Environmental Modelling and Software 117 (2019) 237–249

Contents lists available at ScienceDirect

Environmental Modelling & Software

journal homepage: www.elsevier.com/locate/envsoft

A template framework for environmental timeseries data acquisition

Argyrios Samourkasidis[a,*], Evangelia Papoutsoglou[b], Ioannis N. Athanasiadis[a]

[a] Information Technology Group, Wageningen University, Hollandseweg 1, Wageningen, 6706 KN, the Netherlands
[b] Plant Breeding Group, Wageningen University, Droevendaalsesteeg 1, Wageningen, 6708 PB, the Netherlands

ARTICLE INFO

Keywords:
Environmental timeseries
Internet of things
Syntactic interoperability
Data acquisition
Templates
Big data

ABSTRACT

Environmental timeseries data variety is exploding in the Internet of Things era, making data reuse a very demanding task. Data acquisition and integration remains a laborious step of the environmental data lifecycle. Environmental data heterogeneity is a persistent issue, as data are becoming available through different protocols and stored under diverse, custom formats. In this work, we deal with syntactic heterogeneity in environmental timeseries data. Our approach is based on describing different dataset syntaxes using abstract representations, called templates. We designed and implemented EDAM (Environmental Data Acquisition Module), a template framework that facilitates timeseries data acquisition and integration. EDAM templates are written using programming language-agnostic semantics, and can be reused both for input and output, thus enabling data reuse via transformations across different formats. We demonstrate EDAM generality in seven case studies, which involve scraping online data, extracting observations from a relational database, or aggregating historical timeseries stored in local files. Case studies span different environmental sciences domains, including meteorology, agriculture, urban air quality and hydrology. We also demonstrate EDAM for data dissemination, as instructed by output templates. We identified several syntactic interoperability challenges though the case studies, that include managing with differences in formatting observables, temporal and spatial references, and metadata documentation, and addressed them with EDAM. EDAM implementation has been released under an open-source license.

1. Introduction

Environmental data management, that is, acquisition, processing, storage, and dissemination (Athanasiadis and Mitkas, 2004; Mason et al., 2014) is becoming more challenging in the era of Big Data (BD) and the Internet of Things (IoT). In the contemporary data-rich society, a great variety of sensors and IoT devices enable the collection of large observation volumes, which can be further processed for enabling new knowledge insights. At the same time, this era is characterized as knowledge-poor, since universal data management and heterogeneous data integration remain still open challenges (Negru et al., 2016).

Environmental data acquisition seems to be the most laborious step within the environmental data lifecycle (Terrizzano et al., 2015; Harth et al., 2013; Horsburgh et al., 2009). This is attributed to the *heterogeneity* pertinent to environmental data sources. Environmental datasets are collected and stored under different data models in various forms; mainly in files and relational databases (Horsburgh et al., 2011). Datasets which do not share common data formats and/or communication protocols are difficult to be re-used without human expert involvement. *Syntactic heterogeneity* is a factor which hinders the adoption of a universal strategy to acquire data originating from disparate information sources. It also obstructs the environmental data science core objective: to narrow the data-to-knowledge latency (Elag et al., 2017) by supporting environmental data discovery and access; and by enabling re-usability (Horsburgh et al., 2009; Ames et al., 2012; Athanasiadis, 2015; Holzworth et al., 2015; Granell et al., 2010). FAIR (Findable, Accessible, Interoperable, Reusable) guiding principles for scientific data management and stewardship highlight the importance of scientific data *reusability* and *reproducibility* (Wilkinson et al., 2016). Long-term archival and preservation of digital assets also implies the regular transformation of data between storage formats and media.

There are two approaches to tackle syntactic heterogeneity. The first is to use/adopt frameworks which were designed to facilitate environmental data discovery and accessibility, such as the OGC Sensor Web Enablement (SWE) (Botts et al., 2008) and CUAHSI Hydrologic Information System (HIS) (Horsburgh et al., 2009). Such systems hide the underlying complexity of environmental data sources and expose them in a standardized manner, through established data models (e.g. O&M (Cox, 2011), SensorML (Botts and Robin, 2014), WaterML 2.0 (Taylor, 2014) etc.). The various datasets need to be stored in a common schema

* Corresponding author.
E-mail address: argysamo@gmail.com (A. Samourkasidis).

https://doi.org/10.1016/j.envsoft.2018.10.009
Received 19 March 2018; Received in revised form 8 September 2018; Accepted 26 October 2018
Available online 17 November 2018

in order to be exposed through a data sharing framework. This entails certain modifications, which introduce overhead, and commonly require a strong computer science background to implement (Andrae et al., 2009). The second approach is to develop programming language scripts, each one tailored to the custom data format (Eberle et al., 2013; Woodard, 2016). These custom-to-data scripts usually transform a dataset into a common data schema (Porter et al., 2014), which allows for further processing, analysis or dissemination tasks (Boote et al., 2015). These approaches have been used also for exchanging data between environmental models (i.e (Porter et al., 2014; Horita et al., 2015; Peckham and Goodall, 2013; Jones et al., 2015)).

Both approaches rely upon computer programming skills that are not always available. This contradicts the lowering e-science barriers movement (Swain et al., 2016), that envisions accessing data in an uncomplicated fashion, so that e-scientists can entirely focus on the domain of their expertise, and not on side tasks, such as curating datasets. By the term *e-science* we refer to a "global collaboration in key areas of science" (Hey and Trefethen, 2003) which "promotes innovation in collaborative, computationally- or data-intensive research across all disciplines, throughout the research lifecycle" (International Confer, 2018). Based on our experience, transforming environmental datasets from different sources, in order to fit as input to environmental models requires manual work which is hardly re-usable. For example, different programming languages (e.g. Python (Van Rossum and Drake, 2003), R (Ihaka and Gentleman, 1996), etc.) and data models (e.g. O&M, WaterML 2.0, etc.) are adopted for the *scripting* and *environmental data management framework* approaches, respectively.

In this paper, we outline the design and demonstrate an open source implementation of the Environmental Data Acquisition Module (*EDAM*), that addresses issues of syntactic data heterogeneity using templates. An *EDAM* template is an abstract representation of a data file's contents using programming language-agnostic semantics. *EDAM* supports data acquisition, integration and transformation from a variety of file types and syntaxes through templates. Specifically, *EDAM* is applicable for environmental timeseries datasets stored in various data formats (delimiter-separated files, flat files, etc), at various sources (files, folders, databases, websites), and implementing different data models (tables, key-value pairs).

EDAM employs a declarative approach to enable scientists to annotate their data by means of templates. It automatically parses the data, matches them with templates, stores them and optionally exports them to a format described by an output template. This allows for end-users to query, retrieve, and transform environmental timeseries datasets into their own formats. Also, *EDAM* supports interoperable data dissemination through standardized protocols (e.g. OGC SOS (Broering et al., 2012)). We also demonstrate its front-end graphical user interface (GUI) for creating maps.

We demonstrate *EDAM* in seven cases studies from various environmental domains, including air quality, meteorology, agriculture and hydrology. To the best of our knowledge, this is the first time that structural templates are extensively used for environmental data management tasks (i.e. acquisition, integration and dissemination). We started exploring this approach in (Papoutsoglou et al., 2015), where we investigated a case study for collecting data from a smoky Swiss railway station. Here we extend our work with six more real-world cases:

- scraping meteorological data from the public webpages of the Bureau of Meteorology (BoM) in Australia and the UK Met Office,
- parsing hydrological timeseries data from the Hydrological Observatory of Athens (HOA),
- extracting observations from an air quality archive from BoM, originally stored in a relational database,
- aggregating historical timeseries data from all Dutch weather stations, provided by Koninklijk Nederlands Meteorologisch Instituut (KNMI),
- transforming weather input data of APSIM crop model (Holzworth et al., 2014) into the AgMIP format (Porter et al., 2014).

The rest of the paper is structured as follows: In Section 2 we review contemporary approaches for environmental data acquisition and integration, and introduce readers to environmental data management with web template frameworks. Section 3 presents the *EDAM* architecture; specifically: key requirements, user types, and use scenarios. Section 4 demonstrates *EDAM*, the conducted experiments, the used datasets and the addressed challenges. Finally, in Section 5 we discuss our research findings and lessons learned, conclude the research summarizing key findings and future work.

2. Background and related work

In the environmental data literature, different terms are used for describing the process of obtaining a dataset and transforming it into another format. Specifically the terms: *harmonization* (Porter et al., 2014), *mediation and conversion* (Horsburgh et al., 2011), *management and publication* (Jones et al., 2015), *integration* (Beran et al., 2009), *acquisition and collation* (Mason et al., 2014) and *wrangling* (Terrizzano et al., 2015; Kandel et al., 2011) are synonyms for data acquisition and integration.

In this work, we focus on acquisition and integration of environmental *timeseries* data. In general, the acquisition process works as follows: A *station*, stationary or not, houses one or more *sensors*. A *sensor* measures one or more *observable(s)* producing observations. An observation has a *value* expressed in some units, and refers to a certain *timestamp*, and possibly a location. In this context, environmental observations without a *temporal dimension* (e.g. soil data) are not considered timeseries and thus can not be processed by *EDAM*. Also note that *EDAM* can process location data when they are associated with a timeseries (i.e. observations of latitude, longitude, angle, etc, at a certain timestamp). Location data are stored as regular timeseries and can be combined with other observations. This is elaborated further in subsection 4.1).

In the rest of this section we review approaches that cope with syntactic heterogeneity. First, we present environmental data management frameworks which by design account for syntactic interoperability. Environmental data management frameworks are typically used for preparing inputs required for executing scientific workflows, decision support tools or environmental models. However, not all environmental datasets are offered through such frameworks. Second, in Subsection 2.2 we present the scripting approach which facilitates environmental data transformation to fit into a consistent data format. Last, Subsection 2.3 introduces web template frameworks and presents our previous experiences with them.

