Connecting R to the Sensor Web

DRAFT

Daniel Nüst, Christoph Stasch, Edzer Pebesma

Institute for Geoinformatics, University of Muenster
Weseler Str. 253, 48151 Münster
{daniel.nuest, staschc, edzer.pebesma}@uni-muenster.de

Abstract

Interoperable data exchange and reproducibility are increasingly important for modern scientific research. This paper shows how three open source projects work together to realize this: (i) the R project, providing the lingua franca for statistical analysis, (ii) the Open Geospatial Consortium's Sensor Observation Service (SOS), a standardized data warehouse service for storing and retrieving sensor measurements, and (iii) sos4R, a new project that connects the former two. We show how sos4R can bridge the gap between two communities in science: spatial statistical analysis and visualization on one side, and the Sensor Web community on the other. sos4R enables R users to integrate (near real-time) sensor observations directly into R. Finally, we evaluate the functionality of sos4R. The software encapsulates the service's complexity with typical R function calls in a common analysis workflow, but still gives users full flexibility to handle interoperability issues. We conclude that it is able to close the gap between R and the sensor web.

1 Introduction

As the whole process of environmental analysis is moving to the internet (creating a model web (Geller, 2008)) the data sources also are, which re-

sults in a growing amount of spatially distributed sensor data publicly available through standardized web service interfaces. Yet, client applications to access, to analyze and to visualize this data lag behind compared to spatial data in a raster or vector format. This is partly due to a gap between the research fields of sensor web and data analysis or geostatistics. The former group devotes itself to the development and implementation of standards for sensor data storage and exchange in the realm of Open Geospatial Consortium (OGC) or International Organization for Standardization (ISO) – a rather technical undertaking often based in computer science. The latter group, often domain specialists, actually analyzes the data to infer about the processes that generated it. This organizational and personal difference (in the sense that few people work in both research areas) results in a separate development of tools and knowledge about service-based data retrieval and data processing. We try to narrow this gap by providing a tool that can benefit both communities and further facilitate interdisciplinary collaboration and research. A motivation for this work is to increase acceptance of the Sensor Web ideas and tools, and allow more data analysts and modellers to benefit from the Sensor Web.

The general driving force is to motivate analysts and researchers to adopt open source software and open standardized interfaces, and to conduct reproducible research. To achieve this goal, the main contribution of this work is a Sensor Observation Service (SOS) (Na et al., 2007) client written in R (R Development Core Team, 2010) that addresses the translation of the required observation models to R data structures. We investigate if a (partial) gap closure can be done by a single piece of software as the decisive chain link in a sequence of existing tools.

The term reproducible research was first proposed by Jon Claerbout (Fomel and Claerbout, 2009) to tackle the issues of current scientific publications in the domain of computational sciences, where the components necessary to recreate the presented results are completely provided. It "refers to the idea that the ultimate product of research is the paper along with access to the full computational environment used to produce the results in the paper such as the code, data, etc. necessary for reproduction of the results and building upon the research" (Reproducible Research Planet, 2010). The test set-up is a critical component in experimental sciences. Although in computational science one can expect that the same code and data result in the same outcome, there still is the aspect of sharing the both of these. By using tools that directly connect code and publication reproducible research does not only allow others to replicate results, but also the original authors themselves. We want to support the analysis part of reproducibility from a technical standpoint.

The remainder of this paper is structured as follows. Sections 1.1 to 1.3 present the basic concepts and components upon which the product is built and related work. Section 2 introduces a short use case whose exemplary steps are used to develop the requirements presented in section 3. Afterwards we describe the concept of sos4R in sections 4 to 5 and conclude with an evaluation (section 6). An agenda for future work is outlined in section 7.

1.1 The R project

R is an emerging language and environment for statistical computing and graphics (Vance, 2009). It provides a wide range of statistical techniques and analysis functions as well as capabilities of graphical plotting that stand out and partly underlie its success. A complete description of R's functions and its user and developer communities (ranging from finance to biomedicine) is not possible here, but the most important features from the sos4R perspective are: R is freely available under an open source license; it is extensible via packages, which are small (to large) units of code and data that can be comfortably added to an R environment.

The *Sweave* document format (Leisch, 2005) supports the concept of weaving, where the text and the analysis code are in the same document (introduced by Knuth under the term "literate programming" (Knuth, 1984)) and is particularly well suited for reproducible research as a single document generates text and results. The package *cacher* implements cached computations as a framework for reproducible research (Peng, 2008). By plugging in the most important data source of the OGC Sensor Web into R, a wide range of data becomes readily accessible to the large community of R users.

1.2 The Sensor Observation Service

Providing data via web services supersedes the need for local file copies, which might become outdated. Also, a flexible filtering of data on the service side reduces the communication necessary to download the data. The OGC SOS is a common specification for a web service providing pull-based access to sensor observations and sensor descriptions. It is part of the Sensor Web Enablement Initiative (SWE) (Botts et al., 2008). The observations can be provided in real-time or as archived data sets. They can be subsetted in a flexible way. For example, a query like "Provide observations from water gauge Cologne_Rhine_1 for the time period from 01/01/2010 to 31/08/2010 where the water level is above 5 meters" can be

created in a common format and sent to the SOS which returns only matching observations.

