymetry area are calculated when
            calculating stabilities. The water level file
            contains one header [DateTime level(positive Z
            down)]. From the second line, date/time
            information with the format of [yyyy-mm-dd HH:MM],
             and water level from the highest elevation area
            measurement available (original depth is the
            surface level stated in the *.bth file) should be
            described. Level depths must be equal or greater
            than 0.</ows:Abstract>
      <ComplexData>
        <Default>
          <Format>
            <MimeType>text/csv</MimeType>
          </Format>
        </Default>
        <Supported>
          <Format>
            <MimeType>text/csv</MimeType>
          </Format>
        </Supported>
      </ComplexData>
    </Input>
    <Input minOccurs="1" maxOccurs="1">
      <ows:Identifier>windSpeed</ows:Identifier>
      <ows:Title>Wind Speed</ows:Title>
      <ows:Abstract>The wind speed file is a tab delimitated
            text file with extension of [.wnd]. Wind speed
            data are used for uStar, Lake Number, and
            Wedderburn Number calculations. Time scale and
            resolution of the wind speed must match the water
            temperature inputs. The file starts from one line
            header [dateTime windSpeed]. From the second line,
             date/time information with the format of [yyyy-mm
            -dd HH:MM], and wind speed data in m/s should be
            described.</ows:Abstract>
      <ComplexData>
        <Default>
          <Format>
            <MimeType>text/csv</MimeType>
```

```
        </Format>
      </Default>
      <Supported>
        <Format>
          <MimeType>text/csv</MimeType>
        </Format>
      </Supported>
    </ComplexData>
  </Input>
  <Input minOccurs="1" maxOccurs="1">
    <ows:Identifier>waterTemperature</ows:Identifier>
    <ows:Title>Water Temperature</ows:Title>
    <ows:Abstract>The water temperature file is a tab
           delimitated text file with a file extension of [.
           wtr]. The file should contain one header which
           starts from DateTime, followed by individual
           thermister depths in meters with format of [temp5]
            (see Example 2.2). LakeAnalyzer uses header
           information to acquire thermister depth.
           Temperature data should be inserted from the
           following line. The data starts from the date/time
            inputs, which should be formatted as [yyyy-mm-dd
           HH:MM].</ows:Abstract>
    <ComplexData>
      <Default>
        <Format>
          <MimeType>text/csv</MimeType>
        </Format>
      </Default>
      <Supported>
        <Format>
          <MimeType>text/csv</MimeType>
        </Format>
      </Supported>
    </ComplexData>
  </Input>
  <Input minOccurs="0" maxOccurs="1">
    <ows:Identifier>salinity</ows:Identifier>
    <ows:Title>Salinity</ows:Title>
    <ows:Abstract>The salinity file is a tab delimitated text
            file with the file extension of [.sal]. Salinity
           input is optional for all the outputs. If the
           program locates the salinity file in the correct
           directory, the effect of salinity on the density
           is calculated during the process. Salinity time
           can be independent to the other input files. The
           salinity file contains one header line starting
           from DateTime, and followed by depths of
           measurements in format of [salinity2.0]. The
           second line is the beginning of the actual data
           inputs, starting from date/time in format [yyyy-mm
           -dd HH:MM]. After tab separation, salinity should
           be indicated Practical Salinity Scale (PSS) units