2.1. Environmental data management frameworks

Providing standardised discovery and access services for environmental data is a key requirement for an environmental data management framework (Horsburgh et al., 2011). Examples of such frameworks are the OGC Sensor Web Enablement (SWE), which supports timeseries dissemination through the Sensor Observation Service (SOS) (Broering et al., 2012), and the CUAHSI Hydrologic Information System (HIS) (Horsburgh et al., 2009). Both, provide interoperable data access on two layers: a) communication, b) data representation. Communication is achieved by defining standardized ways to request environmental data (e.g. `GetValues` for CUAHSI-HIS (Ames et al., 2012), `GetObservation` for OGC SOS (Broering et al., 2012)). Data representation deals with data dissemination through standardized information models, which hide the underlying data complexity. For example WaterML 2.0 (Taylor, 2014) is promoted by both frameworks in order to represent hydrological timeseries data.

Environmental data management frameworks can provide

interoperable access to raw data by transforming them to a common data model. This common data model can be part of the framework, or its implementation. In the case of OGC SOS there are different software implementations which use different data models (McFerren et al., 2009). On the other hand, CUAHSI-HIS is founded around the Observations Data Model (Horsburgh et al., 2008). Software tools were implemented to import data into an ODM database. Horsburgh and Tarboton (2007) document a data loader component which imports tabular timeseries into an ODM instance. Mason et al. (2014) present an environmental management framework which utilizes reusable data parsing templates to annotate tabular timeseries and import them into an ODM instance.

2.2. Data integration through scripting

Several efforts are reported in the literature where scripts have been used for environmental timeseries acquisition and integration. By the term *script*, we refer to a small computer program which is intended to automate a task, regardless of whether the programming language in which it was developed is considered a scripting language (e.g. Python) or not (e.g. Java). For example, the Ag-Analytics platform (Woodard, 2016) demonstrates a data warehouse to retrieve data from heterogeneous data sources. It extracts data through custom scripts written in Python, one for every data source. In another example, Harth et al. (2013) employ a Linked Data scripting language, called Data-fu (Stadtmüller et al., 2013), to integrate diverse data sources. Each Data-fu program comes with data source specific rules and queries. In a third line of work, Porter et al. in (Porter et al., 2014) present a data harmonization workflow to promote model inter-comparison and ensemble modelling. Data source specific *translators* were developed and used to integrate heterogeneous datasets into the AgMIP common data schema, in order to facilitate data exchange between crop models.

2.3. Environmental data management with web template systems

Web template systems are designed to create dynamic content and are extensively used in web applications. They are used for automatically generating custom content, such as customer invoices, search results, data reports, etc. Web template systems (e.g. Jinja2 (Ronacher, 2008), Mako (Bayer), Cheetah3 (Broytman and Croy, 2001)) are intuitive to use, and do not require advanced programming skills. Each one comes with a template language, which is used to markup templates. A template is a document which represents a data structure using variables (Geebelen et al., 2008). Dynamic views are rendered by feeding a template with data, and template variables are substituted with values.

Web template systems can support data output by design, but not data input directly. For example in (Samourkasidis and Athanasiadis, 2017), we employed Jinja2 to create on-the-fly dynamic views for environmental data dissemination. In a previous work (Papoutsoglou et al., 2015), we also started experimenting with using template files as a markup for data input, where we presented a platform which used templates to read from local files in a variety of formats.

2.4. Summary

Acquiring and integrating environmental timeseries in a consistent data format is a manual process which requires significant efforts. This is because the vast majority of environmental datasets available in the Environmental Internet of Things (EIoT) are heterogeneous by nature (Hart and Martinez, 2015). Universal data acquisition and integration can be achieved through the scripting approach. Nevertheless, there is a trade-off between generality and complexity. This approach opposes the *lowering e-science barriers*, since it presumes a computer science background (Swain et al., 2016). A web template framework language is much more simple compared to a traditional programming language. In this work, we investigate the use of templates in order to acquire and integrate environmental timeseries datasets, and seek for a compromise in the trade-off between complexity and generality.

3. The EDAM framework

3.1. Objectives

There were three objectives in designing and developing *EDAM*. The first was to *lower e-science barriers* by embracing a *programming language-agnostic* solution. Obtaining timeseries data by writing small computer programs (scripts) has already been investigated (see Subsection 2.2). Thus, we focused on solutions that involve as little as possible programming skills for its end-users, and examine the use of templates written with a simple, programming language-agnostic markup.

The second objective was to apply *EDAM* to a wide variety of case studies, in order to tackle the intrinsic heterogeneity of environmental data sources. This heterogeneity is related to a) data source type (which could be text files, webpages, databases, web services), b) data formats (i.e. comma-separated values (CSV), tab-separated values (TSV), etc.), and c) data models after which available environmental data are structured.

The third objective was to create custom views of timeseries data and disseminate them through standard interoperable protocols, as OGC SWE standards. This enables users to transform data from one format to another, promoting interoperability for environmental modelling and overcoming problems related to the diversity of data models. It also copes with syntactic interoperability by exposing *EDAM*-processed datasets via established information models such as O&M and SensorML.

3.2. Abstract architectural design

There are three key-components involved in *EDAM*: a) input files, b) template files, and c) template engine. Fig. 1 depicts the interaction between *EDAM* components for data input and output. In all cases, the data are extracted from their original source and stored in the *EDAM* database in a unified data model. Then, they can be fetched and presented in a user-defined way using a range of custom templates.

Any kind of text-based source can serve as an input. Inputs are stored in one or more files, locally or remotely. They may be stored in a local nested folder structure, on a website or relational databases from which data could be extracted with SQL queries.

An *EDAM* template is an abstract representation of data file contents. Each template file is bound to a specific data syntax, which is comprised of:

a) a timestamp, which may come in different formats (as we discuss below),
b) a set of observables in a given order, along with optional metadata annotating their semantics. Omitting observables, changing their order of appearance and/or changing timestamp representation results in a different data syntax, i.e. requires a different template. We envision that one template will be needed per sensor vendor, or legacy data formats used for input/output by environmental models. Templates can be used for specifying both input or output data file structures, and are written using the *EDAM* template language.

The *EDAM* template engine and language are the core of the framework, offering various processing capabilities. The template language itself is founded on programming language-agnostic semantics. Besides simple data parsing, the *EDAM* template engine supports mathematical and statistical operations. Both the template engine and language are implemented after Jinja2 (Ronacher, 2008). This enables us to use the Jinja2 mature framework for data dissemination purposes.

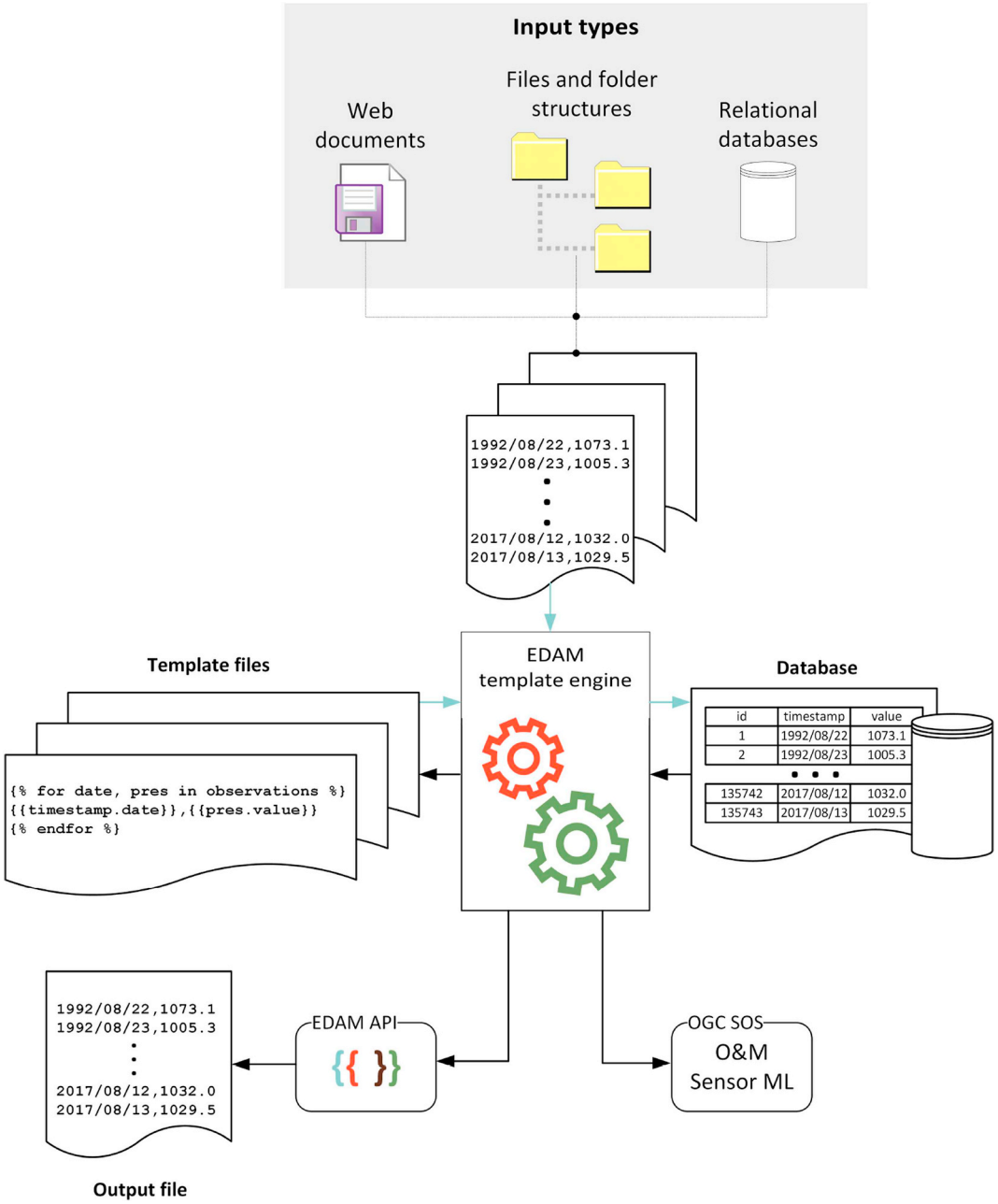


Fig. 1. *EDAM* abstract architectural design. Black and blue arrows depict output and input workflows, respectively. *EDAM* supports data transformation through its API, and standardized data dissemination through OGC SOS. For the depicted example, input and output files are identical, since the same template file was used for the respective processes. (For interpretation of the references to color in this figure legend, the reader is referred to the Web version of this article.)