As the SOS always provides the observations in a common format based on Extensible Markup Language (XML)[1], namely Observations & Measurements (O&M) (Cox, 2007a), clients do not have to adapt different data formats when integrating observations. An observation consists of information about the geographic feature which is observed, the time when the observation was taken, the sensor, the observed phenomenon, and the observation's result. The SOS relies upon the Sensor Model Language (SensorML) (Botts, 2007) for providing sensor metadata like the sensor's position, calibration information, and inputs and outputs of the sensing process. By connecting the powerful R environment with the SOS, a huge amount of features for analysis and visualization become readily accessible to the community of SOS users.

1.3 Related Work

The OGC offers widely used service standards apart from SOS, for example Web Features Service[2] (WFS), Web Coverage Service[3] (WCS) and Web Map Service[4] (WMS). A connection is possible in R through the *rgdal* (Keitt et al., 2010) package which provides bindings to the Geospatial Data Abstraction Library (GDAL)[5]. However, these do not (directly) support typical sensor data, like in situ measurements, but vector and raster ("image") data. WFS could provide sensor observations as point features, but that is not intended.

With this distinction, related work can be seen from different perspectives. First there are other packages in R which allow the comfortable download of near real-time data directly into R data structures. Second, there are other clients for Sensor Observation Services.

For the former, one can find a few packages, for example the *GEOquery* package of the Bioconductor project (Davis, 2007) and *WDI* (Arel-Bundock, 2010) package, which allow access to the Gene Expression Omnibus repository of gene expression experiments, and access to the World Bank's World Development Indicators respectively. The *quantmod* package (Ryan, 2008) provides functions for downloading stock exchange data. However, these use proprietary interfaces for a specific data base.

---

[1] http://www.w3.org/XML/

[2] http://www.opengeospatial.org/standards/wfs

[3] http://www.opengeospatial.org/standards/wcs

[4] http://www.opengeospatial.org/standards/wms

[5] http://www.gdal.org/

The latter perspective comprises mostly viewers of varying abilities:

- simple viewers, like the OpenOOIS.org Map Viewer[6] or the SOS client for OpenLayers[7], which show the positions of sensors on a map and can include links to retrieve observational data;
- advanced viewers, like 52°North Clients (52°North, 2010) or GeoCENS Sensor Web Browser (Liang et al., 2010), which include (three dimensional) visualization of sensors' positions on a map, tabular or graphic display of data;
- plug-ins to geographic information systems, like ArcGIS and uDig (52°North, 2010) or gvSIG (Tamayo et al., 2009), which allow the import, subsequent processing with the respective software and most importantly combining with a large number of other data sources.

2 Application Example

To illustrate the use of the sos4R package, we present the use case of a researcher modelling local weather data. She requires temperature measurements for the region and the time frame of interest. The workflow consists of the following steps:

1. Create a connection to a SOS
2. Set up query parameters
3. Request observation data
4. Analyse and visualize data with R functionalities (not part of this work).

For the sake of simplicity this example does not utilize the (arbitrary) spatial, temporal and value-based querying features of the SOS.

Listing 1 and Figure 1 show the complete code and the result of the analysis, a temperature plot. The code and figures are also available from http://www.nordholmen.net/sos4r/agile2011.

```
library("sos4R"); library("xts")
# 1. step
weathersos = SOS("http://v-swe.uni-muenster.de:8080/WeatherSOS/sos")

# 2. step
station <- sosProcedures(weathersos)[[1]]
temperatureOffering <- sosOfferings(weathersos)[["ATMOSPHERIC_TEMPERATURE"]]
temperature <- sosObservedProperties(temperatureOffering)[1]
september <- sosCreateTimePeriod(sos = weathersos,
                begin = as.POSIXct("2010-09-01 00:00"),
                end = as.POSIXct("2010-09-30 00:00"))
```

[6] http://www.openioos.org/real_time_data/gm_sos.html

[7] http://www.openlayers.org/dev/examples/sos.html

```
# 3. step
obsSept <- getObservation(sos = weathersos, observedProperty = temperature,
                 procedure = station,
                 eventTime = sosCreateEventTimeList(september),
                 offering = temperatureOffering)
data <- sosResult(obsSept)

# 4. step
summary(data); data[1:2,]; names(data)
# create time series
tempSept <- xts(x = data[["urn:ogc:def:property:OGC::Temperature"]],
         order.by = data[["Time"]])
# calculate regression (polynomial fitting) and plot
temp <- data[["urn:ogc:def:property:OGC::Temperature"]]
time <- as.numeric(data[["Time"]])
x = loess(temp~time, na.omit(data), enp.target = 10)
plot(tempSept, main = "Temperature at Station One",
                 xlab = "Time", ylab = paste("Temperature in", attrib-
utes(temp)[["unit of measurement"]]),
                 major.ticks = "weeks")
lines(data$Time, x$fitted, col = 'red', lwd=3)
```

Listing 1. Example analysis with sos4R (lines starting with "#" are comments)