           .</ows:Abstract>
    <ComplexData>
      <Default>
        <Format>
          <MimeType>text/csv</MimeType>
        </Format>
      </Default>
      <Supported>
        <Format>
          <MimeType>text/csv</MimeType>
        </Format>
      </Supported>
    </ComplexData>
  </Input>
  <Input minOccurs="1" maxOccurs="1">
    <ows:Identifier>outputResolution</ows:Identifier>
    <ows:Title>Output Resolution</ows:Title>
    <ows:Abstract>Output resolution specifies the time-step (
           s) of the calculations made for Lake Analyzer. If
           the temporal resolution of the input data is
           coarser than the entry for this input,
           calculations will be made according to input data
           resolution.</ows:Abstract>
    <LiteralData>
      <ows:DataType ows:reference="xs:int"/>
      <UOMs>
        <Default>
          <ows:UOM>s</ows:UOM>
        </Default>
        <Supported>
          <ows:UOM>s</ows:UOM>
        </Supported>
      </UOMs>
      <ows:AnyValue/>
    </LiteralData>
  </Input>
  <Input minOccurs="1" maxOccurs="1">
    <ows:Identifier>totalDepth</ows:Identifier>
    <ows:Title>Total Depth</ows:Title>
    <ows:Abstract>Total depth (m) must be greater or equal to
           than the maximum depth given in the .bth file. If
           the total depth is not included in the .bth file,
           it is assumed that the area at total depth is 0 (
           m2) and the depth area curve is linearly
           interpolated from this depth to the values in the
           .bth file.</ows:Abstract>
    <LiteralData>
      <ows:DataType ows:reference="xs:double"/>
      <UOMs>
        <Default>
          <ows:UOM>m</ows:UOM>
        </Default>
        <Supported>
          <ows:UOM>m</ows:UOM>
        </Supported>
      </UOMs>
      <ows:AnyValue/>
    </LiteralData>
  </Input>
  <Input minOccurs="1" maxOccurs="1">
    <ows:Identifier>windHeight</ows:Identifier>
    <ows:Title>Wind Height</ows:Title>
    <ows:Abstract>Height from surface for wind measurement (m
           ). Height of wind measurement is used for the wind
            speed correction factor in Eqn 11.</ows:Abstract>
    <LiteralData>
      <ows:DataType ows:reference="xs:double"/>
      <UOMs>
        <Default>
          <ows:UOM>m</ows:UOM>
        </Default>
        <Supported>
          <ows:UOM>m</ows:UOM>
        </Supported>
      </UOMs>
      <ows:AnyValue/>
    </LiteralData>
  </Input>
  <Input minOccurs="1" maxOccurs="1">
    <ows:Identifier>windAveraging</ows:Identifier>
    <ows:Title>Wind Averaging</ows:Title>
    <ows:Abstract>Wind averaging (s) is the backwards-looking
            smoothing window used for the calculation of uSt
           and SuSt. This calculation allows for the relevant
            wind duration to influence the calculation of
           wind-derived parameters.</ows:Abstract>
    <LiteralData>
      <ows:DataType ows:reference="xs:int"/>
      <UOMs>
        <Default>
          <ows:UOM>s</ows:UOM>
        </Default>
        <Supported>
          <ows:UOM>s</ows:UOM>
        </Supported>
      </UOMs>
      <ows:AnyValue/>
    </LiteralData>
  </Input>
  <Input minOccurs="1" maxOccurs="1">
    <ows:Identifier>layerAveraging</ows:Identifier>
    <ows:Title>Layer Averaging</ows:Title>
    <ows:Abstract>Thermal averaging (s) is the smoothing
           window used for metaT, metaB, thermD, SmetaT,
           SmetaB, and SthermD. Temporal smoothing for
           thermal layers is intended to minimize the effects
            of internal waves on these parameters.</
           ows:Abstract>
    <LiteralData>
      <ows:DataType ows:reference="xs:int"/>
      <UOMs>
        <Default>
          <ows:UOM>s</ows:UOM>
        </Default>
        <Supported>
```

```
            <ows:UOM>s</ows:UOM>