Regarding data dissemination *EDAM* may offer acquired data as services on the web. Currently, *EDAM* supports data dissemination through OGC Sensor Observation Service, and its own *EDAM* API. The *EDAM* API enables the creation of custom data views, since *EDAM* templates can be called dynamically.

EDAM template language artefacts (keywords or user-defined variables) are located inside *placeholders* (`{{}}`). The *EDAM* template language has four restricted keywords: `station`, `observable`, `sensor`, `timestamp`, which result from the *EDAM* underlying data model. Fig. 2 depicts the *EDAM* unified data model along with the template language restricted keywords. The data model was designed after our assumption of an environmental data source, and it is tailored to the needs of the template language. This is also the reason why we did not reuse any third-party data model. A third-party data model involves a number of external dependencies via foreign keys that would affect the template language syntax, rendering it complex and difficult to use.

User-defined variables are used to annotate the observables found in a dataset. Their semantics are further specified in a metadata file. A template may contain control statements (e.g. if-then-else, for-loops) to provide formatting and control functionality, and set the logic which will be used for data retrieval.

Next to the template file, there is the metadata file. It is drafted by users in order to further annotate data parsed from input files. Metadata include information commonly not stored directly in the original input files, as for example units of measurement for observables, or station locations. Such additional metadata, which may include terms from ontologies, are necessary for enriching the semantics of the original data. How this works is further detailed in the following section.

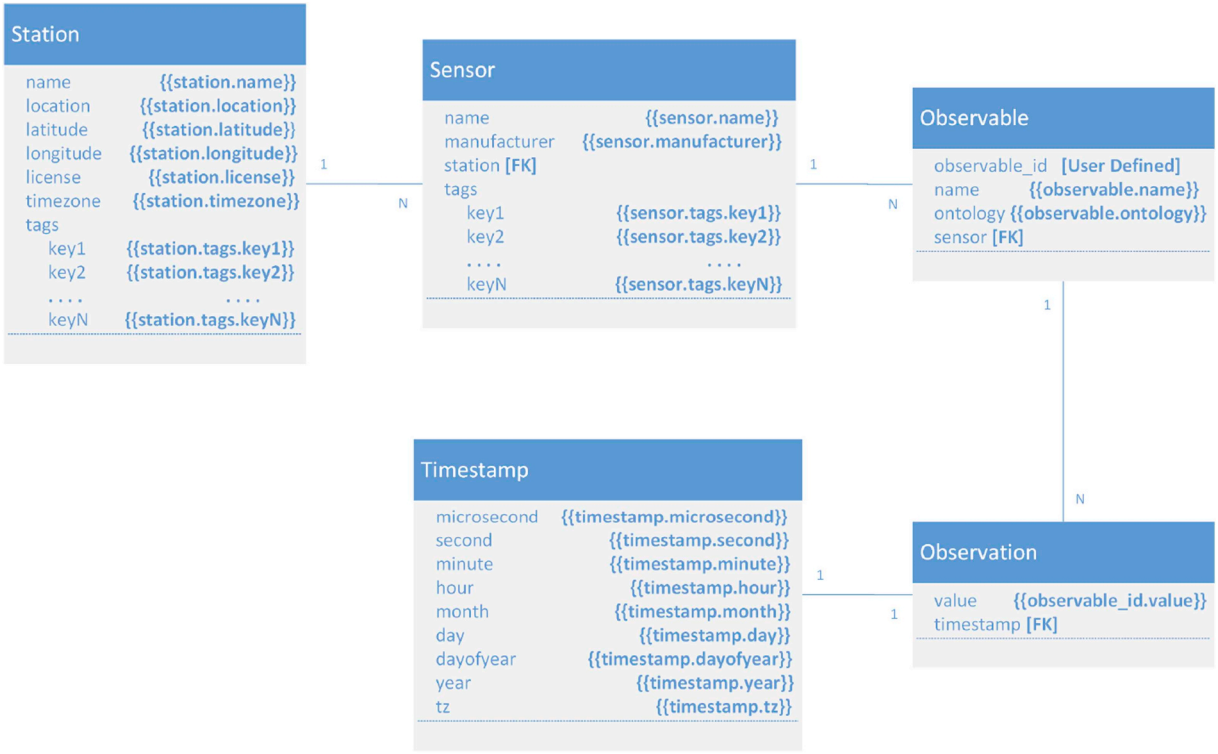


Fig. 2. *EDAM* unified data model. A *station* houses a number of *sensors* which measure *observables* and produce *observations*. An *observation* has a value for a given *timestamp*. Data curators define `observable_ids` which represent *observables*.

3.3. Workflow

The *EDAM* workflow operates in two phases: *data input* and *data output*. *Data input* concerns data acquisition, preprocessing, and storage processes. *Data output* involves the discovery, transformation and dissemination of information. Fig. 3 depicts the workflow for data input and output, accordingly. We identify two user roles in the *EDAM* system:

Data curators are interested in sharing data with *EDAM* added-value services, and import datasets into the system. They draft *input* templates making new data sources available.

Data consumers are e-scientists (i.e. modellers, researchers, decision makers) who are interested in third party data stored in *EDAM*. They use *EDAM* to a) view available datasets, b) render graphs, c) apply filters on data and d) download them in various formats (i.e. csv, txt, etc.). They may create custom data views by *editing* template files (e.g. change the order of columns, omit columns, etc.). Software agents can also be considered as *data consumers*. They interact with the system using OGC SOS, or the *EDAM* API.

A workflow to input data into *EDAM* is as follows: A *data curator* drafts the *input template*, documents all relevant metadata in a *metadata file*, and finally provides a data source. An *input template* has the original data source structure. *Data curators* may provide *EDAM* with metadata using the *metadata files*, to augment the original information with additional details about the *station*, the involved *observables*, and their corresponding *sensors* and *units of measurement*. While this step is optional, it is critical towards data reusability and interoperability.

Fig. 4 shows a sample input file from the UK Met Office (Fig. 4a), and the corresponding template file (Fig. 4b). *EDAM* keyword `{{station}}` has been used for annotating all data relevant to the station name and location. The keyword `{{timestamp}}` is used to parse the component of the date to which the observations correspond to. In this case, we used `{{timestamp.year}}` and `{{timestamp.month}}` to parse the year and the month respectively. Generic Jinja2 keywords, such as `{%for %}` are used to parse all data reported in the file. User-defined keywords are used as variable names to annotate observable values, as `{{tmax.value}}`, `{{tmin.value}}`, `{{af.value}}`, `{{rain.value}}`, and `{{sun.value}}`. Fig. 5 depicts the corresponding metadata file, which defines additional station and observable metadata. Data curators can specify the timezone (station attribute), which will be used to complement all the station related timestamps. The value of the timezone attribute can be either the format as code keyword, or a timezone from the *tz database* (Wikipedia contributors and L, 2018). In case the format as code keyword is used, *EDAM* automatically identifies the corresponding timezone from the station's location and assigns it to the related timeseries. Greenwich Mean Time (GMT) is the default timezone, which is used when no location is provided, or the timezone attribute in the metadata file is omitted. Data curators also use the metadata file to relate a user-defined keyword (e.g. `tmax`) with a) its corresponding observable name (e.g. *Temperature Maximum*) and b) the unit it was reported (e.g. *Celsius*). There is also a section to store metadata about the utilized sensors, which in this example are unknown.

3.4. Implementation and modes of operation

In Table 1 we depict *EDAM* implemented functions organized by *when* they are utilized. *Input* functions are applied by *EDAM* during the process of *data input*. *Processing* functions concern statistical and conditional filters which are written in *output templates* and are applied during *data output*. Last but not least, *dissemination* functions are added-value services offered for *EDAM*-imported datasets.