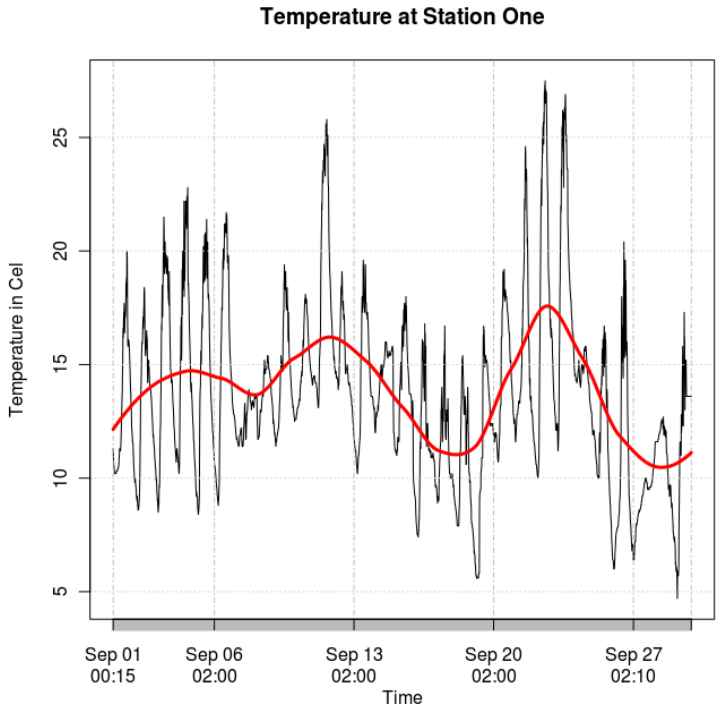

Fig. 1. Time series plot of exemplary analysis showing temperature values from September 2009 together with a regression line

3 Requirements Analysis

The sos4R client naturally implements the "core profile" from the SOS specification which comprises the three basic operations for information retrieval, namely *GetCapabilities* for retrieving service descriptions, *GetObservation* for querying observation data, and *DescribeSensor* for retrieving sensor metadata encoded as SensorML.

The software must allow for the comfortable creation and exploration of these operations including exceptional events. This should also be possible for non-experts, in the sense that the user shall not be closely familiar with the whole range of OGC specifications related to SOS and, to a certain extent, even without knowing the specification of the core operations themselves. The request creation shall be supported by sensible default settings, but still allow full flexibility for advanced settings. The responses shall be transformed to common R data structures and thereby allow immediate subsequent analysis. Convenience functions shall be provided to explore e.g. the capabilities of a service.

As the SOS standard has been made suitable for almost any possible type of observation, the possible encodings it may return are too large to cope with for a typical client. O&M as response format is very flexible and supports many, sometimes uncommon or rarely needed, possibilities. Thus, an XML Schema[8] profile which limits the processable markup is needed. In the absence of a generic simple observation profile, the O&M profile of the 52°North SOS[9] was chosen. This SOS is a stable, compliancy tested open source implementation and the profile suffices for common use cases. In brief, this profile restricts the different possibilities for encoding temporal, spatial and result information to certain pre-defined types. Naturally, not all existing SOSs support the same encodings and filters as in the 52°North profile or support them completely themselves. That is why exchangeability must be seen as the most important requirement. The user shall be given flexible tools to exchange only the necessary processing steps to include her own data markup.

A potential problem was identified early during the development. If the user ought to be "shielded from" the SOS specification, in order to lower the threshold for new users, the difference in technical terms of a SWE expert became apparent. Naturally used terms like "procedure" or "observed property" were introduced for clarity in specifications, but are not the typical "laymen" terms of professionals from other domains.

---

[8] http://www.w3.org/XML/Schema

[9] https://wiki.52north.org/bin/view/Sensornet/SensorObservationService

Thus, a qualitative survey was conducted with a small number of expert users with either a lot experience with SOS or none at all, or vice versa, having high experience with R or close to none.

The survey stated general tasks, e.g. "Create (a connection to) a SOS instance, e.g. based on the URL [...]." or "Request observation data of a known location and phenomenon [...] for a certain time period". The participants were asked to outline a short R script of how they would expect a SOS client in R could be used. The questionnaire was accompanied by a short introductory description of R and SOS.

The following could be observed for experienced R/inexperienced SOS users: a typical user immediately wanted to plot and run calculations on the returned object, showing that she/he was not aware that the response contains more (meta-) information than just the plain data as a table for example. None of the subjects used the typical SWE terms, but rather commands like "read" to request data, "sensors" for procedures, and "phenomenon" for observed properties, even "location" referring to a feature of interest. Different results are naturally possible with people from other domains. Furthermore, it became also very apparent that the users wrote rather simple function calls with all the settings they require as arguments, i.e. no higher class objects as input parameters.

Experienced SOS users followed the workflow laid out by the SOS operations, starting with a GetCapabilities request and then starting a GetObservation operation with the respective parameters. As the threshold is considerably lower for these types of end users, we do not see any requirement for special accommodation.