          </Supported>
        </UOMs>
        <ows:AnyValue/>
      </LiteralData>
    </Input>
    <Input minOccurs="1" maxOccurs="1">
      <ows:Identifier>outlierWindow</ows:Identifier>
      <ows:Title>Outlier Window</ows:Title>
      <ows:Abstract>Outlier window (s) is the window size (
             seconds) for outlier removal, where measurements
             outside of the bounds ( μ ± 2.5 · σ ) based on the
              standard deviation and the mean inside the
             outlier window are removed. Outlier removal is
             performed on .wtr and .wnd files prior to down-
             sampling (if applicable).</ows:Abstract>
      <LiteralData>
        <ows:DataType ows:reference="xs:int"/>
        <UOMs>
          <Default>
            <ows:UOM>s</ows:UOM>
          </Default>
          <Supported>
            <ows:UOM>s</ows:UOM>
          </Supported>
        </UOMs>
        <ows:AnyValue/>
      </LiteralData>
    </Input>
    <Input minOccurs="0" maxOccurs="1">
      <ows:Identifier>maxWaterTemp</ows:Identifier>
      <ows:Title>Maximum Water Temperature</ows:Title>
      <ows:Abstract>Maximum allowed water temperature (°C),
             where all values of .wtr file not fitting this
             criteria are removed before outlier checking.</
             ows:Abstract>
      <LiteralData>
        <ows:DataType ows:reference="xs:double"/>
        <UOMs>
          <Default>
            <ows:UOM>°C</ows:UOM>
          </Default>
          <Supported>
            <ows:UOM>°C</ows:UOM>
          </Supported>
        </UOMs>
        <ows:AnyValue/>
      </LiteralData>
    </Input>
    <Input minOccurs="0" maxOccurs="1">
      <ows:Identifier>minWaterTemp</ows:Identifier>
      <ows:Title>Minimum Water Temperature</ows:Title>
      <ows:Abstract>Minimum allowed water temperature (°C),
             where all values of .wtr file not fitting this
             criteria are removed before outlier checking.</
             ows:Abstract>
      <LiteralData>
        <ows:DataType ows:reference="xs:double"/>
        <UOMs>
          <Default>
            <ows:UOM>°C</ows:UOM>
          </Default>
          <Supported>
            <ows:UOM>°C</ows:UOM>
          </Supported>
        </UOMs>
        <ows:AnyValue/>
      </LiteralData>
    </Input>
    <Input minOccurs="0" maxOccurs="1">
      <ows:Identifier>maxWindSpeed</ows:Identifier>
      <ows:Title>Maximum Wind Speed</ows:Title>
      <ows:Abstract>Maximum allowed wind speed (m/s), where all
              values of .wnd file not fitting this criteria are
              removed before outlier checking.</ows:Abstract>
      <LiteralData>
        <ows:DataType ows:reference="xs:double"/>
        <UOMs>
          <Default>
            <ows:UOM>m/s</ows:UOM>
          </Default>
          <Supported>
            <ows:UOM>m/s</ows:UOM>
          </Supported>
        </UOMs>
        <ows:AnyValue/>
      </LiteralData>
    </Input>
    <Input minOccurs="0" maxOccurs="1">
      <ows:Identifier>minWindSpeed</ows:Identifier>
      <ows:Title>Minimum Wind Speed</ows:Title>
      <ows:Abstract>Minimum allowed wind speed (m/s), where all
              values of .wnd file not fitting this criteria are
              removed before outlier checking.</ows:Abstract>
      <LiteralData>
        <ows:DataType ows:reference="xs:double"/>
        <UOMs>
          <Default>
            <ows:UOM>m/s</ows:UOM>
          </Default>
          <Supported>
            <ows:UOM>m/s</ows:UOM>
          </Supported>
        </UOMs>
        <ows:AnyValue/>
      </LiteralData>
    </Input>
    <Input minOccurs="1" maxOccurs="1">
      <ows:Identifier>metaMinSlope</ows:Identifier>
      <ows:Title>Minimum Metalimnion slope</ows:Title>
      <ows:Abstract>Minimum slope for the range of the