EDAM software has been developed in Python, and is available as open-source software on GitHub (Samourkasids et al., 2018) under the GNU Affero General Public License Version 3. It is also distributed as an autonomous Python package through the Python Package Index (pip) (Python Software Foundation, 2018), and can be installed on a computer with Python installed by typing `pip install edam`. The pandas Python library (McKinney, 2011) supports the *EDAM input* and *processing* functions. The *EDAM dissemination* functions are implemented with

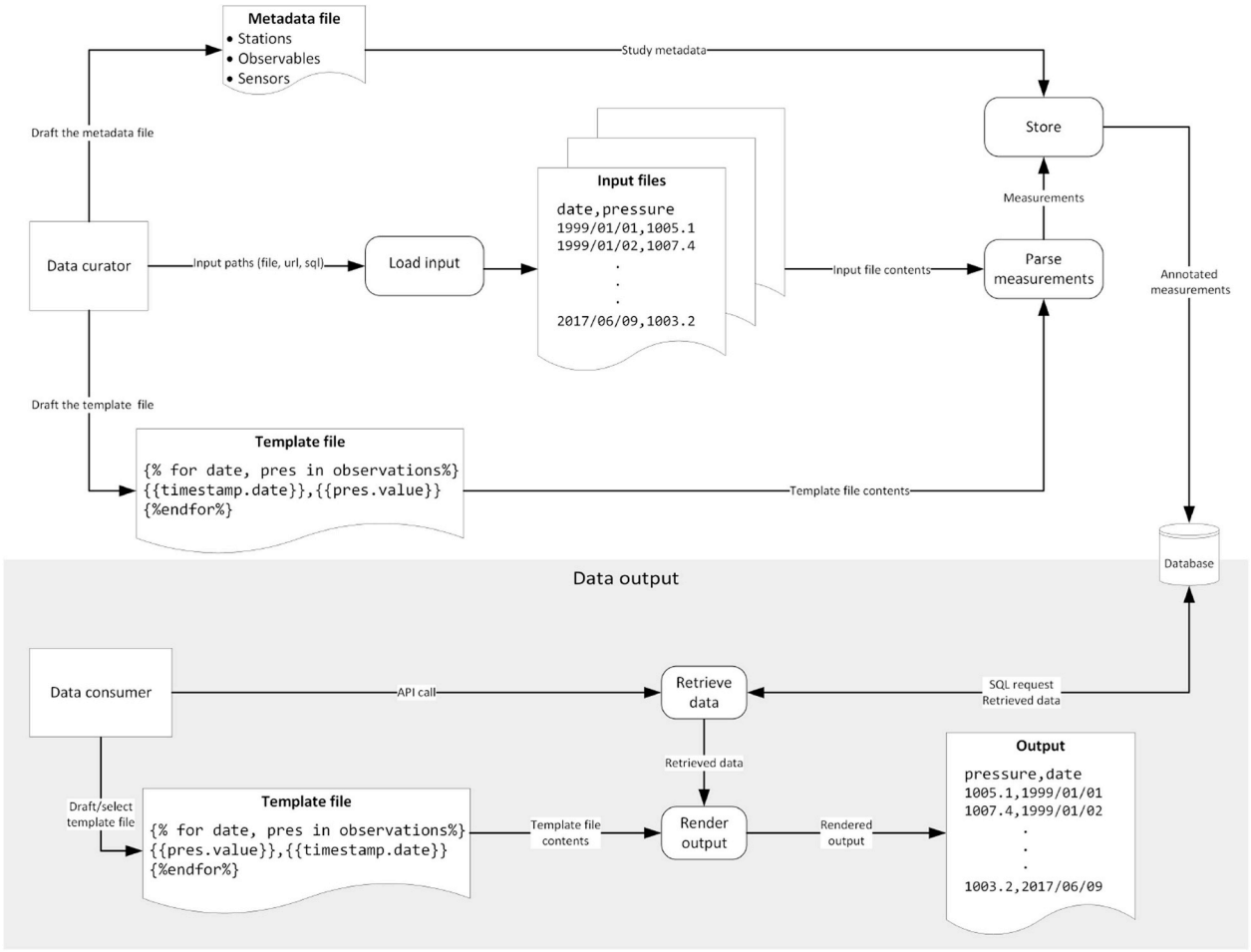


Fig. 3. *EDAM* workflow. Upper part depicts *data input*, and lower part *data output*. The database component belongs to both. The output template file differs from the input one, as the order of variables is reverted. This is reflected on the *output* data structure.

```
Durham
Location: 426700E 541500N, Lat 54.768 Lon -1.585, 102 metres amsl
Missing data (more than 2 days missing in month) is marked by  ---.
   yyyy  mm   tmax    tmin      af    rain     sun
              degC    degC    days      mm   hours
   1880   1    4.2    -1.4      22    13.5     ---
   1880   2    8.7     0.6      12    44.3     ---
   1880   3    9.2     1.0      12    32.5     ---
```

(a) Original input file

```
{{station.name}}
Location:{{station.location}}, Lat {{station.latitude}} Lon {{station.longitude}}, {{station.tags.altitude}}
Missing data (more than 2 days missing in month) is marked by  ---.
   yyyy  mm   tmax    tmin      af    rain     sun
              degC    degC    days      mm   hours
{%for timestamp, tmax,tmin,af,rain,sun in station.data%}
{{timestamp.year}}  {{timestamp.month}}  {{tmax.value}}  {{tmin.value}}  {{af.value}}  {{rain.value}}  {{sun.value}}
{%endfor%}
```

(b) Input template

Fig. 4. Acquiring meteorological data from a dataset inspired by UK Met Office. (a) depicts the input file and (b) its corresponding template file.

```
Station:
    license: Attribution
    region: United Kingdom
    url: http://www.metoffice.gov.uk/
    timezone: Europe/London
    tags:
        key1:value1
Observables:
    - observable_id: tmin
      name: Temperature Minimum
    - observable_id: tmax
      name: Temperature Maximum
    - observable_id: rain
      name: Rain
    - observable_id: af
      name: Days of air frost
    - observable_id: sun
      name: Sunshine duration
Units of measurement:
    - name: Celsius
      symbol: C
      relevant_observables: tmin, tmax
    - name: Days
      symbol: D
      relevant_observables: af, sun
    - name: Millimeters
      symbol: mm
      relevant_observables: rain
Sensors:
    - name: Generic sensor
      manufacturer: Unknown
      relevant_observables: tmin,tmax,af,sun,rain
      tags:
            generic: True
            analog: False
```

Figure 5. The metadata file for the input dataset reported in Fig. 4b. There are four sections (*Station*, *Observables*, *Units of measurement*, *Sensors*), under which *data curators* define metadata. This is where an `observable_id` is defined and related with its corresponding observable name. Users also reference these `observable_ids` to relate an observable with the relative unit of measurement and sensor. The sensors utilized in this study are unknown and thus are defined as *Generic*. The same result would be produced in case the *Sensors* section was omitted.

the Flask web framework (Ronacher, 2010), and acquired timeseries are offered as OGC SOS services through the Python implementation reported in (Samourkasidis and Athanasiadis, 2017). The hardware requirements of *EDAM* are minimal. We installed *EDAM* and tested its functionalities on a Raspberry Pi 2 Model B mini-computer (Raspberry Pi Foundation, 2018) without any issues.

EDAM operates as a local standalone system. This means, that *EDAM*-parsed datasets are stored and can be accessed locally on user's computer. Installation automatically creates a folder in the home directory, in which the user should store *templates* and *metadata files*.

After installing *EDAM* two commands are available via the command line: `edam` and `viewer`. These commands reflect the two distinct modes of operation: the command-line mode and the graphical user interface mode.

In the command line mode, *data curators* utilize the `edam` command to define the *input arguments* (i.e. input, template and metadata file), in order to parse and store a dataset. Optionally, they can define the *output parameters* (i.e. template and metadata file), in order to transform a dataset on-the-fly.

In the Graphical User Interface (*GUI*) mode, we assume that some datasets have already been imported in *EDAM*'s database and the user wants to disseminate them via the *EDAM* web services. The `viewer` command starts the *EDAM* web services, which currently are the API, OGC SOS and the web front-end. Human users can access these services on their browser, and machines via the appropriate protocol. Note that the web front-end includes information about the *EDAM* API, how to access the stored datasets, and the OGC SOS instance.

Table 1
EDAM functions distinguished by when they are applied. *Input* functions are applied by EDAM during data input. *Processing* functions are placed on *output* templates. *Dissemination* services are automatically offered for EDAM acquired datasets.

#	Function	Description
Input	URI generation (**I1**)	Generate URIs based on a pattern. Each URI represents a data source, either online (i.e. URL) or a file (i.e. URI)
	Online parsing (**I2**)	Acquire online data sources via a URL
	File parsing (**I3**)	Acquire text data sources via a URI
	Database parsing (**I4**)	Acquire data sources from a relational database via a connection string and an SQL query
	Folder exploration (**I5**)	Navigate through folders and utilize **I3** feature
	Metadata curation (**I6**)	Update station or observable with metadata found on timeseries resource (file or online source)
	Conditional filtering (**I7**)	Input a data point based on a condition. It can be used for QA/QC purposes
	Timestamp assembly (**I8**)	Construct a timestamp out of many components (i.e. day, month, year, hour). It supports for complex timestamp components (i.e. julian dates and years)
	Relationship establishment (**I9**)	Resolves a relation between a data point and its related metadata. This function resembles the functionality of Foreign Keys in relational databases
	Timeseries merging (**I10**)	Associate timeseries of a station, which are originally offered as multiple ones
Processing	Resampling (**P1**)	Upsample or downsample timeseries data. Resampling is performed upon a user-selected aggregation or interpolation method*
	Summarization (**P2**)	Generate a summary of statistical values for timeseries data. The summary concerns: *count*, *mean*, *std*, *min*, *25%*, *50%*, *75%*, *max*
	Conditional export (**P3**)	Similar to **I7**, it facilitates QA/QC
Dissemination	Map projection (**D1**)	Stations are projected on a map based on location metadata. Should they not be provided, EDAM attempts to estimate them via station name.
	OGC SOS (**D2**)	Acquired datasets are offered as services through OGC SOS
	Data transformation (**D3**)	A dataset can be exported with a different template. This feature is available in cases where the output template is compatible with the dataset**

* This uses the resample function of the pandas library.
** In order for a template to be compatible, it should contain the same *observable_id* with the requested dataset. *Generic templates* are by-design compatible.

Table 2
The seven data sources we selected to evaluate *EDAM*. They are distinguished based on a) how they are available (*Source)*, b) how are they modelled (*Data model*), c) the preamble type (*Preamble*), and d) whether related metadata are included in the dataset or not (*Metadata*). **External** metadata are declared by *data curators* in metadata files.