An interpretation of the results allows the following statements with regard to the user interface. Functions to directly access the original SOS operations with the SWE terms must be provided for users who are familiar with these standards. An argument against other terminology is the confusion of users moving from simple use cases to more advanced levels or going deeper into Sensor Web concepts. Thus, more abstract functions containing a translation between domains, i.e. the SWE domain and the respective application domain, can be implemented based on further user testing or based on dictionaries like in Annex B to O&M (Cox, 2007a). These simpler functions, named after common functions like “read”, encapsulate the complexity of standards where full features are not needed.

4 Software Design

Three main challenges have been identified: designing a web service client in a command line based environment, ensuring exchangeability to enable the integration of different SOS profiles, and mapping the conceptual models of OGC specifications to R classes.

4.1 User Interface

The basic R user interface is a command line prompt. It allows help documents to be opened, provides command completion and a history, and it can be used in the same way on all platforms. Syntax highlighting for R source files is available for many standard editors, but there are dedicated programming environments as well. Nevertheless, the primary target user interface for sos4R is the console and all functions are designed with regard to that. This is a limiting factor to the requirement of exploration which profits from a graphical user interface. We suggest different tools, like sensor catalogues and registries (Jirka et al., 2009), for exploring a service's capabilities for the first time.

4.2 Component Exchangeability

Extensibility is a crucial advantage of open source software, but not all users might be able or willing to download the source code of a program to introduce minor changes to get software to work for their use case. For a client of a web service standard that does not (and cannot, see section 3) implement all possible features out of the box, users must be able to adapt the processing chain. Therefore, three key components in a SOS connection are easily exchangeable.

- Parsers: parsing functions create R objects based on textual encoding. These are used by the service interface receiving responses.
- Converters: conversion functions transform a given character string representation of a data value into the correct atomic data type in R, e.g. double.
- Encoders: encoding functions translate an R representation of an object to external objects, often character strings, so that they can be transferred to other software. These are used by the service interface sending requests.

Similar to a template method pattern (Freeman et al., 2004) in object oriented programming, the basic workflow is fixed but the single steps have interchangeable components. The sequence in a client server setup can be abstracted to five steps (see Figure 2):

1. building a request (based on user input);
2. transforming the request object to a transferable format – the encoding;
3. sending it to the service using a certain method and receiving the response – the transfer;
4. processing the response (includes converting) – the parsing;
5. and analyzing the data.

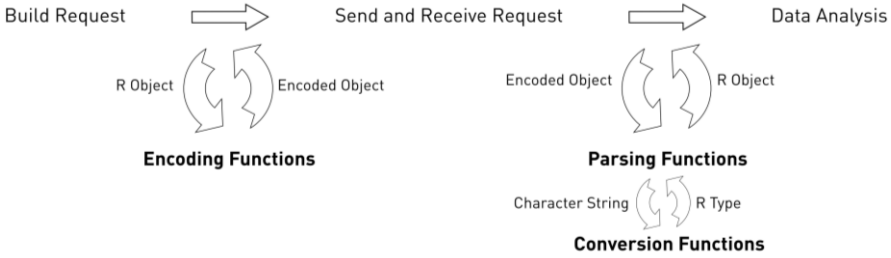

Fig. 2. The sensor data analysis workflow: steps and exchangeable components

The code mechanism takes advantage of the possibility to store function objects as variables. The functions performing the first and last step are provided in a named list when creating a SOS connection. This list can be exchanged by a user. The simple example in listing 2 illustrates this.

```
myParseSensorML <- function(obj) {
        root <- xmlRoot(obj)
        return(xmlName(root))
}
mysos = SOS(url = "http://sos.org/sos",
        parsers = SosParsingFunctions("DescribeSensor" = myParseSensorML))
```

Listing 2. Exchange of parsing function for DescribeSensor operation.

In this example, the user does not want the response object of the DescribeSensor operation to be stored completely, but only keep the name of the root element. Adding the name of this function to a named list, where the names correspond to an XML element name or the operation that is performed, makes parts of the parsing exchangeable. This list is created with a utility functions that combines default functions with the user-defined ones, which is useful if only a certain element must be handled specially.

The same is possible for the conversion functions where possible names are the definition of a data field or its unit of measurement.

We took a different approach for encoders, because the possible encodings and request documents are already defined in the service. SOS supports two HTTP[10] based bindings: GET with a key-value pair encoding in the URL[11]; and POST with XML encoded requests. Therefore a single generic method suffices for each binding and is added to the encoders list named after the connection method. This method must then be implemented for all objects which need to be encoded and R resolves the correct method based on the input objects.

4.3 Mapping OGC Specifications to R Classes

The transfer of OGC requests and data structures to R objects is a key aspect. We describe a selection of elements and operations from the various specifications in this section. Some general considerations that originate partly in the manual mapping procedure are as follows.

- The 52°North SOS profile has been used as a guideline for the included features. Many optional (but rarely used) elements and attributes of specifications are not included;
- XML elements are mapped to S4 classes in R, which in turn contain slots with contained elements (potentially as lists) and attributes.
- The extra layer of element types is omitted for brevity.
- Extensions of the XML type system, e.g. substitution groups or abstract classes, are only partially implemented.