             metalimnion (kg m-3 per meter), which is used to
             calculated values of metaT, metaB, SmetaT, and
             SmetaB according to Eqn 2.</ows:Abstract>
      <LiteralData>
        <ows:DataType ows:reference="xs:double"/>
        <UOMs>
          <Default>
            <ows:UOM>(kg/m^(-3))/m</ows:UOM>
          </Default>
          <Supported>
            <ows:UOM>(kg/m^(-3))/m</ows:UOM>
          </Supported>
        </UOMs>
        <ows:AnyValue/>
      </LiteralData>
    </Input>
    <Input minOccurs="1" maxOccurs="1">
      <ows:Identifier>mixedTempDifferential</ows:Identifier>
      <ows:Title>Mixed Temperature Differential</ows:Title>
      <ows:Abstract>Minimum surface to bottom thermistor
             temperature differential (°C) before the case of '
             mixed' is applied. When 'mixed' is true, all
             thermal layer calculations are no longer
             applicable, and values are given as the depth of
             the bottom thermistor.</ows:Abstract>
      <LiteralData>
        <ows:DataType ows:reference="xs:double"/>
        <UOMs>
          <Default>
            <ows:UOM>°C</ows:UOM>
          </Default>
          <Supported>
            <ows:UOM>°C</ows:UOM>
          </Supported>
        </UOMs>
        <ows:AnyValue/>
      </LiteralData>
    </Input>
    <Input minOccurs="1" maxOccurs="1">
      <ows:Identifier>figRes</ows:Identifier>
      <ows:Title>Plot Resolution</ows:Title>
      <ows:Abstract>Resolution of the figure in dots per inch</
             ows:Abstract>
      <LiteralData>
        <ows:DataType ows:reference="xs:int"/>
        <UOMs>
          <Default>
            <ows:UOM>dpi</ows:UOM>
          </Default>
          <Supported>
            <ows:UOM>dpi</ows:UOM>
          </Supported>
        </UOMs>
        <ows:AllowedValues>
          <ows:Value>50</ows:Value>
          <ows:Value>100</ows:Value>
          <ows:Value>200</ows:Value>
```

```
          <ows:Value>300</ows:Value>
          <ows:Value>400</ows:Value>
          <ows:Value>500</ows:Value>
        </ows:AllowedValues>
      </LiteralData>
    </Input>
    <Input minOccurs="1" maxOccurs="1">
      <ows:Identifier>figUnits</ows:Identifier>
      <ows:Title>Figure Units</ows:Title>
      <ows:Abstract>Units of measure for figure size</
            ows:Abstract>
      <LiteralData>
        <ows:DataType ows:reference="xs:string"/>
        <ows:AllowedValues>
          <ows:Value>inches</ows:Value>
          <ows:Value>centimeters</ows:Value>
          <ows:Value>points</ows:Value>
        </ows:AllowedValues>
      </LiteralData>
    </Input>
    <Input minOccurs="1" maxOccurs="1">
      <ows:Identifier>figWidth</ows:Identifier>
      <ows:Title>Figure Width</ows:Title>
      <ows:Abstract>Width of figure (relative to figUnits)</
            ows:Abstract>
      <LiteralData>
        <ows:DataType ows:reference="xs:double"/>
        <ows:AnyValue/>
      </LiteralData>
    </Input>
    <Input minOccurs="1" maxOccurs="1">
      <ows:Identifier>figHeight</ows:Identifier>
      <ows:Title>Figure Height</ows:Title>
      <ows:Abstract>Height of figure (relative to figUnits)</
            ows:Abstract>
      <LiteralData>
        <ows:DataType ows:reference="xs:double"/>
        <ows:AnyValue/>
      </LiteralData>
    </Input>
    <Input minOccurs="1" maxOccurs="1">
      <ows:Identifier>leftMargin</ows:Identifier>
      <ows:Title>Left Margin</ows:Title>
      <ows:Abstract>Space between left edge of figure and y-
            axis (relative to figUnits)</ows:Abstract>
      <LiteralData>
        <ows:DataType ows:reference="xs:double"/>
        <ows:AnyValue/>
      </LiteralData>
    </Input>
    <Input minOccurs="1" maxOccurs="1">
      <ows:Identifier>rightMargin</ows:Identifier>
      <ows:Title>Right Margin</ows:Title>