Datasets	Source				Data model			Preamble		Metadata	
	file	folder	http	database	tabular	abstract	other	tabular	key-value	included	external
AgMIP	✓				✓						✓
APSIM	✓				✓				✓	✓	✓
BoM (Met)			✓				✓[a]		✓	✓	✓
UK Met			✓		✓				✓	✓	✓
KNMI	✓				✓			✓		✓	✓
Swiss TPH		✓			✓				✓	✓	✓
HOA			✓			✓			✓	✓	✓
BoM (Air)				✓		✓				✓	

[a] BoM (Met) data model has a *tabular*-like format. This is why some observables are repeated in more than one *columns*.

4. Demonstration

We demonstrate *EDAM* extended outreach by acquiring environmental timeseries data from diverse data sources. In Table 2 we name the seven sources we identified. Each of them poses a different challenge: a) timeseries with complex timestamp structures in custom formats and datasets which have essential *metadata* in their *preamble* (APSIM, AgMIP), b) *online* datasets having a simple timeseries structure (UK Met Office), or a more complex one (BoM (Met)), c) datasets stored in one *file* (KNMI) or dispersed in multiple files within folders (Swiss TPH), and d) *abstract data models* applied to text files (HOA) and *relational databases* (BoM (Air)).

In Subsection 4.1 we describe the case studies against which we evaluated *EDAM*. We also highlight challenges associated with each dataset. These challenges were addressed by employing *EDAM input* functions during the development of the input templates. Table 3 presents the exact functions used to cope with challenges for each case study. Besides timeseries data, storing corresponding metadata is an essential requirement for *EDAM*. The `metadata curation (I6)` function was applied on every dataset.

For all case studies we developed *EDAM* templates as needed and successfully parsed the datasets using a single *EDAM* command. The developed templates are available as Supplementary Material A, and also on the *EDAM* GitHub repository, along with detailed instructions on how to repeat the experiments, with the exception of Swiss TPH and BoM as original data are not publicly available.

4.1. Test cases

4.1.1. AgMIP and APSIM weather data files

The Agricultural Model Intercomparison and Improvement Project (AgMIP) (Porter et al., 2014) have brought agricultural model data sharing into the spotlight. Within AgMIP, various agricultural model data inputs and outputs (such as the APSIM (Keating et al., 2003)) were transformed into the common AgMIP data scheme. Here we worked only with the weather data files.

Note that, AgMIP and APSIM data files use different *timestamp formats*. APSIM uses *days of year* and *years*, while AgMIP timestamp is represented through *year*, *month*, *date* components. We addressed the challenge of composing these into one universal timestamp with the `timestamp assembly (I8)` function.

Another challenge was related to metadata encoded in the preamble of APSIM data files. The APSIM weather file includes station metadata above the timeseries data, such as station name, location and others. We addressed this challenge of extracting metadata from the preamble with the `metadata curation (I6)` function.

Table 3
Input functions utilization for each *EDAM* test case.

Datasets	Input functions									
	I1	I2	I3	I4	I5	I6	I7	I8	I9	I10
AgMIP			✓					✓		
APSIM			✓			✓		✓		
UK Met	✓	✓				✓	✓	✓		
BoM (Met)	✓	✓				✓		✓[a]		✓
KNMI			✓			✓			✓	
Swiss TPH			✓[b]		✓	✓				✓
HOA	✓	✓				✓				✓
BoM (Air)				✓					✓	

[a] According to BoM (Met) data model the timestamp of certain data points is projected on their corresponding header column.

[b] The observables of a Swiss TPH station are formatted differently. Thus, parsing is accomplished through more than one *template* files.

4.1.2. UK Meteorological Office

In the context of Open Data, the UK Meteorological Office reports historical observations of 27 weather stations. For every station, monthly observations are stored in one text document. New observations are appended every month and each weather station can be found on a certain web location. They follow the pattern: http://www.metoffice.gov.uk/pub/data/weather/uk/climate/stationdata/{station_name}/data.txt, where {station name} is replaced with an actual station name.

Data points reported have special markers for the quality of the reported values. Markers are weakly defined in each document preamble. For example, estimated data is marked with a $*$ after the value and missing values are represented through the $-$ notation. Such markers make it difficult to parse and reuse the data directly. Capturing such observation-specific quality attributes is a further challenge that *EDAM* in its current version does not support. We used the `conditional filtering (I7)` function in order to filter out the missing values.

4.1.3. Australian Bureau of Meteorology (meteorological datasets)

The Bureau of Meteorology (BoM) in Australia offers historical meteorological timeseries for a number of weather stations across Australia. They concern *daily* observations which are published every month as HTML and CSV documents with the same structure. Users can access the timeseries by crafting URLs which comprise information about the requested station id, and month/year. For example, the URL for the meteorological data about Adelaide station (*5002* station id) for *October 2018* is:http://www.bom.gov.au/climate/dwo/201810/text/IDCJDW5002.201810.csv

The challenge in acquiring BoM timeseries is in regard to their structure. It is a common practice in delimiter-separated files that every row corresponds to one observation for a given timestamp. However, each BoM row reports *two observations* for the *same* daily timestamp. These two observations report the same measured quantity at different

times on the same day. Thus the sampling hour, needs to complement the daily timestamp for the bi-daily observations, is included in the dataset's header. We addressed this challenge with the `timestamp assembly (I8)` function. We also utilized custom Jinja2 macros in order to support this type of tabular timeseries.

4.1.4. Koninklijk Nederlands Meteorologisch Instituut (KNMI)

The Royal Netherlands Meteorological Institute (KNMI) provides weather services for the Netherlands. They offer historical observations as text documents. For our study, we parsed historical observations from 37 Dutch weather stations from 1901 to 2016. Each weather station reports daily observations for 39 observables. The dataset comes as a whole in a single text file of 158 MB, which includes metadata in the preamble.

The challenge here was to separate the metadata from the timeseries using templates. We addressed this challenge by utilizing `metadata curation (I6)` and `relationship establishment (I9)` functions.

4.1.5. Swiss Tropical and Public Health Institute

The Swiss Tropical and Public Health Institute (TPH) monitors air quality in train stations among others. Both stationary and moving stations are used, consisting of multiple sensing units. Each sensing instrument exports its measurements in a file in a sensor-dependant format. All station-related files are stored in a folder structure. Additionally, there are multiple data formats associated with a station as different sensor types are involved in the various studies. For example, the GPS sensor exports its readings in a file with seven columns (date, time, latitude, longitude, speed, bearing, altitude).

The challenge with the Swiss datasets is related to the aforementioned folder-tree structure. Not all file types are present in all folders, so *EDAM* is challenged to match the various files found against several templates in order to extract observations. We addressed this challenge by navigating folders with the `folder exploration (I5)` function, and matching files with the corresponding templates with the `file parsing (I3)` function. We associated the different datasets to the corresponding station with the `timeseries merging (I19)` function. The other challenge was to combine location data with the other observations into one output file. Specifically, each sensor took observations at different time intervals. *EDAM* automatically solves the issue by combining together observations sharing the same timestamp. In a data fusion scenario, output templates could be used for homogenising the reporting timestamps of the various sensors involved in a study.

4.1.6. Hydrological Observatory of Athens

The Hydrological Observatory of Athens (HOA) offers a service endpoint for hydrological timeseries. Several observed properties are reported for 23 stations. Each of them is offered separately on the web, and every observed property dataset has a unique URL. Timeseries are reported under the same *abstract format*, consisting of a preamble with relevant metadata (i.e. about the station, observed property, unit of measurement, etc), and the actual timeseries in the form of key-value pairs.

The challenge in acquiring HOA timeseries concerns the abstract data format. In non-abstract data formats a given file column corresponds to a certain observable, which is mentioned in the header. In contrast, the HOA abstract data format mentions the observed property at the preamble of each document. We addressed this challenge by drafting an abstract input template. The specific `observable_id` was defined dynamically based on the metadata found in the document preamble.

Again, here each station reports several files, one for each observable, but all have exactly the same format. Instead of drafting as many templates as the available observed properties we use a generic template that includes the `observable_id`. In a data fusion scenario, output templates can be used for linking together the various observables of the same station.

4.1.7. Australian Bureau of Meteorology (air quality dataset)

BoM developed a historical database that contains hourly air quality timeseries in several locations in Australia. We were given access to this PostgreSQL database that contains data of common pollutants as SO_2, O_3, CO, NO_2 and PM_{10}, in 99 stations and corresponds to a period of 20 years (1988–2008). In total, there are about 15 million records. Observations are stored in key-value pairs, with detailed metadata about the stations, and observed quantities. Metadata are stored in a different relational database system (i.e. Oracle). In total, there are four tables in this implementation. The challenge in acquiring these timeseries lies in the relational databases and the chosen structure. Data and metadata are stored in different tables across different database systems. In the observations table, each observation is associated with the corresponding station. In the station metadata (i.e. name, location, altitude) table, each station is referenced with the aforementioned identifier. We addressed the challenge of realizing the external relationships so data are appropriately linked when harvested, with the `relationship establishment (I9)` function.