The implementation of specifications consists of four components: classes (in an object oriented sense) for XML elements, functions for parsing/decoding (XML) character representations, functions for encoding classes (to XML), and functions for creating instances of the classes. The last functions preferably utilize only R data types and structures. Differences in encodings, as some elements are only possible in certain encoding types, are not explicitly mentioned.

---

[10] http://www.w3.org/Protocols/

[11] It must be noted that the GET binding is not part of the official standard, as that section was accidentally left out, but defined in a best-practice paper available at http://www.oostethys.org/best-practices/best-practices-get.

4.3.1 OGC Web Services Common

The OGC Web Services Common specification (OWS Common) specifies "[...] many of the aspects that are, or should be, common to all or multiple OWS interface Implementation Specifications" (Whiteside, 2007). OWS common forms the basis for all OGC service specification including the SOS. It comprises operations (request and response) and their transfer (using HTTP), encodings, parameters, and data structures. This specification is implemented generically, and the classes and methods could be extracted to a separate package and reused by other packages. Some elements not used in the SOS specification are omitted.

GetCapabilities is the core operation of OWS Common and both the request and response are implemented completely. *ExceptionReports*, which wrap exceptions to handle erroneous states in the service or illegal requests, are fully covered (see Figure 3).

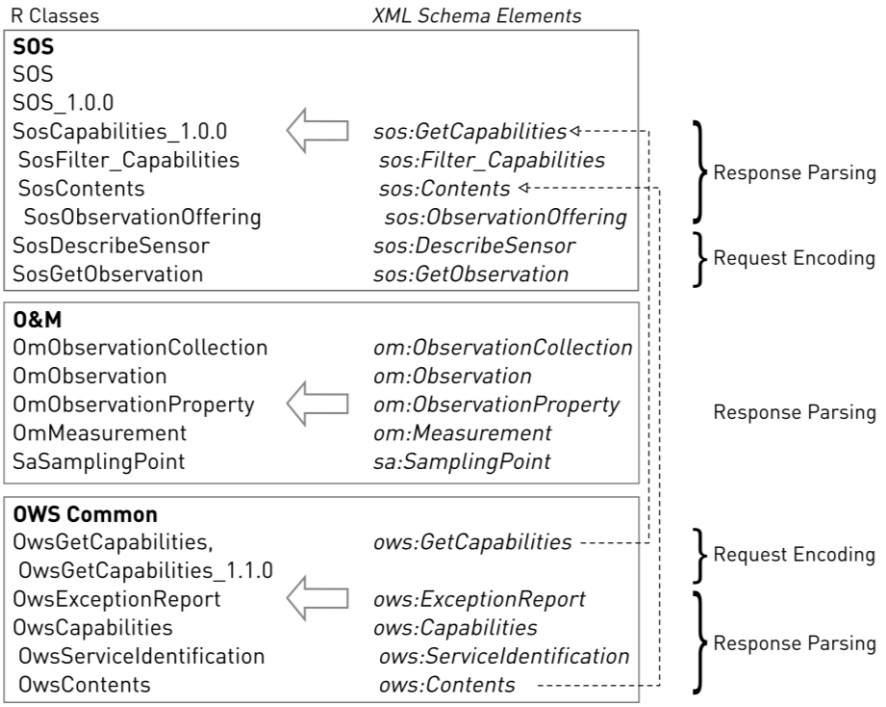

Fig. 3. Overview of important mappings from XML elements to R classes

4.3.2 Sensor Observation Service

Classes are implemented equivalent to the important objects in connecting and retrieving data from a SOS (see Figure 3). These are the class SOS itself, and classes for the core operations.

The virtual[12] class *SOS* represents the concept of a connection to a SOS and can be used with different functions for extracting information from a SOS. It contains slots for elements that are unlikely to change between versions. It has a sub-class, *SOS_1.0.0*, with additional or specifically redefined slots for all required information for a connection to a SOS instance implementing version 1.0.0 of the specification. This structure loosely follows the adapter pattern (Freeman et al., 2004), where new service interfaces can be supported by adding a new adapter class, but common methods are reused and the user experience does not change. But the distinction between versions also allows easy adaption of future specifications.

The classes for core operation requests contain all request parameters in a suitable format, i.e. (lists of) objects of classes presented here.

4.3.3 Geography Markup Language

The Geography Markup Language (GML) (Portele, 2003) provides basic spatial and temporal data types in the service requests and responses of a SOS. Only the elements and attributes that we see as most common are included, e.g. not the GML metadata element. At some points flattening is used to remove levels in the class model by not implementing an extra class for a type or element. For example, the *description* element is not a class, but just a character slot to hold the content of a *description* element. Otherwise, the full element hierarchy is transferred into R classes. Abstract schema elements are virtual classes.

Basic spatial classes are implemented, e.g. *Point*. The structure for more complex elements like *LineString* or *Polygon* is laid out in the code. Temporal elements are represented with classes for time instants and time periods, which eventually hold an object of class POSIXt, the R class for calendar dates and times.