      <ows:Abstract>Space between right edge of figure and
            right axis</ows:Abstract>
      <LiteralData>
        <ows:DataType ows:reference="xs:double"/>
        <ows:AnyValue/>
      </LiteralData>
    </Input>
    <Input minOccurs="1" maxOccurs="1">
      <ows:Identifier>topMargin</ows:Identifier>
      <ows:Title>Top Margin</ows:Title>
      <ows:Abstract>Space between the top edge of the figure
            and the top of the plot axis</ows:Abstract>
      <LiteralData>
        <ows:DataType ows:reference="xs:double"/>
        <ows:AnyValue/>
      </LiteralData>
    </Input>
    <Input minOccurs="1" maxOccurs="1">
      <ows:Identifier>botMargin</ows:Identifier>
      <ows:Title>Bottom Margin</ows:Title>
      <ows:Abstract>Space between the bottom edge of the figure
            and the bottom of the plot x-axis</ows:Abstract>
      <LiteralData>
        <ows:DataType ows:reference="xs:double"/>
        <ows:AnyValue/>
      </LiteralData>
    </Input>
    <Input minOccurs="1" maxOccurs="1">
      <ows:Identifier>fontName</ows:Identifier>
      <ows:Title>Font Name</ows:Title>
      <ows:Abstract>Font name for plot text</ows:Abstract>
      <LiteralData>
        <ows:DataType ows:reference="xs:string"/>
        <ows:AllowedValues>
          <ows:Value>Arial</ows:Value>
          <ows:Value>Times New Roman</ows:Value>
          <ows:Value>Helvetica</ows:Value>
        </ows:AllowedValues>
      </LiteralData>
    </Input>
    <Input minOccurs="1" maxOccurs="1">
      <ows:Identifier>fontSize</ows:Identifier>
      <ows:Title>Font Size</ows:Title>
      <ows:Abstract>Font sive for plot text</ows:Abstract>
      <LiteralData>
        <ows:DataType ows:reference="xs:int"/>
        <ows:AllowedValues>
          <ows:Value>8</ows:Value>
          <ows:Value>9</ows:Value>
          <ows:Value>10</ows:Value>
          <ows:Value>11</ows:Value>
          <ows:Value>12</ows:Value>
          <ows:Value>14</ows:Value>
        </ows:AllowedValues>
      </LiteralData>
    </Input>
    <Input minOccurs="1" maxOccurs="1">
      <ows:Identifier>heatMapMin</ows:Identifier>
      <ows:Title>Minimum Heat Map Value</ows:Title>
      <ows:Abstract>Value that represents the minimum heatmap
            color</ows:Abstract>
      <LiteralData>
        <ows:DataType ows:reference="xs:double"/>
        <ows:AnyValue/>
      </LiteralData>
    </Input>
    <Input minOccurs="1" maxOccurs="1">
      <ows:Identifier>heatMapMax</ows:Identifier>
      <ows:Title>Maximum Heat Map Value</ows:Title>
      <ows:Abstract>Value that represents the maximum heatmap
            color</ows:Abstract>
      <LiteralData>
        <ows:DataType ows:reference="xs:double"/>
        <ows:AnyValue/>
      </LiteralData>
    </Input>
  </DataInputs>
  <ProcessOutputs>
    <Output>
      <ows:Identifier>results_wtr</ows:Identifier>
      <ows:Title>Raw Results</ows:Title>
      <ComplexOutput>
        <Default>
          <Format>
            <MimeType>text/csv</MimeType>
          </Format>
        </Default>
        <Supported>
          <Format>
            <MimeType>text/csv</MimeType>
          </Format>
        </Supported>
      </ComplexOutput>
    </Output>
    <Output>
      <ows:Identifier>results</ows:Identifier>
      <ows:Title>Raw Results</ows:Title>
      <ComplexOutput>
        <Default>
          <Format>
            <MimeType>text/csv</MimeType>
          </Format>
        </Default>
        <Supported>
          <Format>
            <MimeType>text/csv</MimeType>
          </Format>
        </Supported>
      </ComplexOutput>
    </Output>
    <Output>
      <ows:Identifier>N2</ows:Identifier>
      <ows:Title>Buoyancy frequency</ows:Title>
      <ComplexOutput>
        <Default>
```

```
          <Format>
            <MimeType>image/png</MimeType>