4.2. Demonstrating EDAM output

With regard to dissemination services, an example output of *EDAM* is shown in Fig. 6. We demonstrate *EDAM* acquisition and integration for all Australian weather stations for July 2017. Specifically, *EDAM* utilizes `URI generation (I1)` to discover 478 BoM stations. Employing `online parsing (I2)`, and using one *template* for all stations, *EDAM* acquired and stored approximately 210,000 data points. Above operations were realized through a *single EDAM* command, that looks like:

```
edam -input "http://www.bom.gov.au/climate/dwo/201707/text/IDCJDW%7b2-8%7d0%7b01-82%7d.201707.csv" -template bom.tmpl -metadata bom.yaml
```

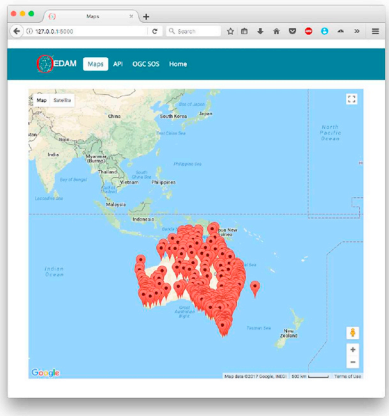
EDAM processing capabilities are statistical filters and conditional exports. Fig. 7 exhibits them when applied to a UK Met dataset. Specifically, we aggregate daily into monthly observations with the `resampling (P1)` function. Consequently, we illustrate the `conditional export (P3)` function exporting only those datapoints which satisfy a given condition. *Processing functions* are typed in the *output template* files.

The `data transformation (D3)` function facilitates the dataset transformations from one format to another. We demonstrate this feature with the AgMIP dataset and the WebXTREME service (Klein et al., 2017). The latter is a web service which, given an input in a certain format, calculates extreme weather indicators. Transforming an *EDAM* curated dataset requires the draft of a template file for the target format. Fig. 8 demonstrates the creation of a custom data view, by simply drafting a new template file.

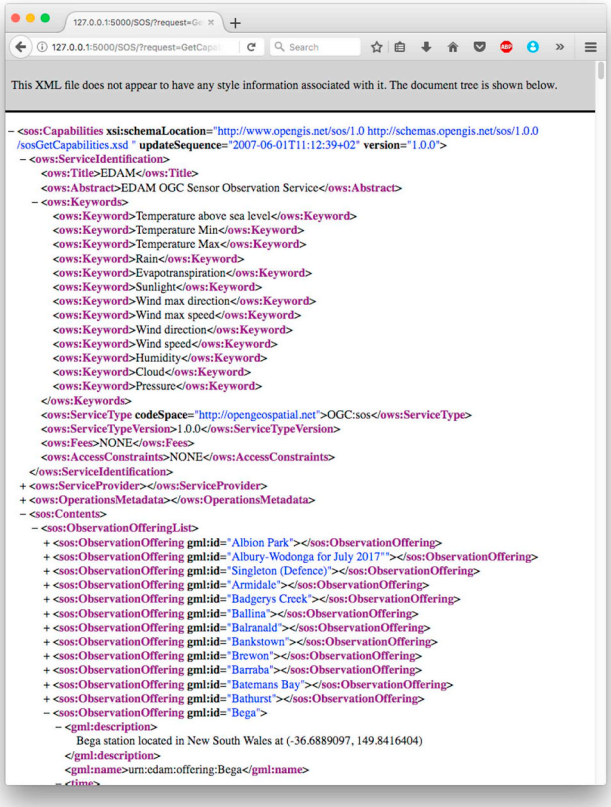
4.3. Lessons learned

While we aimed with this work to lower the barrier for e-scientists, we realized early that non-standard data formats usually lead to complex templates. This is due to the inherent complexities of environmental data domain, and the poor design choices that often come with legacy formats. The most complex data format we faced was BoM Met. In all other cases, each column represented a single observable. However, in the case of BoM Met the same observable was reported in two columns. Each column reported measurements taking place at different times in a day. The exact time each measurement was taken was noted in the header. Using *EDAM* functions and Jinja2 utility helpers, we successfully acquired and integrated BoM datasets.

Another factor which leads to complex templates is when metadata are mixed with timeseries data. This is the case with the KNMI dataset,



(a) Sample map projection with EDAM



(b) Sample OGC SOS output with EDAM

Fig. 6. Issuing a single *EDAM* command we parsed 478 online weather stations provided by BoM in Australia. Following parsing, these data are (a) projected on a map, (b) offered as services via OGC SOS.

where metadata about all stations and all their observables precede the observations.

Parsing HTML tables using templates was rather cumbersome. Initially we tried to parse BoM Met weather stations in their HTML form. However, HTML comprises numerous tags which provide an aesthetic view to the page (e.g. colors, aligns, fonts). These tags hinder the draft of a *reusable* template file, and render its composition a rather complex process. Thus, in its current release, *EDAM* cannot directly parse timeseries stored along with HTML tags, rather these should be stripped out as a pre-processing step.

There are also challenges in the way timestamps are represented in different datasets. *EDAM* provides users with an intuitive mechanism to annotate different timestamp components. Among the *EDAM* case studies we successfully parsed all different timestamp representations. In most of the datasets we experimented with, timestamp components were spread in more than one column and they were not in an ISO 8601 format. For instance, in APSIM weather files the timestamp is as ordinal date, comprised of two columns: The first one for the year and the second for the day of the year (Julian date). *EDAM* internally composes a universal timestamp object, so *data consumers* during data output can transform a timestamp in as many components as they want.

Timezone information is essential especially for spatially diverse datasets. Among all case studies, the timezone of the reported observations was explicitly reported only in one (HOA). All other datasets contained timezone information neither on the dataset nor on the corresponding metadata files. Interestingly, the BoM online portal which serves observations for the whole Australian continent, does not state the timezone in which observations are reported. *EDAM* is able to assign timezone information to datasets, either using station-level metadata or deriving it from the station geolocation. In cases where no timezone information is declared the GMT timezone is used.

While this is not a performance study, we measured some performance indicators. The KNMI dataset was the most voluminous dataset we parsed (158 MB). *EDAM* parsed and stored over 24 million datapoints in less than 8 minutes on a PC with 16 GB RAM. Nevertheless, volume is not the only constraint. In our attempt to discover BoM Met weather stations, *EDAM* generated and requested 574 unique URIs. From the 574 generated stations, 478 existed. Submitting the HTTP GET requests, reading the responses, and downloading the datasets took about 5 minutes. Station data were about 2 MB in total. Iterating through the 478 station timeseries and storing them took almost 11 minutes. In another example, the AgMIP dataset which consisted of one 1 MB file and more than 90,000 datapoints, was parsed and stored in about 2 seconds.

5. Discussion and conclusions

Today, environmental datasets are either available through interoperable environmental data management frameworks or can be found in raw, non-standardized formats. Both approaches require significant effort and usually a computer science background in order for data to be acquired, integrated and re-used. In this work we present *EDAM*, a template framework, as a universal strategy of acquiring and integrating diverse environmental *timeseries* data. *EDAM* copes with diversity in terms of data storage type (i.e. files, webpages, databases) and data format (i.e. relational, key-value pairs).

The *EDAM* data acquisition and integration capabilities have been investigated in the light of several test cases. Using *EDAM* we acquired and integrated datasets with different characteristics, demonstrating its generality. The evaluation of the software against timeseries with simple and more complex structure provides insights about the system's extended outreach. *EDAM* supports not only timeseries stored in files (as the template parsing files introduced in (Mason et al., 2014)), but also from webpages and relational databases.

Data transformation into consistent data formats and dissemination through standardized protocols is essential for syntactic interoperability in the IoT era. EDAM users can transform legacy environmental datasets between data formats by using *EDAM* templates. In this way, *EDAM* contributes towards a) environmental model re-usability by transforming data inputs/outputs in scientific workflows (Granell et al., 2010), and b) environmental data FAIRness as it facilitates timeseries *re-usability*, and *interoperability* and enhances *reproducibility* (Wilkinson et al., 2016). It also promotes further environmental data discovery and access through standardized dissemination protocols, i.e. the OGC Sensor Observation Service.

We consider that *EDAM* also contributes towards *lowering the e-*

```
{{station.name}}
Location:{{station.location}}, Lat {{station.latitude}} Lon {{station.longitude}}, {{station.tags.altitude}}
Missing data (more than 2 days missing in month) is marked by  ---.
   yyyy  mm   tmax    tmin      af    rain     sun
              degC    degC    days      mm   hours
{%for timestamp, tmax,tmin,af,rain,sun in   resample(station.data, 'Y', 'mean') %}
{% if tmax.value!="---" and tmin.value!="---" and af.value!="---" and rain.value !="---" and sun.value !="---" %}
   {{timestamp.year}}  {{timestamp.month}}   {{tmax.value}}    {{tmin.value}}      {{af.value}}    {{rain.value}}     {{sun.value}}
{%endif%}
{%endfor%}
```

(a) Example output template, highlighting aggregation and conditional output functions

```
Durham
Location: 426700E 541500N, Lat 54.768 Lon -1.585, 102 metres amsl
Missing data (more than 2 days missing in month) is marked by  ---.
   yyyy  mm   tmax    tmin      af    rain     sun
              degC    degC    days      mm   hours
   1890  12   12.408    4.633     5.917   54.883   105.667
   1891  12   11.767    4.017     6.667   51.975   104.35
   1892  12   11.125    3.342     8.583   59.508   107.1
```

(b) Resulting output file

Fig. 7. Demonstrating EDAM processing functions. (a) depicts the output template. `P1` function (blue color-box) down-samples monthly observations to yearly ('Y' argument), using `mean` aggregation method. `P3` function (magenta color-box) filters missing values (i.e. '—') from output. The resulting custom data view is depicted in (b). (For interpretation of the references to color in this figure legend, the reader is referred to the Web version of this article.)

```
@DATE    YYYY  MM  DD  SRAD  TMAX  TMIN  RAIN  WIND  DEWP  VPRS  RHUM
1980001  1980   1   1  15.0  26.0  12.2   0.0   1.4   4.8   8.6    25
1980002  1980   1   2   6.9  21.2   9.5   0.0   1.6   8.1  10.8    42
1980003  1980   1   3  10.7  22.2  14.7   8.0   1.5  11.9  14.0    52
```