4.3.4 Observation and Measurements, and Sampling Features

The specification Observations and Measurements consists of two parts: part one defines an observation schema (Cox, 2007a); part two defines sampling features (Cox, 2007b). Figure 3 shows the classes for observation collection, observations, and measurements. The specifications variety is

[12] The R term for the notion of abstract classes.

resembled in the slot types of classes, which might accept any object type. Where possible, fixed structures are mapped directly, e.g. the *observedProperty* of an observation is an object of class *SwePhenomenonProperty*. *SamplingPoint* is the only mapped element of part two of the specification. It is commonly used to encode positions of sensors.

The goal of the workflow is to make the actual values of an observation response directly usable in R. Therefore the *result* element of each *Observation* is parsed directly into an R *data.frame* object (see section 4.3.5).

4.3.5 SWE Common

The SWE Common specification is part of SensorML (Botts, 2007) and provides common data types used in the several sensor related specifications. The amount of classes from this specification is rigorously limited to essential ones, because SWE Common uses many abstract type declarations and possibly very complex nesting. The *CompositePhenomenon* holding a list of *Phenomenon* elements is supported. The classes respectively elements *Phenomenon* and *TextBlock* are essential for the parsing of a *Values* element, which itself is not a class but parsed directly to a *data.frame*. The former two elements contain metadata (amongst others, units of measurement and markup information). The latter contains the actual data values. A *data.frame* is a matrix-like structure with indexed columns of different types (like numeric, character, or list) and indexed rows. All metadata attributes are attached to the *data.frame* object.

4.3.6 OpenGIS Filter Encoding

OpenGIS Filter Encoding (Vretanos, 2005) specifies a general query language in XML. SOS implementations can indicate the supported filters in the service metadata description in the element *Filter_Capabilities*. It is completely implemented so that users can browse an instance's supported filters. Functions help in the comfortable creation of the most common filters in a GetObservation operation. These are temporal filtering in the *eventTime* element and spatial filtering with a bounding box in the *featureOfInterest* element.

Temporal filtering operands are after, before, during, and equals. The operand classes contain a slot for the respective sensible time element of GML, for example a *TimeInstant* for a “before” query.

Spatial filtering operands are contains, intersects, and overlaps. The operand classes contain a slot for spatial elements of GML.

Result filtering based on parameter values is only possible via manual creation of the property filters.

5 Implementation

The presented concepts and requirements are implemented in the project sos4R[13], which is published as open source software under GNU General Public License[14] (GPL).

The software relies on two packages that both originate from the Omegahat Project for Statistical Computing (Temple Lang, 2000). They are available under BSD license[15]. The first is package *XML* (Temple Lang, 2010), and it provides functions for parsing XML, both document-based (which is used) and event-driven approaches, and creating XML. The second is package *RCurl* (Temple Lang, 2007), a package for composing HTTP request, i.e. GET and POST operations, to web servers. It builds upon the powerful library *libcurl*[16] and its extensive list of features, like HTTPS, cookies, and authentication, which can be exploited by expert users when creating a SOS connection. A test of the package *XMLSchema*[17], which allows automatic creation of R classes and conversion functions from XML objects to objects of these classes yielded technical problems, as the package cannot handle the large number of (partly circular) references in the schemata.

The implementation makes extensive use of S4 classes (Genolini, 2008). These allow object-oriented programming in R, including type safety, inheritance and encapsulation (also of the construction of objects).

Default values of parameters are based partly on personal experience, the aforementioned survey (e.g. the temporal operator), and the metadata description of service and contents. The number of required parameters could thereby be lowered to one in the case of the GetObservation operation, i.e. the offering.

Convenience functions, which can be used to create the most common elements solely in R code, serve the requirement of encapsulating the large-scale OGC specifications. For instance, there is a function for time periods, which accepts R classes for time (POSIXt) for begin and end times.

The software was successfully tested with Sweave and allows the original response documents to be saved for archiving and reproducibility.

---

[13] http://www.nordholmen.net/sos4R/

[14] http://www.gnu.org/licenses/old-licenses/gpl-2.0.txt

[15] http://www.opensource.org/licenses/bsd-license.php

[16] http://curl.haxx.se/

[17] http://www.omegahat.org/XMLSchema/

6 Discussion and Conclusion

In this paper we presented the concept and implementation of the connection of the statistical analysis environment R to the Sensor Web's data provision service, the SOS. It supports the user in creating requests for observational data based on the service description metadata. The requests include flexible subsetting (thematic, spatial and temporal) for volume reduction of transferred files. For common use cases, this is possible using only R calls and without any contact to the actual request mechanism or documents that were sent and received. Thanks to the online data storage and open data format, users are not restricted to specific file formats, integration of most up-to-date data (potentially directly from the source holder) is easy, and observations can even be requested in near real-time. The existing R tools for reproducible research are complemented well by sos4R and reproducibility can be increased.

The exchangeability features presented in section 4.2 are part of a more abstract common methodology for statistical analysis of sensor observations. The use case in section 2 and Figure 2 contain a common workflow for the import and processing of data from the Sensor Web into an analysis environment. There are fixed, ordered steps due to dependencies of the operations but also flexible components if necessary.