            <Encoding>Base64</Encoding>
          </Format>
        </Default>
        <Supported>
          <Format>
            <MimeType>image/png</MimeType>
            <Encoding>Base64</Encoding>
          </Format>
        </Supported>
      </ComplexOutput>
    </Output>
    <Output>
      <ows:Identifier>SN2</ows:Identifier>
      <ows:Title>Parent buoyancy frequency</ows:Title>
      <ComplexOutput>
        <Default>
          <Format>
            <MimeType>image/png</MimeType>
            <Encoding>Base64</Encoding>
          </Format>
        </Default>
        <Supported>
          <Format>
            <MimeType>image/png</MimeType>
            <Encoding>Base64</Encoding>
          </Format>
        </Supported>
      </ComplexOutput>
    </Output>
    <Output>
      <ows:Identifier>Ln</ows:Identifier>
      <ows:Title>Lake number</ows:Title>
      <ComplexOutput>
        <Default>
          <Format>
            <MimeType>image/png</MimeType>
            <Encoding>Base64</Encoding>
          </Format>
        </Default>
        <Supported>
          <Format>
            <MimeType>image/png</MimeType>
            <Encoding>Base64</Encoding>
          </Format>
        </Supported>
      </ComplexOutput>
    </Output>
    <Output>
      <ows:Identifier>SLn</ows:Identifier>
      <ows:Title>Parent lake number</ows:Title>
      <ComplexOutput>
        <Default>
          <Format>
            <MimeType>image/png</MimeType>
            <Encoding>Base64</Encoding>
          </Format>
        </Default>
        <Supported>
          <Format>
            <MimeType>image/png</MimeType>
            <Encoding>Base64</Encoding>
          </Format>
        </Supported>
      </ComplexOutput>
    </Output>
    <Output>
      <ows:Identifier>metaB</ows:Identifier>
      <ows:Title>Metalimnion bottom depth</ows:Title>
      <ComplexOutput>
        <Default>
          <Format>
            <MimeType>image/png</MimeType>
            <Encoding>Base64</Encoding>
          </Format>
        </Default>
        <Supported>
          <Format>
            <MimeType>image/png</MimeType>
            <Encoding>Base64</Encoding>
          </Format>
        </Supported>
      </ComplexOutput>
    </Output>
    <Output>
      <ows:Identifier>SmetaB</ows:Identifier>
      <ows:Title>Parent metalimnion bottom depth</ows:Title>
      <ComplexOutput>
        <Default>
          <Format>
            <MimeType>image/png</MimeType>
            <Encoding>Base64</Encoding>
          </Format>
        </Default>
        <Supported>
          <Format>
            <MimeType>image/png</MimeType>
            <Encoding>Base64</Encoding>
          </Format>
        </Supported>
      </ComplexOutput>
    </Output>
    <Output>
      <ows:Identifier>metaT</ows:Identifier>
      <ows:Title>Metalimnion top depth</ows:Title>
      <ComplexOutput>
        <Default>
          <Format>
            <MimeType>image/png</MimeType>
            <Encoding>Base64</Encoding>
          </Format>
        </Default>
        <Supported>
          <Format>
            <MimeType>image/png</MimeType>
            <Encoding>Base64</Encoding>
          </Format>
        </Supported>
      </ComplexOutput>
    </Output>
    <Output>
      <ows:Identifier>SmetaT</ows:Identifier>
      <ows:Title>Parent metalimnion top depth</ows:Title>
      <ComplexOutput>
        <Default>
          <Format>
            <MimeType>image/png</MimeType>
            <Encoding>Base64</Encoding>
          </Format>
        </Default>
        <Supported>
          <Format>
            <MimeType>image/png</MimeType>
            <Encoding>Base64</Encoding>
          </Format>
        </Supported>
      </ComplexOutput>
    </Output>
    <Output>
      <ows:Identifier>T1</ows:Identifier>