(a) Original input file

```
@DATE    YYYY  MM  DD  SRAD  TMAX  TMIN  RAIN  WIND  DEWP  VPRS  RHUM
{%for timestamp, srad, tmax, tmin, rain, wind, dewp, vprs, rhum in station.data%}
1980001  {{timestamp.year}}   {{timestamp.month}}   {{timestamp.day}}  {{srad.value}}  {{tmax.value}}
 {{tmin.value}}   {{rain.value}}   {{wind.value}}   {{dewp.value}}   {{vprs.value}}    {{rhum.value}}
{%endfor%}
```

(b) Input template

```
DATE,AIRTMAX,AIRTMIN,RAIN
{%for timestamp, tmax, tmin, rain in station.data%}
{{timestamp.year}}-{{timestamp.month}}-{{timestamp.day}},{{tmax.value}},{{tmin.value}},{{rain.value}}
{%endfor%}
```

(c) Output template

```
DATE,AIRTMAX,AIRTMIN,RAIN
1980-1-1,26.0,12.2,0.0
1980-1-2,21.2,9.5,0.0
1980-1-3,22.2,14.7,8.0
1980-1-4,24.0,10.0,0.0
```

(d) Resulting output file

Figure 8. Data format transformation through a template. (b) was the template we used to read data from input file (a). Output template (c), creates a custom data view by changing the order and omitting some observables. The resulting output is depicted in (d).

science barriers (Swain et al., 2016). In contrast with most methodologies for acquiring EIoT datasets reported in the literature, *EDAM* does not presuppose a strong computer science background. We argue that templates offer a compromise between generality and complexity. The system is founded around a template language which uses programming language-agnostic semantics. Users are not required to have more programming skills than they already have in order to draft an *EDAM* template. As we demonstrated in Section 4, the templates drafted with *EDAM* language are reusable, and can be used for both data input and output.

The *EDAM* design embraces the open source principles, and allows for future extensions. On the processing layer, the system offers some pre-implemented processing functions which can be called by end-users. These support the *on-the-fly* calculation of values which were not originally stored in the database, and facilitate sensor data fusion, and/or aggregation. External users more advanced with computer science background can extend the system by defining such processing functions.

5.1. Future work

Future work may focus on issues related to semantic interoperability. *EDAM* supports metadata annotation of observables using ontologies. While these annotations are stored in the system, they are not fully utilized. In its current version, *EDAM* lacks a semantic layer to act upon datasets and templates. In principle, a dataset that was acquired through an input template can be transformed with another template only if both templates utilize the very same *observable_ids*. The *observable_ids* are drafted by data curators, and represent certain observables. Future work may investigate the use of a reasoner to resolve relations between the different *observable_ids*. In this way, a certain data file format can be represented through a single template, and by assigning synonym terms in an ontology we could enable automatic transformation into other formats.

Another direction for future work is to support environmental timeseries datasets in other formats. In this work we evaluated *EDAM* against text-based documents and relational databases. However, environmental datasets are also available in data cubes and non-relational databases. *EDAM* could be extended to support such other sources.

5.2. Conclusions

In this work we provided a proof-of-concept and a tested implementation of a template system that can be used for environmental timeseries acquisition and integration. We demonstrated that the use of templates for data acquisition in the Environmental Internet of Things provides a compromise between generality and complexity. We designed and implemented an open-source, extensible template framework, called *EDAM*, to support environmental timeseries data acquisition, integration and dissemination services, without the prerequisite of a strong computer science background. We enable users to extract datasets and create custom views out of them by defining the desired output format as a template. In this way, users can re-use environmental timeseries data into scientific workflows. *EDAM* also supports opening legacy datasets as services on the web through OGC SOS. Currently, *EDAM* supports data acquisition and integration of timeseries stored in relational databases, files in folder structures, and webpages. The test cases we used to evaluate *EDAM* provided us with insights about its general-purpose nature. The novelty of this approach is that we are not trying to propose another standard, but rather that we have developed a specific language for describing data file structures in a generic way, using templates. Also, such templates are programming language-agnostic so that users of different computer literacy profiles could develop them.

Software availability

Name of software: EDAM (Environmental Data Acquisition Module).

Developers: Argyrios Samourkasidis, Evangelia Papoutsoglou, Ioannis N. Athanasiadis.

Contact: argysamo@gmail.com.

Software required: Any operating system with Python 3.

Program language: Python 3.

License: GNU General Public License.

Software availability: Released via Python's pip package management system. Source code on https://github.com/BigDataWUR/edam.

Acknowledgements

We would like to offer our special thanks to Mr Stavros Foteinopoulos for his valuable and constructive suggestions during the design and implementation of the software. We are grateful to Dr Robert Argent and Dr Andre Zerger from the Australian Bureau of Meteorology for providing us with the historical air quality database. We would also like to express our gratitude to Dr Ming-Yi Tsai and Dr Mark Davey from the Swiss Tropical and Public Health Institute for providing us with the air quality dataset and case study. IA was partially supported by the Wageningen University and Research Strategic Investment Theme programme *Resilience*. Finally, we are grateful to Dr S. Osinga and the three anonymous reviewers for their constructive feedback and comments.

Appendix A. Supplementary data

Supplementary data to this article can be found online at https://doi.org/10.1016/j.envsoft.2018.10.009.

References

Ames, D.P., Horsburgh, J.S., Cao, Y., Kadlec, J., Whiteaker, T., Valentine, D., 2012. Hydrodesktop: Web services-based software for hydrologic data discovery, download, visualization, and analysis. Environ. Model. Software 37, 146–156. https://doi.org/10.1016/j.envsoft.2012.03.013.

Andrae, S., Gruber, G., Hecke, A., Wieser, A., 2009. Sensor web enablement–standards and open source implementations for observation data. In: Schweizer, J., van Herwijnen, A. (Eds.), Proceedings 1st International Snow Science Workshop (ISSW), Birmensdorf, Swiss Federal Institute for Forest, Snow and Landscape Research WSL, Davos, Switzerland.

Athanasiadis, I.N., 2015. Challenges in modelling of environmental semantics. In: Environmental Software Systems. Infrastructures, Services and Applications. Springer, pp. 19–25. https://doi.org/10.1007/978-3-319-15994-2_2.

Athanasiadis, I.N., Mitkas, P.A., 2004. An agent-based intelligent environmental monitoring system. Management of Environmental Quality 15, 238–249. https://doi.org/10.1108/14777830410531216.

M. Bayer, Mako templates for python, http://www.makotemplates.org; accessed 1-August-2017.

Beran, B., Cox, S.J.D., Valentine, D., Zaslavsky, I., McGee, J., 2009. Web services solutions for hydrologic data access and cross-domain interoperability. International Journal on Advances in Intelligent Systems 2 (2&3), 317–324.

Boote, K.J., Porter, C.H., Hargreaves, J., Hoogenboom, G., Thorburn, P., Mutter, C., 2015. Agmip training in multiple crop models and tools. In: HANDBOOK OF CLIMATE CHANGE AND AGROECOSYSTEMS: The Agricultural Model Intercomparison and Improvement Project Integrated Crop and Economic Assessments, Part 2. World Scientific, pp. 393–410.

Botts, M., Robin, A., 2014. OGC SensorML: Model and XML, Encoding Standard 12-000. Open Geospatial Consortium.

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

Broering, A., Stasch, C., Echterhoff, J., 2012. OGC Sensor Observation Service 2.0, Implementation Standard 12-006. Open Geospatial Consortium.

Broytman, O., Croy, T., 2001. Cheetah3, the python-powered template engine. http://cheetahtemplate.org, Accessed date: 1 August 2017.

Cox, S., 2011. Observations and Measurements - XML implementation, Implementation Standard 10-025r1. Open Geospatial Consortium.

Eberle, J., Clausnitzer, S., Hüttich, C., Schmullius, C., 2013. Multi-source data processing middleware for land monitoring within a web-based spatial data infrastructure for