The component exchangeability is powerful and supplies the required degree of flexibility. The mapping of SOS and O&M specifications, including the data structures from SWE Common, works well for all tested use cases. The presented design decisions are valid. However, a complete modelling of SWE Common elements like multidimensional complex data arrays, to R classes, could allow more flexibility. This could comprise coercion functions to lists or time series classes. The automation of that coercion is cumbersome (for example automatic detection of the attribute that holds temporal information) and not exploited yet.

A similar case is filtering, where a full R implementation of (property-based) result filtering can assist users that are not familiar with XML and details of specifications. Here we see an especially high gain with an integration of the spatial, temporal, and upcoming spatio-temporal classes from other R packages as filtering input.

Shortcomings of the software can be found in few areas. The exploration capabilities based on service metadata descriptions are not visual and require previous knowledge of available information.

Generic standards limit the interoperability. We try to compensate with swappable segments in the analysis workflow until future versions of the standards supply improvement. A considerable deficit is the integration in-

to spatial, temporal and spatio-temporal data structures, which is merely indirect. An automatic integration of the downloaded data structures into suitable spatial or temporal data structures would be optimal, for example using the packages *sp* (Bivand et al., 2009) or *xts* (Ryan, 2010). This requires common markup of data and an internal logic which can automatically detect temporal, rational, or ordinal variables. Beyond that coherent spatio-temporal data structures are not supported. We trace this back to a general lack of integrated spatio-temporal data structures and analysis tools.

The client was successfully tested with several SOS instances based on implementations by 52°North[18], OOTethys[19] and Degree[20]. Other (open and closed source) implementations exist and must be tested for too, as soon as public test instances are available (for example Mapserver[21], istSOS[22]). The limitation of many of the specifications to the 52°North SOS profile did not prevent connection to other services. In fact, only expected adjustments for unknown data fields (converters) had to be made. Several of the R classes implemented in this work, in particular those for OWS Common (GetCapabilities) and specifications related to GML can be reused when an R client for another OGC web service is needed.

The software performed well during development, but further performance testing is required, especially with large data sets. Regarding the processing of response documents in R, the available memory is a limiting factor which could be handled by event-based parsing techniques. Regarding the data transfer, O&M might not be the appropriate format for massive data sets. Instead, a SOS could respond in a binary compressed format, like netCDF[23] for gridded data, and the user adapts the parsing function to the respective import method of that format. Our pragmatic approach reveals a lot of challenges and pitfalls of current software systems. The Sensor Web community focuses on developing standards and services. In our experience there are content providers as early adopters, but not many actual analyses are based on SOS data. We see the practical approach to provide a tool to other researchers to support collaboration and reproducibility as sound and viable. sos4R is the first SOS client for a software environment that focuses on coherent analysis of the data (but still includes visual presentation) and certainly is a novelty in the field.

---

[18] https://52north.org/communities/sensorweb/sos/

[19] http://www.oostethys.org/

[20] http://wiki.deegree.org/deegreeWiki/deegree3/SensorObservationService

[21] http://mapserver.org/ogc/sos_server.html

[22] http://istgeo.ist.supsi.ch/software/istsos/

[23] http://www.unidata.ucar.edu/software/netcdf

7 Outlook and Future Work

Implementing the SOS core profile was the first step taken. However, we also see potential in implementing the transactional profile for which use cases spanning different scientific areas shall be developed. Output of analysis, e.g. some spatial, temporal or spatio-temporal interpolation or forecast done in R could provide a feeding layer from a variety of specific sources. Or (intermediate) results of an analysis could be published and archived in a SOS together with R scripts as procedures.

The list of features for enhancement is lengthy, so only a few ideas shall be listed here: the plotting of observation offerings or service capabilities on a map; a stronger connection with package *sp* (e.g. coercion functions between GML spatial classes and sp classes); a better connection with time series classes in R (e.g. direct conversion of a downloaded time series).

A related research topic is the modelling of spatio-temporal data in R: at the time of writing, R support for handling of spatio-temporal data in a fully referenced way is in its infancy. Another topic of a more general scope is the development of a simple observation profile for O&M. A simple profile can limit the possible data types and structures for the sake of easier interoperability.

The sos4R package was tested with available open source SOS implementations. To aid wide adaptation, upcoming service implementations must be continuously tested. If new service instances contain new result markups their processing can be added to the code by any user thanks to the open source strategy. The next step regarding the package is to discover how the software is adapted by users. A long term goal of an open source project naturally is to build up a solid user and developer base. In the best case there is a large overlap between these two groups. But the success also depends on the adaptation by data providers and the number of SOS with interesting data. That is why an overall objective is to motivate content owners and analysts to make their data available through the SOS interface.

Acknowledgements

This work was generously supported by the 52°North Student Innovation Prize for Geoinformatics 2010. We also thank the reviewers for their valuable comments.

References

52°North - Initiative for Geospatial Open Source Software (2010) https://52north.org/, Last date accessed 2010-10-20.

Arel-Bundock, V. (2010) WDI: Search and download data from the World Bank's World Development Indicators', http://cran.r-project.org/package=WDI.

Bivand, R. S. Pebesma, E. J. (2008), Applied spatial data analysis with R, Springer, NY, http://www.asdar-book.org/.