      <ows:Title>Mode one vertical seiche period</ows:Title>
      <ComplexOutput>
        <Default>
          <Format>
            <MimeType>image/png</MimeType>
            <Encoding>Base64</Encoding>
          </Format>
        </Default>
        <Supported>
          <Format>
            <MimeType>image/png</MimeType>
            <Encoding>Base64</Encoding>
          </Format>
        </Supported>
      </ComplexOutput>
    </Output>
    <Output>
      <ows:Identifier>ST1</ows:Identifier>
      <ows:Title>Parent mode one vertical seiche period</
          ows:Title>
      <ComplexOutput>
        <Default>
          <Format>
            <MimeType>image/png</MimeType>
            <Encoding>Base64</Encoding>
          </Format>
        </Default>
```

```
        <Supported>
          <Format>
            <MimeType>image/png</MimeType>
            <Encoding>Base64</Encoding>
          </Format>
        </Supported>
      </ComplexOutput>
    </Output>
    <Output>
      <ows:Identifier>St</ows:Identifier>
      <ows:Title>Schmidt stability</ows:Title>
      <ComplexOutput>
        <Default>
          <Format>
            <MimeType>image/png</MimeType>
            <Encoding>Base64</Encoding>
          </Format>
        </Default>
        <Supported>
          <Format>
            <MimeType>image/png</MimeType>
            <Encoding>Base64</Encoding>
          </Format>
        </Supported>
      </ComplexOutput>
    </Output>
    <Output>
      <ows:Identifier>thermD</ows:Identifier>
      <ows:Title>Thermocline depth</ows:Title>
      <ComplexOutput>
        <Default>
          <Format>
            <MimeType>image/png</MimeType>
            <Encoding>Base64</Encoding>
          </Format>
        </Default>
        <Supported>
          <Format>
            <MimeType>image/png</MimeType>
            <Encoding>Base64</Encoding>
          </Format>
        </Supported>
      </ComplexOutput>
    </Output>
    <Output>
      <ows:Identifier>SthermD</ows:Identifier>
      <ows:Title>Parent thermocline depth</ows:Title>
      <ComplexOutput>
        <Default>
          <Format>
            <MimeType>image/png</MimeType>
            <Encoding>Base64</Encoding>
          </Format>
        </Default>
        <Supported>
          <Format>
            <MimeType>image/png</MimeType>
            <Encoding>Base64</Encoding>
          </Format>
        </Supported>
      </ComplexOutput>
    </Output>
    <Output>
      <ows:Identifier>uSt</ows:Identifier>
      <ows:Title>u star (turblent velocity scale from wind)</
            ows:Title>
      <ComplexOutput>
        <Default>
          <Format>
            <MimeType>image/png</MimeType>
            <Encoding>Base64</Encoding>
          </Format>
        </Default>
        <Supported>
          <Format>
            <MimeType>image/png</MimeType>
            <Encoding>Base64</Encoding>
          </Format>
        </Supported>
      </ComplexOutput>
    </Output>
    <Output>
      <ows:Identifier>SuSt</ows:Identifier>
      <ows:Title>Parent u star (turblent velocity scale from
            wind)</ows:Title>
      <ComplexOutput>
        <Default>
          <Format>
            <MimeType>image/png</MimeType>
            <Encoding>Base64</Encoding>
          </Format>
        </Default>
        <Supported>
          <Format>
            <MimeType>image/png</MimeType>
            <Encoding>Base64</Encoding>
          </Format>
        </Supported>
      </ComplexOutput>
    </Output>
    <Output>
      <ows:Identifier>wTemp</ows:Identifier>
      <ows:Title>Water temperature</ows:Title>
      <ComplexOutput>
        <Default>
          <Format>
            <MimeType>image/png</MimeType>
            <Encoding>Base64</Encoding>
          </Format>
        </Default>
        <Supported>
          <Format>