siberia. ISPRS Int. J. Geo-Inf. 2 (3), 553–576. https://doi.org/10.3390/ijgi2030553.
Elag, M.M., Kumar, P., Marini, L., Myers, J.D., Hedstrom, M., Plale, B.A., 2017. Identification and characterization of information-networks in long-tail data collections. Environ. Model. Software 94, 100–111. https://doi.org/10.1016/j.envsoft.2017.03.032.
Python Software Foundation, P.S., 2018. PyPI - the Python Package Index. https://pypi.python.org/pypi, online, Accessed date: 22 February 2018.
Geebelen, K., Michiels, S., Joosen, W., Geebelen, K., Michiels, S., Joosen, W., 2008. Dynamic reconfiguration using template based web service composition. In: Proc. 3rd workshop on Middleware for Service Oriented Computing. MW4SOC '08, ACM, ACM, New York, NY, USA, pp. 49–54. http://doi.acm.org/10.1145/1462802.1462811.
Granell, C., Díaz, L., Gould, M., 2010. Service-oriented applications for environmental models: Reusable geospatial services. Environ. Model. Software 25 (2), 182–198.
Hart, J.K., Martinez, K., 2015. Towards an environmental internet of things. Earth and Space Science 2 (5), 194–200. https://doi.org/10.1002/2014EA000044.
Harth, A., Knoblock, C., Stadtmller, S., Studer, R., Szekely, P., 2013. On-the-fly integration of static and dynamic sources. In: Hartig, O., Sequeda, J., Hogan, A., Matsutsuk, T. (Eds.), Proceedings 4th International Workshop on Consuming Linked Data (COLD2013). vol. 1034 CEUR-WS.org.
Hey, T., Trefethen, A., 2003. e-science and its implications. Phil. Trans. Roy. Soc. Lond.: Mathematical, Physical and Engineering Sciences 361 (1809), 1809–1825.
Holzworth, D.P., Huth, N.I., deVoil, P.G., Zurcher, E.J., Herrmann, N.I., McLean, G., Chenu, K., van Oosterom, E.J., Snow, V., Murphy, C., Moore, A.D., Brown, H., Whish, J.P., Verrall, S., Fainges, J., Bell, L.W., Peake, A.S., Poulton, P.L., Hochman, Z., Thorburn, P.J., Gaydon, D.S., Dalgliesh, N.P., Rodriguez, D., Cox, H., Chapman, S., Doherty, A., Teixeira, E., Sharp, J., Cichota, R., Vogeler, I., Li, F.Y., Wang, E., Hammer, G.L., Robertson, M.J., Dimes, J.P., Whitbread, A.M., Hunt, J., van Rees, H., McClelland, T., Carberry, P.S., Hargreaves, J.N., MacLeod, N., McDonald, C., Harsdorf, J., Wedgwood, S., Keating, B.A., 2014. Apsim evolution towards a new generation of agricultural systems simulation. Environ. Model. Software 62, 327–350. https://doi.org/10.1016/j.envsoft.2014.07.009.
Holzworth, D.P., Snow, V., Janssen, S., Athanasiadis, I.N., Donatelli, M., Hoogenboom, G., White, J.W., Thorburn, P., 2015. Agricultural production systems modelling and software: Current status and future prospects. Environ. Model. Software 72, 276–286. https://doi.org/10.1016/j.envsoft.2014.12.013.
Horita, F.E., de Albuquerque, J.P., Degrossi, L.C., Mendiondo, E.M., Ueyama, J., 2015. Development of a spatial decision support system for flood risk management in Brazil that combines volunteered geographic information with wireless sensor networks. Comput. Geosci. 80, 84–94. https://doi.org/10.1016/j.cageo.2015.04.001. https://doi.org/10.1016/j.cageo.2015.04.001.
Horsburgh, J.S., Tarboton, D.G., 2007. CUAHSI ODM streaming data loader design specifications, Design Specification Document 1.1. Consortium of Universities for the Advancement of Hydrologic Science (CUAHSI. https://www.cuahsi.org/uploads/pages/img/ODM_SDL_Design_Specifications_(2).pdf.
Horsburgh, J.S., Tarboton, D.G., Maidment, D.R., Zaslavsky, I., 2008. A relational model for environmental and water resources data. Water Resour. Res. 44 (5). http://doi.org/10.1029/2007WR006392.
Horsburgh, J.S., Tarboton, D.G., Piasecki, M., Maidment, D.R., Zaslavsky, I., Valentine, D., Whitenack, T., 2009. An integrated system for publishing environmental observations data. Environ. Model. Software 24 (8), 879–888. https://doi.org/10.1016/j.envsoft.2009.01.002.
Horsburgh, J.S., Tarboton, D.G., Maidment, D.R., Zaslavsky, I., 2011. Components of an environmental observatory information system. Comput. Geosci. 37 (2), 207–218. https://doi.org/10.1016/j.cageo.2010.07.003.
Ihaka, R., Gentleman, R., 1996. R: A language for data analysis and graphics. J. Comput. Graph Stat. 5 (3), 299–314. https://doi.org/10.1080/10618600.1996.10474713. arXiv.
IEEE International Conference on eScience, 2018. What is eScience? https://escience-conference.org, Accessed date: 26 July 2018.
Jones, A.S., Horsburgh, J.S., Reeder, S.L., Ramírez, M., Caraballo, J., 2015. A data management and publication workflow for a large-scale, heterogeneous sensor network. Environ. Monit. Assess. 187 (6), 1–19. https://doi.org/10.1007/s10661-015-4594-3.
Kandel, S., Paepcke, A., Hellerstein, J., Heer, J., 2011. Wrangler: Interactive visual specification of data transformation scripts. In: Proceedings of SIGCHI Conference on Human Factors in Computing Systems. ACM, pp. 3363–3372. https://doi.org/10.1145/1978942.1979444.
Keating, B., Carberry, P., Hammer, G., Probert, M., Robertson, M., Holzworth, D., Huth, N., Hargreaves, J., Meinke, H., Hochman, Z., McLean, G., Verburg, K., Snow, V., Dimes, J., Silburn, M., Wang, E., Brown, S., Bristow, K., Asseng, S., Chapman, S., McCown, R., Freebairn, D., Smith, C., 2003. An overview of apsim, a model designed for farming systems simulation. Eur. J. Agron. 18 (3), 267–288. modelling Cropping Systems: Science, Software and Applications. https://doi.org/10.1016/S1161-0301(02)00108-9.
Klein, T., Samourkasidis, A., Athanasiadis, I.N., Bellocchi, G., Calanca, P., 2017. webxtreme: R-based web tool for calculating agroclimatic indices of extreme events. Comput. Electron. Agric. 136, 111–116. https://doi.org/10.1016/j.compag.2017.03.002.
Mason, S.J., Cleveland, S.B., Llovet, P., Izurieta, C., Poole, G.C., 2014. A centralized tool for managing, archiving, and serving point-in-time data in ecological research laboratories. Environ. Model. Software 51, 59–69.
McFerren, G., Hohls, D., Fleming, G., Sutton, T., 2009. Evaluating sensor observation service implementations. Proceedings International Geoscience and Remote Sensing Symposium (IGARSS), vol. 5. IEEE, pp. 363–366. https://doi.org/10.1109/IGARSS.2009.5417655.
McKinney, W., 2011. Pandas: a foundational python library for data analysis and statistics. In: Workshop Python for High Performance and Scientific Computing (SC11). ACM, New York, NY, USA.
Negru, C., Pop, F., Mocanu, M., Cristea, V., 2016. A Unified Approach to Data Modeling and Management in Big Data Era. Springer International Publishing, pp. 95–116. Ch. Data Science and Big Data Computing. https://doi.org/10.1007/978-3-319-31861-5_5.
Papoutsoglou, E., Samourkasidis, A., Tsai, M.-Y., Davey, M., Ineichen, A., Eeftens, M., Athanasiadis, I.N., 2015. Towards an air pollution health study data management system-a case study from a smoky swiss railway. In: Johannsen, V.K., Jensen, S., Wohlgemuth, V., Preist, C., Eriksson, E. (Eds.), Adjunct Proc. 29th EnviroInfo and 3rd ICT4S Conference. University of Copenhagen, 978-87-7903-712-0, pp. 65–74.
Peckham, S.D., Goodall, J.L., 2013. Driving plug-and-play models with data from web services: A demonstration of interoperability between CSDMS and CUAHSI-HIS. Comput. Geosci. 53, 154–161. modeling for Environmental Change. https://doi.org/10.1016/j.cageo.2012.04.019.
Raspberry Pi Foundation, 2018. Raspberry Pi Model B. https://www.raspberrypi.org/products/raspberry-pi-2-model-b/, Accessed date: 12 July 2018.
Porter, C.H., Villalobos, C., Holzworth, D., Nelson, R., White, J.W., Athanasiadis, I.N., Janssen, S., Ripoche, D., Cufi, J., Raes, D., et al., 2014. Harmonization and translation of crop modeling data to ensure interoperability. Environ. Model. Software 62, 495–508. https://doi.org/10.1016/j.envsoft.2014.09.004.
Ronacher, A., 2008. Jinja2. http://jinja.pocoo.org, Accessed date: 12 December 2016.
Ronacher, A., 2010. Flask. http://flask.pocoo.org, Accessed date: 12 December 2016.
Samourkasidis, A., Athanasiadis, I.N., 2017. A miniature data repository on a Raspberry Pi. Electronics 6 (1). http://doi.org/10.3390/electronics6010001.
Samourkasids, A., Athanasiadis, I.N., Papoutsoglou, E., 2018. Edam software. https://github.com/BigDataWUR/EDAM, Accessed date: 22 February 2018.
Stadtmüller, S., Speiser, S., Harth, A., Studer, R., 2013. Data-fu: a language and an interpreter for interaction with read/write linked data. In: Proceedings 22nd International Conference on World Wide Web. ACM, pp. 1225–1236.
Swain, N.R., Christensen, S.D., Snow, A.D., Dolder, H., Espinoza-Dvalos, G., Goharian, E., Jones, N.L., Nelson, E.J., Ames, D.P., Burian, S.J., 2016. A new open source platform for lowering the barrier for environmental web app development. Environ. Model. Software 85, 11–26. https://doi.org/10.1016/j.envsoft.2016.08.003 http://www.sciencedirect.com/science/article/pii/S1364815216304625.
Taylor, P., 2014. OGC WaterML 2.0: Part 1- Timeseries, Implementation Standard 10-126r4. Open Geospatial Consortium.
Terrizzano, I., Schwarz, P.M., Roth, M., Colino, J.E., 2015. Data wrangling: The challenging journey from the wild to the lake. In: Proceeding 7th Biennial Conference on Innovative Data Systems Research (CIDR). Online Proceedings, Asilomar, California, USA..
Van Rossum, G., Drake, F.L., 2003. Python language reference manual. Network Theory United Kingdom.
Wikipedia contributors, List of tz database time zones — Wikipedia, the free encyclopedia. https://en.wikipedia.org/w/index.php?title=List_of_tz_database_time_zones&oldid=851163045, Accessed date: 7 August 2018.
Wilkinson, M.D., Dumontier, M., Aalbersberg, I.J., Appleton, G., Axton, M., Baak, A., Blomberg, N., Boiten, J.-W., da Silva Santos, L.B., Bourne, P.E., et al., 2016. The FAIR guiding principles for scientific data management and stewardship. Sci. Data 3 (160018). http://doi.org/10.1038/sdata.2016.18.
Woodard, J., 2016. Big data and Ag-Analytics: An open source, open data platform for agricultural & environmental finance, insurance, and risk. Agric. Finance Rev. 76 (1), 15–26. https://doi.org/10.1108/afr-03-2016-0018.