Botts, M. (2007) OGC Implementation Specification 07-000: OpenGIS Sensor Model Language (SensorML), Technical Report, Open Geospatial Consortium.

Botts, M., Percivall, G., Reed, C. and Davidson, J. (2008) OGC Sensor Web Enablement: Overview and High Level Architecture, in S. Nittel, A. Labrinidis and A. Stefanidis (Eds.), GeoSensor Networks, Vol. 4540 of Lecture Notes in Computer Science, Springer Berlin / Heidelberg, pp. 175–190. http://dx.doi.org/10.1007/978-3-540-79996-2_10.

Cox, S. (2007*a*) OGC Implementation Specification 07-022r1: Observations and Measurements - Part 1 - Observation schema, Technical Report, Open Geospatial Consortium.

Cox, S. (2007*b*) OGC Implementation Specification 07-022r3: Observations and Measurements - Part 2 - Sampling Features, Technical Report, Open Geospatial Consortium.

Davis, S. and Meltzer, P. S. (2007) GEOquery: a bridge between the Gene Expression Omnibus (GEO) and BioConductor, Bioinformatics 23(14), pp. 1846–1847, http://dx.doi.org/10.1093/bioinformatics/btm254.

Fomel, S. and Claerbout, J. F. (2009) Guest editors' introduction: Reproducible Research, Computing in Science and Engineering 11, pp. 5–7.

Freeman, E., Freeman, E., Bates, B. and Sierra, K. (2004) Head First Design Patterns, O'Reilly Media, http://www.worldcat.org/isbn/0596007124.

Geller, Gary N., M. F. (2008) Looking Forward: Applying an Ecological Model Web to assess impacts of climate change, Biodiversity 9(3&4).

Genolini, C. (2008), A (Not So) Short Introduction to S4', http://cran.r-project.org/other-docs.html.

Jirka, S., Bröring, A. and Stasch, C. (2009), Discovery Mechanisms for the Sensor Web, Sensors 9, 2661–2681, http://www.mdpi.com/1424-8220/9/4/2661/.

Keitt, T. H., Bivand, R., Pebesma, E. and Rowlingson, B. (2010) rgdal: Bindings for the Geospatial Data Abstraction Library, http://CRAN.R-project.org/package=rgdal

Knuth, D. E. (1984) Literate Programming, The Computer Journal 27, pp. 97–111.

Leisch, F. (2005) Sweave: Dynamic Generation of Statistical Reports Using Literate Data Analysis, http://www.stat.uni-muenchen.de/~leisch/Sweave/.

Liang, S., Chang, D., Badger, J., Rezel, R., Chen, S., Huang, C. and Li, R. (2010) GeoCENS: Geospatial Cyberinfrastructure for Environmental Sensing, Extended Abstracts for Presentation at GIScience 2010. http://www.giscience2010.org/pdfs/paper_219.pdf.

Na, A., Priest, M., Niedzwiadek, H. and Davidson, J. (2007) OGC Implementation Specification 06-009r6: Sensor Observation Service, Technical Report, Open Geospatial Consortium.

Peng, R. D. (2008), Caching and Distributing Statistical Analyses in R, Journal of Statistical Software 26(7), http://www.jstatsoft.org/v26/i07/.

Portele, C. (2003), OpenGIS Geography Markup Language (GML) Encoding Standard 07-036, Open Geospatial Consortium.

R Development Core Team (2010), R: A Language and Environment for Statistical Computing, R Foundation for Statistical Computing, Vienna, Austria. ISBN 3-900051-07-0, http://www.R-project.org.

Reproducible Research Planet (2010). http://www.rrplanet.com/, Last date accessed 2011-01-03.

Ryan, J. A. (2008), Quantitative Financial Modelling & Trading Framework for R, http://www.quantmod.com, Last date accessed 2011-01-01.

Ryan, J. A., Ulrich, J. M. (2010) xts: Extensible Time Series, http://cran.r-project.org/package=xts.

Tamayo, A., Huerta, J., Granell, C., Diaz, L. and Quiros, R. (2009), 'gvSOS: A New Client for the OGC Sensor Observation Service Interface Standard, *Transactions in GIS 13, pp. 47–61.*

Temple Lang, D. (2000) The Omegahat Environment: New Possibilities for Statistical Computing, Journal of Computational and Graphical Statistics 9, pp. 423–451, http://www.jstor.org/stable/1390938.

Temple Lang, D. (2007), R as a Web Client – the RCurl package, Journal of Statistical Software, http://cran.r-project.org/web/packages/RCurl/. www.omegahat.org/RCurl/RCurlJSS.pdf

Temple Lang, D. (2010) XML: Tools for parsing and generating XML within R and S-plus', http://cran.r-project.org/package=XML

Vance, A. (2009) Data Analysts Captivated by R's Power, http://www.nytimes.com/2009/01/07/technology/business-computing/07program.html, Last date accessed 2010-10-20.

Vretanos, P. A. (2005) OpenGIS Filter Encoding Implementation Specification 04-095, Technical Report, Open Geospatial Consortium.

Whiteside, A. (2007), OGC Implementation Specification 06-121r3: OGC Web Services Common Specification, Technical Report, Open Geospatial Consortium.