            <MimeType>image/png</MimeType>
            <Encoding>Base64</Encoding>
          </Format>
        </Supported>
      </ComplexOutput>
    </Output>
    <Output>
      <ows:Identifier>W</ows:Identifier>
      <ows:Title>Wedderburn number</ows:Title>
      <ComplexOutput>
        <Default>
          <Format>
            <MimeType>image/png</MimeType>
            <Encoding>Base64</Encoding>
          </Format>
        </Default>
        <Supported>
          <Format>
            <MimeType>image/png</MimeType>
            <Encoding>Base64</Encoding>
          </Format>
        </Supported>
      </ComplexOutput>
    </Output>
    <Output>
      <ows:Identifier>SW</ows:Identifier>
      <ows:Title>Parent Wedderburn number</ows:Title>
      <ComplexOutput>
        <Default>
          <Format>
            <MimeType>image/png</MimeType>
            <Encoding>Base64</Encoding>
          </Format>
        </Default>
        <Supported>
          <Format>
            <MimeType>image/png</MimeType>
            <Encoding>Base64</Encoding>
          </Format>
        </Supported>
      </ComplexOutput>
    </Output>
    <Output>
      <ows:Identifier>wndSpd</ows:Identifier>
      <ows:Title>Wind speed</ows:Title>
      <ComplexOutput>
        <Default>
          <Format>
            <MimeType>image/png</MimeType>
            <Encoding>Base64</Encoding>
          </Format>
        </Default>
        <Supported>
          <Format>
            <MimeType>image/png</MimeType>
            <Encoding>Base64</Encoding>
```

```
            </Format>
          </Supported>
        </ComplexOutput>
      </Output>
    </ProcessOutputs>
  </ProcessDescription>
```

B. Source Code

Streaming WPS	Extension for the 52°North WPS to allow of Inputs and Outputs over WebSockets. https://github.com/autermann/streaming-wps
Matlab WPS	Extension for the 52°North WPS to offer Matlab functions and scripts as OGC Web Processing Service algorithms. https://github.com/autermann/matlab-wps
streaming-wps-js	Streaming WPS JavaScript Bindings https://github.com/autermann/streaming-wps-js
WPS Commons	52°North WPS convenience classes and bootstrapping code. https://github.com/autermann/wps-commons
Matlab Connector	Matlab function execution on (pooled) remote Matlab instances. https://github.com/autermann/matlab-connector
Lake-Analyzer	Matlab source code for Lake Analyzer https://github.com/autermann/Lake-Analyzer
YAML API	A Jackson-like API to read and create YAML nodes (based on SnakeYAML). https://github.com/autermann/yaml

C. XML Namespaces

For clarity XML name spaces are omitted in XML Listings. Their respective value can be found in the following table:

Prefix	Namespace
xlink	http://www.w3.org/1999/xlink
xml	http://www.w3.org/XML/1998/namespace
xs	http://www.w3.org/2001/XMLSchema
xsi	http://www.w3.org/2001/XMLSchema-instance
soap	http://www.w3.org/2003/05/soap-envelope
wsa	http://www.w3.org/2005/08/addressing
ows	http://www.opengis.net/ows/1.1
wps	http://www.opengis.net/wps/1.0.0
stream	https://github.com/autermann/streaming-wps

Plagiatserklärung des Studierenden

Hiermit versichere ich, dass die vorliegende Arbeit über *Streaming Web-Services for Calculating Live Hydrological Derivatives* selbstständig verfasst worden ist, dass keine anderen Quellen und Hilfsmittel als die angegebenen benutzt worden sind und dass die Stellen der Arbeit, die anderen Werken – auch elektronischen Medien – dem Wortlaut oder Sinn nach entnommen wurden, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht worden sind.

Münster, den 5. Mai 2014 _____

Ich erkläre mich mit einem Abgleich der Arbeit mit anderen Texten zwecks Auffindung von Übereinstimmungen sowie mit einer zu diesem Zweck vorzunehmenden Speicherung der Arbeit in eine Datenbank einverstanden.

Münster, den 5. Mai 2014 _____